Oracle® E-Business Suite

Oracle Application Framework Developer's Guide
Release 12.2.10

September 2020

Table of Contents

Oracle Application Framework Developer's Guide 7

Chapter 1: Getting Started 7

Preface 7

Introduction to OA Framework 9

Setting Up Your Development Environment 14

Building and Running 'Hello, World! 19

OA Framework Runtime Utilities 40

Chapter 2: OA Framework Essentials 42

JSP Application Primer 42

Anatomy of an OA Framework Page 49

OA Framework State Management 61

Chapter 3: Basics of Building an OA Framework Application 71

Implementing the Model 71

Implementing the View 94

Implementing the Controller 108

Error Handling 117

Creating Attribute Sets 125

Internationalization 130

Files in a Typical OA Framework Application 137

Chapter 4: Implementing Specific UI Features 140

Accordion 140

Advanced Tables 148

Attachments 177

Auto-Repeating Layouts 210

Bound Values 213

Branding 219

Bulleted List 223

Buttons (Action/Navigation) 225

Buttons (Global) 231

Buttons (Links) 242

Charts and Graphs 244

Classic Tables 271

Color Picker 295

Concurrent Processing 297

Content Containers 301

Contextual Information 303
Custom HTML 305
Data Export/Import 307
Date Picker 313
Declarative Page Flow 317
Default Favorites 326
Dialog Pages 334
Dynamic User Interface 337
EL (Expression Language) Support 351
File Upload/Download 354
Flexfields 357
Gesture Support 380
Headers and Subheaders 381
HGrid 386
Hide/Show 396
Hide/Show SubTab Layout 402
Home Page 405
Images in Your Pages 411
Include Content (URL and Servlet) 416
Infotile 418
Inline Messaging, Tips, Hints and Bubble Text 423
Instruction Text 427
Keyboard Shortcuts 429
Layered Layout 432
List of Values (LOV) 435
List Search 451
Locator Element: Breadcrumbs 488
Locator Element: Page/Record Navigation 496
Locator Element: Train 501
Masonry Layout 504
Menu Component 510
Message Box 514
Message Rich Text Editor 517
Mobile Applications 524
Notifications (Workflow Worklist) 535
Page Access Tracking 537
Page Contents Bottom Line (the 'Ski') 539

Page Footer 540
Page Layout (How to Place Content) 542
Page Security 557
Page Stamps 563
Panel Splitter 566
Pop-Ups 569
Printable Page 576
Processing Page 578
Quick Links 581
Rating Bar 583
Record History 586
Related Links / Shortcuts 588
RESTful Service Interface 590
Rich Content Container 599
Rich Interactions 603
Save Model ('Warn About Changes') 626
Search 629
Separator Line 664
Shuttle 665
Standard Web Widgets 669
Submitting the Form 684
SubTab Navigation 688
Switchers 694
Tabs / Navigation 696
Tree 715
Chapter 5: Implementing Server-Side Features 720
Java Entity Objects 720
PL/SQL Entity Objects 739
Business Service Objects Architecture 747
Business Service Objects in Detail 754
Business Service Object Methods in Detail 762
Oracle Application Web Services 770
View Objects in Detail 773
Application Modules in Detail 784
Service Frequently Asked Questions (FAQ) 788
Testing Business Service Objects 790
Deploying Services 800

Introduction to Service-Oriented Architecture and Services 801
Using Services 803
Entity Object and View Object Attribute Setters 810
Chapter 6: Advanced OA Framework Development Topics 815
Supporting the Browser Back Button 815
Browser Back Button Support Use Cases 827
Advanced Java Entity Object Development Topics 844
Controlling UIX Rendering Output 849
OA Framework and AOL/J Caching 854
Application Module and Connection Pooling 855
Advanced View Object Development Topics 865
Accelerated Validation 874
Oracle Applications Java Caching Framework Developer's Guide, Release 12 884
Chapter 7: Testing and Debugging 909
Discovering Page, Technology Stack and Session Information 914
Debugging OA Framework Applications 909
Logging 922
Inspecting the MDS Repository Content 925
Testing OA Framework Applications 931
Oracle Applications Framework Troubleshooting 936
Chapter 8: OA Framework File Standards 937
OA Framework File Standards (Naming, Package Structure and Standard Content) 937
OA Framework Model Coding Standards 952
OA Framework View Coding Standards 966
OA Framework Controller Coding Standards 976
Oracle Applications Java Coding Standards 984
Chapter 9: Extending and Deploying OA Framework Applications 989
Deploying Customer Extensions 989
Extending OA Framework Applications 993
Accessing an OA Framework Page From an External Application 999
Configurable Pages 1000
Creating an End-User Personalizable Page 1010
Developer Information for Administrator-Level Personalizations 1011
Chapter 10: Integrating with Other Products 1014
Portlets 1014
Integrating with Oracle ADF 1018
Embedding OBIEE Analytics in an OA Framework Page 1019

JTT/OA Framework Interoperability 1020
Form/OA Framework Page Integration 1024
Appendices 1027
Appendix A: Summary of OA Component Properties 1027
Appendix B: Oracle Applications Framework Profile Options 1028
Appendix C: OA Framework ToolBox Technical Reference Manual (TRM) 1043
Appendix D: Oracle Applications Framework URL Parameters 1050
Appendix E: OA Framework Development FAQ 1055
Appendix F: OA Framework Known Issues 1056
Appendix G: Oracle Applications Framework Extensible Regions 1062
Appendix H: Oracle JDeveloper White Paper 1063
Appendix I: Recommended Browsers for Oracle E-Business Suite Release 12 1071
Appendix J: Oracle Application Framework Support Guidelines for Customers 1072
Glossary 1075
OA Framework Glossary 1075
Oracle Copyright Notice 1079

Chapter 1: Getting Started

Preface

This book describes how to set up your development environment, and then build, test and deploy Oracle E-Business Suite OA Framework applications. It also includes the coding standards followed by the Oracle E-Business Suite Development staff, instructions for creating pages that comply with the Oracle Browser Look and Feel (BLAF) UI Guidelines, and information on extending the products shipped by Oracle E-Business Suite Development.

Note: Some of the screenshots used in this book were captured on Release 11.5.10, which used the BLAF Look-and-Feel. Although the colors and interface elements of these images consequently have the 11.5.10 appearance, the functionality they illustrate also applies to Release 12 and the Oracle Look-and-Feel.

Contents

- Audience
- Related Publications
- Typographic Conventions
- Send Us Your Comments

Audience

This documentation is written for the application developer and assumes familiarity with Java and SQL.

Related Publications

Additional Oracle JDeveloper 10*g* helpsets that apply to OA Framework application development include:

- OA Framework ToolBox Tutorial
- OA Extension Component Reference
- Getting Started with the OA Extension
- Getting Started with JDeveloper
- Developing Business Components

As an application designer, you should also be familiar with the Oracle Browser Look and Feel (BLAF) UI Guidelines and the documentation for the Oracle Database.

Typographic Conventions

This manual uses the following typographic conventions to distinguish important elements from the body of the manual.

Command and Example Syntax

Commands and examples appear in a monotype font, as follows:

Syntax:

```
OAPageContext.getParameter("<parameterName>");
```

Example:

```
/*
** Creates a SupplierEOImpl entity object and a corresponding row in the SuppliersVO.
*/
public void createSupplier()
{
  OAViewObject vo = getSuppliersVO();
  vo.insertRow(vo.createRow());
}
```

Command and example syntax adhere to the following conventions:

Convention	Explanation
`plain monotype`	Used for code fragments and examples.
`< Italic monotype in angle brackets >`	Indicates developer-supplied values.

`...`	An ellipsis indicates that the actual code extends beyond the example shown.
`/*` `*/`	A C-style comment.
`/**` `*/`	A Javadoc comment.
`//`	A Java comment.
Indentation	Oracle standard indentation helps to show code structure.

Send Us Your Comments

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this manual. Your input is an important part of the information used for revisions.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most?

If you find any errors or have any other suggestions for improvement, please indicate the document title, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: `appsdoc_us@oracle.com`
- FAX: (650) 506-7200 Attn: Oracle E-Business Suite Documentation Manager
- Postal service:

  Oracle Corporation
  Oracle E-Business Suite Documentation Manager
  500 Oracle Parkway
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Introduction to OA Framework

This document provides an overview of OA Framework and discusses:

- Architecture
- Key features
- Summary

Overview

Oracle Application Framework (OA Framework) is the Oracle E-Business Suite development and deployment platform for HTML-based business applications. OA Framework consists of a set of middle-tier runtime services and a design-time extension to Oracle JDeveloper called Oracle E-Business Suite Extension (OA Extension).

During the first few years after the Internet evolution, the software industry witnessed an influx of rapidly changing technologies. These technologies matured, yet there are still a myriad of low-level and complex technologies that are hard to learn and implement. Under these circumstances, OA Framework has emerged as an integrated platform for developing and deploying Oracle E-Business Suite HTML-based applications, leveraging technological advances without taking on associated complexity. Since its inception, OA Framework embraces the following principles:

End User Productivity

The shift from client-server to multi-tier deployments comes with many cost savings, but not without compromise. HTML-based applications started out very much like old mainframe terminals; actions on the client side resulted in a round trip to the middle tier.

Over time, user interface interactivity improved. OA Framework has always kept user interface interactivity a top priority with features such as partial page rendering (PPR), hot keys, smart choice lists and auto-completion of fields with lists of values. In addition, Oracle focuses a wealth of resources and expertise on user behavior and psychology, to develop a set of user interface layout and interaction standards, commonly known as the LAF (Look-And-Feel) guidelines. BLAF is the default look and feel that all OA Framework applications assume, but can be personalized in many ways to meet customer branding and style requirements. OA Framework's implementation of BLAF standards yields a consistent user experience and further enhances user productivity.

Enterprise-Grade Performance and Scalability

OA Framework has aggressive performance and scalability targets. Most Oracle E-Business Suite application pages have sub-second response times to most user interactions. It takes a bit longer the first time a page is accessed within the same Java Virtual Machine, but thereafter, most of the commonly needed information (such as user information) is cached in the middle tier, allowing faster response. Resources are conserved through a number of resource pooling mechanisms and the swapping of idle resource data between memory and database.

Developer Productivity

OA Framework is designed around the simple Model-View-Controller (MVC) architecture. To shield application developers from costs associated with the rapidly changing technological landscape, Oracle has adopted a declarative flavor of the MVC architecture. Key building blocks of an application are defined in a descriptive manner using a simple JDeveloper user interface and then saved in an industry standard XML format. Oracle extends access and benefits of the OA Framework development environment to all Oracle E-Business Suite customers and partners. Customers and partners can leverage the proven OA Framework technology to add extensions to their Oracle E-Business Suite applications.

Application Customizability

Oracle is able to exploit its twenty plus years of experience in building and deploying business applications, to architect OA Framework with durable and economical customizations. Oracle has kept that goal in focus and produced a very compelling solution with plenty of flexibility to tailor the user interface (look-and-feel) and business logic. Thanks to the declarative and object oriented nature of OA Framework, application personalization and extensibility is readily available at a fraction of the industry startup cost and at a very minimal maintenance cost, if any.

Open Standards

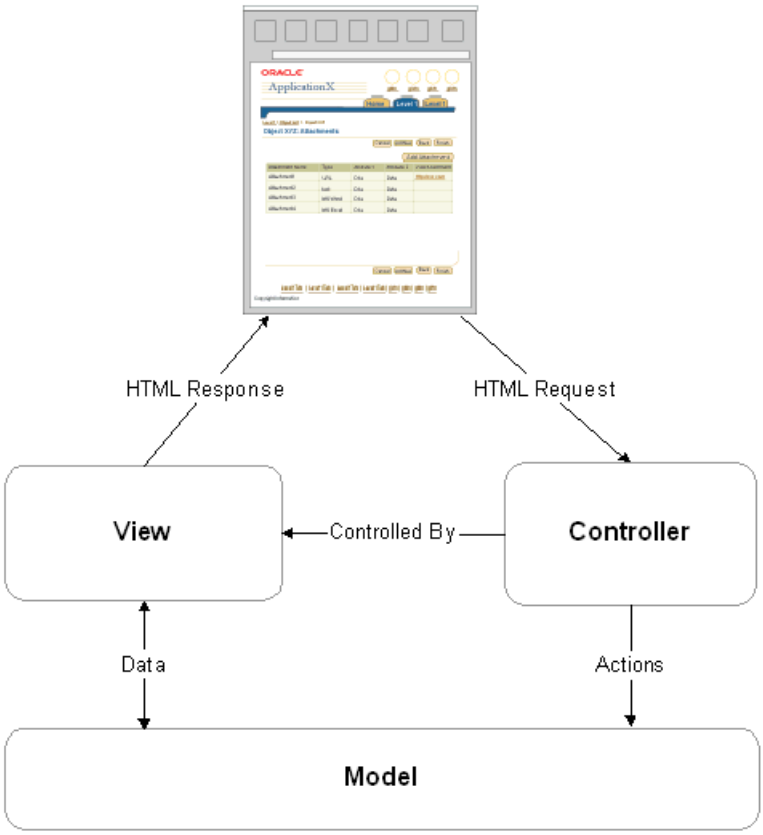
Oracle continues to be a champion of industry standards and an active participant in the development of several emerging standards. OA Framework technologies have driven several industry standards and have adopted several others as they were published. Several Oracle technology architects are active members on a number of standards drafting committees. OA Framework is J2EE based and features several industry standards such as XML, HTML, Java, JSP, SQL and Web Services.

Architecture

OA Framework is based on the industry-standard J2EE MVC design pattern. Developers manipulate the application's metadata using

Oracle JDeveloper OA Extension, while OA Framework uses the most efficient manner to execute the application. The MVC architecture is a component-based design pattern with clean interfaces between the Model, View, and Controller. The Model is where the application implements its business logic. The View is where the application implements its user interface and the Controller is where the application handles user interaction and directs business flow.

Figure 1: OA Framework MVC architecture.



OA Extension offers the following design time tools:

- UML tools to model and generate business logic.
- Guided user interface to lay out client user interfaces.
- Code generation for Controller classes.

The OA Framework Model is implemented using Oracle Business Components for Java (BC4J). BC4J provides optimized, ready-to-use implementations of the J2EE design patterns that developers otherwise would have to code, debug, and test by hand. By leveraging BC4J's combination of tested code and productivity tools inside the Oracle JDeveloper IDE, development teams can focus immediately and only, on writing business logic and user interfaces instead of on designing, coding, and debugging handcrafted application "plumbing" code.

OA Framework View is implemented using UI XML (UIX). UIX uses XML to describe the components and hierarchy that make up an application page. UIX also provides runtime capabilities to translate that metadata into HTML output so that it can be shown on a Browser or a mobile device. The metadata used to describe the UI is loaded into a database repository, called Meta Data Services (MDS), at deployment time and optionally at design time as well.

The OA Controller, which is a pure Java class implementation, handles user and application-driven interactions. Simple page flows (such as a 2-step transaction) are implemented directly into the Controller object; others are implemented using Oracle Workflow.

Key Features

This section the following key features of OA Framework:

- Integrated development environment
- Durable personalizations and extensions
- Consistent and compelling user interface
- User interface interactivity
- Object oriented reuse
- Portal interoperability

- Built-in security
- Deployment environment

Integrated Development Environment

Oracle JDeveloper with OA Extension (OA Extension) is a world-class J2EE-based integrated development environment. Oracle customers and third party consultants have access to the same tools used by Oracle E-Business Suite developers to build complementary applications as well as extend the Oracle E-Business Suite applications. OA Extension provides features such as easy-to-use wizards, a hierarchy navigator, and a property sheet. These features enable developers to populate the metadata for declarative application business logic and user interfaces. JDeveloper offers a wealth of productivity tools such as the UML modeler, code coach, integrated debugger, local testing environment and documentation generator.

With the OA Extension software comes a wealth of documentation and learning aids including a Developer's Guide, Javadoc, Online Help, a Sample Library and a rich set of Tutorials.

Durable Personalizations and Extensions

Personalization is about declaratively tailoring the UI look-and-feel, layout or visibility of page content to suit a business need or a user preference. Examples of personalization include:

- Tailoring the color scheme of the UI.
- Tailoring the order in which table columns are displayed.
- Tailoring a query result.

Extensibility is about extending the functionality of an application beyond what can be done through personalization. Examples of extensibility include:

- Adding new functional flows.
- Extending or overriding existing functional flows.
- Extending or overriding existing business logic.

OA Framework is designed with durable personalization and extensibility capabilities, achieved through the declarative architecture and the underlying object oriented implementation.

Declarative UI component definitions are stored in the form of metadata in a database repository. Personalizations are translated into offsets from the base metadata definition and stored separately. At runtime, all applicable personalization metadata is loaded from the repository and layered over the base metadata definition to produce the net effect. Product upgrades and patching only affect the base metadata definition so that customer personalizations are preserved and continue to function properly.

Personalizations can be implemented at several levels by one of three authors: application developer, application administrator and end user.

An end-user can create a personalization to be applied to specific user interface components that is only visible in the context of that authoring user. For example, an end user may save an employee search result sorted by manager and hide the employee's date of birth column. Once this personalized view is saved under a given name, the user can retrieve that view again in the future by that name.

Application administrators and application developers have the flexibility to tailor the user experience at several levels. They can author personalizations that affect all users, users of a particular locale, users of a particular organization, users with a particular role and in the context of a particular function. Several levels can apply at the same time with a predetermined precedence order that yields a very personalized user experience.

Using a combination of OA Extension wizards and built-in personalization screens, several user interface and business logic extensions are made possible at a minimal cost to development with little-to-no maintenance cost. In addition, Oracle E-Business Suite customers continue to enjoy the extensibility features offered by Oracle Flexfields, Oracle Workflow and Business Events.

Consistent and Compelling User Interface

OA Framework offers developers a wide range of user interface components that make the building of applications into a more assembly process, freeing developers from the repetitive composition of common user interface constructs. Moreover, OA Framework's declarative approach to building application user interfaces frees developers from the need to learn a vast array of changing technologies, while offering end users a consistent application look and experience. OA Framework user interface components range from simple widgets such as buttons and fields to compound components such as tables-in-tables and hierarchical grids.

User Interface Interactivity

OA Framework is always exploring the technology frontiers to enrich the interactivity of HTML-based user interfaces. Along those lines, OA Framework provides several features:

Partial Page Rendering (PPR)

PPR is a means by which designated parts of a page, rather than the whole page, is refreshed when the user performs certain actions. OA Framework supports PPR on actions such as: table record-set navigation, table sorting, table column totaling, adding a row to a table, row-level and cell-level detail disclosure, toggling the visibility of a Hide/Show component, populating a LOV, subtab navigation, Gantt chart refreshing and descriptive Flexfields context switching. Moreover, developers can declaratively enable PPR events on several components. For example, a developer can:

- Configure the selection of a poplist to cause related fields to render, be updatable, be required or be disabled based on the selected value.
- Configure the value change of a text field to set related field values (for example, if you set a Supplier value and tab to the next field, the dependent Supplier Site defaults automatically).
- Configure the selection of a master table's record to automatically query and display related rows in a detail table.

Accelerator (Hot) Keys

OA Framework supports mnemonic accelerator keys for selected buttons and enables developers to assign numeric access keys to product specific user actions.

Enhanced Save Model

OA Framework provides a default implementation to warn users when they are about to lose changes such as when they click on a link that takes them outside the context of the current transaction. Developers can override the default behavior on a component-by-component basis.

Smart Poplist

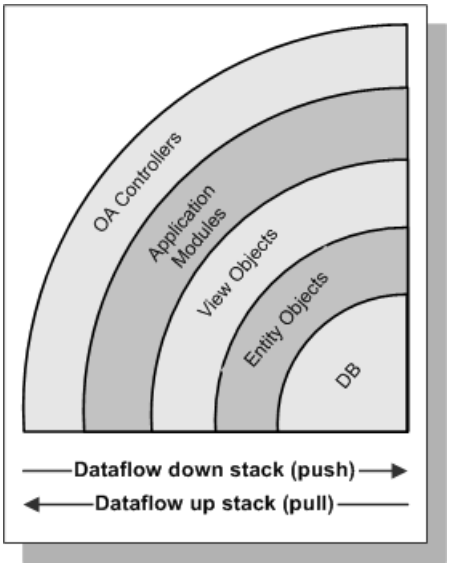
OA Framework supports a personalizable hybrid between a static poplist and a searchable list of values. The poplist includes the most popular values a user uses to populate a particular attribute. The user can personalize the values that show up in the poplist by picking new values from a list of values. Moreover, the user can personalize the order in which values are listed in the poplist as well as remove less popular values. This feature is also referred to as a LOV Choicelist.

LOV Auto Completion

Lists of values (LOVs) are used when the list of possible values is long and the user may want to conduct a search before picking a value. In some business scenarios, especially with clerical jobs, the user uses a small set of values or may find it faster to type a partial value. If the user enters a partial value in a field that is associated with an LOV, OA Framework conducts a search before bringing up the LOV window. If the search leads to a unique record, OA Framework completes the rest of value for the unique record and saves the user from having to use the LOV window.

Object Oriented Reuse

OA Framework applications can be abstracted into a series of concentric layers, like an onion. The core layer represents the database and the surface layer represents the application pages. In between is a number of business logic and user interface layers. This layering allows for generic code and components to be implemented at the inner layers to maximize their reuse across the outer layers. For example, attribute validation is implemented at the Entity Object (a BC4J object-oriented representation of a database table in the middle tier) level. All application pages that provide the user with the ability to populate or update the value of the subject attribute would receive attribute validation for free through the underlying entity object. On the user-interface side, reusable components can be saved as shared regions in the metadata services (MDS) repository and reused across several pages. An administrator can choose to personalize the shared region such that the personalization impacts all instances of the same region across pages or personalize the shared region only in the context of the current page.



Portal Interoperability

OA Framework offers developers a simple approach to publishing OA Framework components (commonly known as regions) as portlets in Oracle WebCenter Portal, Oracle Portal or any WSRP-compliant portal. Portal applications provides you with a common, integrated starting point for accessing all your data. OA Framework allows you to personalize the application region that is displayed as a portlet. Any personalizations you make to that portlet region appear only when you display that region from the same portlet.

Built-in Security

HTML-based applications offer great user and administrator convenience, but special care must be taken to ensure that these applications are secure. Developing HTML applications that are truly unbreakable is very difficult, historically requiring application developers to also be security experts. In fact, most application developers are not security experts, and they should not need to be. It is the responsibility of the application framework to ensure that HTML transactions are authorized, private, and free from tampering. OA Framework provides built in protection against known HTML hacking strategies, leaving the application developer free to concentrate on application functionality. Also, since UI components are defined in metadata rather than in code, the security protection offered by OA Framework can be advanced to keep up with the state of the art, without requiring applications to be rewritten.

Deployment Environment

OA Framework applications are deployed using standard Oracle 10*g* AS / Apache and Oracle10*g* Database servers. The database and application tiers can be deployed on several platforms including Linux, UNIX and Windows. Application pages can be rendered on most modern browsers, as per My Oracle Support Knowledge Document 389422.1, *Recommended Browsers for Oracle E-Business Suite Release 12*.

Summary

Based on the Model-View-Controller (MVC) architecture, OA Framework lets application developers focus on the business requirements rather than on the underlying technologies. By using declarative and guided-coding techniques, OA Framework allows application developers who are not necessarily J2EE experts to quickly become productive. OA Framework-based applications offer a highly consistent user experience with the highest levels of user interactivity without a client footprint. Applications are optimized for sub-second response to most user interactions and competitive scalability trends. OA Framework exploits its declarative and object-oriented architecture to offer the most durable personalization and extensibility capabilities on the market, at a fraction of the cost. OA Framework features translate to lower costs of ownership, better user experience and competitive deployments.

Setting Up Your Development Environment

This document describes how to configure and test an OA Framework Release 12 development environment for the following use cases:

- Customer, Consultant or Support Representative Using JDeveloper on Windows.
- Customer, Consultant or Support Representative Using JDeveloper on Linux.

Note: Oracle employees who have installed Oracle JDeveloper OA Extension and want to set up and test this environment should select the Customer link.

Customer, Consultant or Support Representative Using JDeveloper on Windows

This section contains instructions to configure and test OA Framework if you are a customer, consultant or support representative using JDeveloper on Windows. It provides an overview of the directory structure and discusses how to:

- Configure the JDEV_USER_HOME environment variable.
- Obtain a database connection file.
- Create a desktop shortcut to JDeveloper.
- Assign Toolbox responsibilities.
- Launch JDeveloper and configure the database connection and user.
- Test the setup.

Overview

These instructions assume you have successfully installed the JDeveloper OA Extension zip file which creates the following directory structure on your drive of choice.

Directory	Description
Tip: To open any of the documentation in the **jdevdoc** directories, open the `jdevdoc\index.htm`.	
`jdevdoc\javadoc\fwk`	Includes OA Framework Javadoc.
`jdevdoc\javadoc\aolj`	Includes AOL/J Javadoc.
`jdevdoc\javadoc\bc4j`	Includes BC4J Javadoc.
`jdevdoc\javadoc\uix`	Includes UIX Javadoc.
`jdevdoc\toolbox`	Includes OA Framework ToolBox Tutorial lesson/lab documentation.
`jdevdoc\devguide`	Includes the OA Framework Developer's Guide.
`jdevbin\`	Includes an extended version of the Oracle JDeveloper 10*g* executable and OA Framework class libraries.
`jdevhome\`	Includes the OA Framework ToolBox Tutorial source and developer working area.

Task 1: Configuring the JDEV_USER_HOME Environment Variable

Warning: This is a requirement for JDeveloper. Do *not* skip this task.

Configure the `JDEV_USER_HOME` environment variable using **Windows XP** or **Windows 2000**:

1. Go to your desktop and select **My Computer**, right-click and select **Properties**.
2. On the **System Properties** dialog, select the **Advanced** tab.
3. On the **Advanced** page, select the **Environment Variables...** button.
4. On the **Environment Variables** dialog, select the **New...** button from the **User variables for <username>** box.
5. On the **New User Variable** dialog, enter `JDEV_USER_HOME` in the **Variable Name** field. Set the **Variable Value** field to `<drive>:\jdevhome\<project>` where `<drive>` is the drive where you installed the JDeveloper OA Extension zip file and `<project>` is the subdirectory where your project will reside. For example: `c:\jdevhome\jdev`.
6. Select **OK** in each of the dialogs you opened to save the new user environment variable.

Note: In order to keep projects separate using the same JDeveloper instance, you can modify the `JDEV_USER_HOME` environment variable before launching JDeveloper each time to point to a different location for a different project. For example,

`c:\jdevhome\jdevproject1` or `c:\jdevhome\jdevproject2`.

Warning: The variable value should not contain a leading space before the drive name. If it does, your environment will not work properly.

Task 2: Obtaining a Database Connection File

Obtain the FND database connection (`.dbc`) file from the system administrator who installed the OA Framework database where you want to do your development. Place this file in the `<JDEV_USER_HOME>\dbc_files\secure` directory.

Task 3: Creating a Desktop Shortcut to JDeveloper

To facilitate launching JDeveloper, create a desktop shortcut to jdevbin\jdev\bin\jdevw.exe.

Alternatively, if you are working on multiple projects, you can create a separate batch file (`jdev.cmd` or `jdev.bat`) to set the `JDEV_USER_HOME` environment variable for each of your projects. The example below illustrates what the content of each batch file should look like:

```
@echo off

setlocal
set JDEV_HOME=D:\jdeveloper\1213_x5_8919491
set JDEV_USER_HOME=D:\jdevhome\<project>
start "Jdeveloper" %JDEV_HOME%\jdev\bin\jdevw.exe
endlocal
```

Modify the content of the batch file so that `JDEV_HOME` is set to the path of your JDeveloper installation and `JDEV_USER_HOME` is set to your local project directory and use case.

Note: It is convenient to maintain multiple batch scripts that each point to a different JDeveloper project and create separate shortcuts for each of these batch scripts/projects.

Task 4: Assigning ToolBox Responsibilities

If you have not already done so as part of your installation verification, assign the following ToolBox Tutorial responsibilities to a test user. Refer to the *Oracle E-Business Suite Security Guide* for information about creating users and assigning responsibilities to users.

Note: Use an existing user in your system or create a new test user.

- OA Framework ToolBox Tutorial (responsibility key is FWK_TBX_TUTORIAL).
- OA Framework ToolBox Tutorial Labs (responsibility key is FWK_TOOLBOX_TUTORIAL_LABS).

Task 5: Launching JDeveloper and Configuring the Database Connection and User

Use this procedure to launch JDeveloper and configure the database connection and user:

1. Select the desktop shortcut created in Task 3 to launch Oracle JDeveloper.
2. Select **File > Open** from the main menu, then navigate to <JDEV_USER_HOME>\myprojects. Open the OA Framework ToolBox Tutorial workspace file (toolbox.jws).
3. Expand the toolbox.jws in the JDeveloper System Navigator, to display its contents. Select the `Tutorial.jpr` project, then select **Project > Project Settings**.
4. Expand the `Oracle Applications` node, which is In the **Project Settings** dialog, and select **Runtime Connection**.
5. Locate the DBC file that you saved in Task 2 by using the **Browse...** button, which is In the **Connection** box. The file should be in the `<JDEV_USER_HOME>\dbc_files\secure` directory.
6. Specify the **User Name** and **Password** for the test user. This is the user that you assigned the ToolBox responsibilities to in Task 4. Select **OK**.
7. Repeat Steps 3 - 6 for the `LabSolutions.jpr` project.
8. Expand the **Connections** node in the JDeveloper System Navigator and then expand the **Database** node. Right-click on the **Database** node and select **New Connection...** to open the **Connection Wizard**. Follow the JDeveloper instructions to define a new database connection for the Oracle E-Business Suite database identified by the DBC file you selected above.
9. Select the `Tutorial.jpr` project In the System Navigator. Right-click and select **Edit Business Components Project...**.

10. Select the **Connection** option in the **Business Components Project Wizard** and set the **Connection Name** to the connection you just defined. Select **OK** to save your changes.

11. Repeat steps 8 - 10 for the LabSolutions.jpr project.

- Confirm the environment encoding of your JDeveloper if it is appropriately set.
  To see the environment encoding, see your preferences of JDeveloper.
  If Encoding is not set to "UTF-8", set it to "UTF-8".

Task 7: Test your Setup

Perform the following steps to test your setup:

Tip: If you want pages to look as they do in the OA Framework ToolBox Tutorial / Sample Librar, use Internet Explorer 5.0+ as your default browser.

1. Open the `toolbox.jws` workspace in the JDeveloper Navigator using the instructions in Task 5 above.

2. Go to the System Navigator, select `toolbox.jws` and then select **Project > Rebuild toolbox.jws** from the main menu. You should get 0 errors (warnings are okay and expected).

3. Go to the System Navigator, expand the Tutorial.jpr project again, then select **Project > Show Categories** from the main menu.

   Note: this helps to organize the files in a large project.

4. Expand the **HTML Sources** category beneath `Tutorial.jpr`. Select test_fwktutorial.jsp, then select **Run > Run test_fwktutorial.jsp** from the main menu. Perform the following:

   - Select **Hello, World!** from the list of lesson links displayed on the Test Framework ToolBox Tutorial page. This runs a very simple page.

     Note: If you can't run the Hello, World! page; revisit the steps listed above to ensure that you completed everything correctly. If the problem persists, follow the support procedure described in the Release Notes accompanying this ARU.

You are now ready for hands-on experience with the Oracle JDeveloper OA Extension. The ToolBox Tutorial lessons can be launched from `jdevdoc\index.htm`

Customer, Consultant or Support Representative Using JDeveloper on Linux

This section contains instructions to configure and test OA Framework if you are a customer, consultant or support representative using JDeveloper on Linux. It provides an overview of the directory structure and discusses how to:

- Configure the JDEV_USER_HOME and JDEV_JAVA_HOME environment variables.
- Launch JDeveloper on Linux.
- Create an OA Workspace and Project
- Obtain a database connection file.
- Assign Toolbox responsibilities.
- Configure the database connection and user.
- Test the setup.

Overview

These instructions assume you have successfully installed the JDeveloper OA Extension zip file which creates the following directory structure on your drive of choice.

Directory	Description
Tip: To open any of the documentation in the **jdevdoc** directories, open the `jdevdoc\index.htm`.	
`jdevdoc\javadoc\fwk`	Includes OA Framework Javadoc.
`jdevdoc\javadoc\aolj`	Includes AOL/J Javadoc.
`jdevdoc\javadoc\bc4j`	Includes BC4J Javadoc.
`jdevdoc\javadoc\uix`	Includes UIX Javadoc.
`jdevdoc\toolbox`	Includes OA Framework ToolBox Tutorial lesson/lab documentation.
`jdevdoc\devguide`	Includes the OA Framework Developer's Guide.

`jdevbin\`	Includes an extended version of the Oracle JDeveloper 10*g* executable and OA Framework class libraries.
`jdevhome\`	Includes the OA Framework ToolBox Tutorial source and developer working area.

Task 1: Configuring the JDEV_USER_HOME and JDEV_JAVA_HOME Environment Variables

Attention: These commands must be executed from the bourne shell.

1. Assign a value to the JDEV_USER_HOME variable. For example:

   ```
   JDEV_USER_HOME=/home/<username>/jdevhome/jdev
   ```

2. Assign a value to the JDEV_JAVA_HOME variable.

   Example - OS Red Hat version 2.1:

   ```
   JDEV_JAVA_HOME=/jdevbin/linux/j2sdk1.4.2_03
   ```

   Example - OS Red Hat version 3.0:

   ```
   JDEV_JAVA_HOME=/jdevbin/linux/j2sdk1.4.2_04
   unset LD_ASSUME_KERNEL
   ```

   Note: Both Red Hat versions, (2.1 and 3.0); have been tested successfully by Oracle JDeveloper OA Extension.

3. Export the two variables:

   ```
   export JDEV_USER_HOME
   export JDEV_JAVA_HOME
   ```

Task 2: Launching JDeveloper on Linux

Run this command from the bourne shell to launch JDeveloper:

/jdevbin/jdev/bin/jdev -verbose

Task 3: Create an OA Workspace and OA Project

Create a new OA Workspace and empty OA Project in JDeveloper. This deploys the standard OA Framework utility files so they are made part of the project.

Task 4: Obtaining a Database Connection File

Obtain the FND database connection (`.dbc`) file from the system administrator who installed the OA Framework database where you want to do your development. Place this file in the `<JDEV_USER_HOME>\dbc_files\secure` directory.

Task 5: Assigning ToolBox Responsibilities

If you have not already done so as part of your installation verification, assign the following ToolBox Tutorial responsibilities to a test user. Refer to the *Oracle E-Business Suite Security Guide* for information about creating users and assigning responsibilities to users.

Note: Use an existing user in your system or create a new test user.

- OA Framework ToolBox Tutorial (responsibility key is FWK_TBX_TUTORIAL).
- OA Framework ToolBox Tutorial Labs (responsibility key is FWK_TOOLBOX_TUTORIAL_LABS).

Task 6: Configuring the Database Connection and User

Use this procedure to configure the database connection and user:

1. Launch JDeveloper and then select **File > Open** from the main menu. Navigate to <JDEV_USER_HOME>\myprojects and open the OA Framework ToolBox Tutorial workspace file (toolbox.jws).

2. Expand the toolbox.jws in the JDeveloper System Navigator, to display its contents. Select the `Tutorial.jpr` project, then select **Project > Project Settings**.

3. Expand the `Oracle Applications` node, which is In the **Project Settings** dialog, and select **Runtime Connection**.

4. Locate the DBC file that you saved in Task 2 by using the **Browse...** button, which is In the **Connection** box. The file should be in the `<JDEV_USER_HOME>\dbc_files\secure` directory.

5. Specify the **User Name** and **Password** for the test user. This is the user that you assigned the ToolBox responsibilities to in Task 3. Select **OK**.

6. Repeat Steps 2 - 5 for the `LabSolutions.jpr` project.

7. Expand the **Connections** node in the JDeveloper System Navigator and then expand the **Database** node. Right-click on the **Database** node and select **New Connection...** to open the **Connection Wizard**. Follow the JDeveloper instructions to define a new database connection for the Oracle E-Business Suite database identified by the DBC file you selected above.

8. Select the `Tutorial.jpr` project In the System Navigator. Right-click and select **Edit Business Components Project...**.

9. Select the **Connection** option in the **Business Components Project Wizard** and set the **Connection Name** to the connection you just defined. Select **OK** to save your changes.

10. Repeat steps 7 - 9 for the LabSolutions.jpr project.

Task 7: Testing the Setup

Tip: To use Mozilla as your default browser, create a symbolic link.

To test your setup:

1. Open the OA Framework ToolBox Tutorial workspace file by selecting **File > Open** from the main menu. Navigate to `<JDEV_USER_HOME>\myprojects` and open the file `toolbox.jws`.

2. Select `toolbox.jws` and select **Project > Rebuild toolbox.jws** from the System Navigator main menu. You should get 0 errors (warnings are okay and expected).

3. Expand the `Tutorial.jpr` project and then select **Project > Show Categories** from the System Navigator main menu. (This helps to organize the files in a large project).

4. Expand the HTML Sources category beneath `Tutorial.jpr`. Select `test_fwktutorial.jsp`, and select **Run > Run test_fwktutorial.jsp** from the main menu:

5. Select **Hello, World!** from a list of lesson links displayed on the Test Framework ToolBox Tutorial page, to run a very simple page.

   Note: If you can't run the Hello, World! page; revisit the steps listed above to ensure that you completed everything correctly. If the problem persists, check the Oracle JDeveloper OA Extension FAQ for troubleshooting tips. If it still doesn't work, send an email to the OA Framework support mail list (see the OA Framework web site for additional information about this).

Building and Running 'Hello, World!'

Overview

This tutorial leads you through using Oracle JDeveloper with OA Extensions to create a very simple page.The tutorial needs minimal explanation, has very few steps, and includes no BC4J.

The tutorial assumes a basic prior knowledge of working with Oracle JDeveloper. You should also have your development environment set up and working. For assistance with this, see Setting Up Your Development Environment.

This lab should take approximately 1-3 hours.

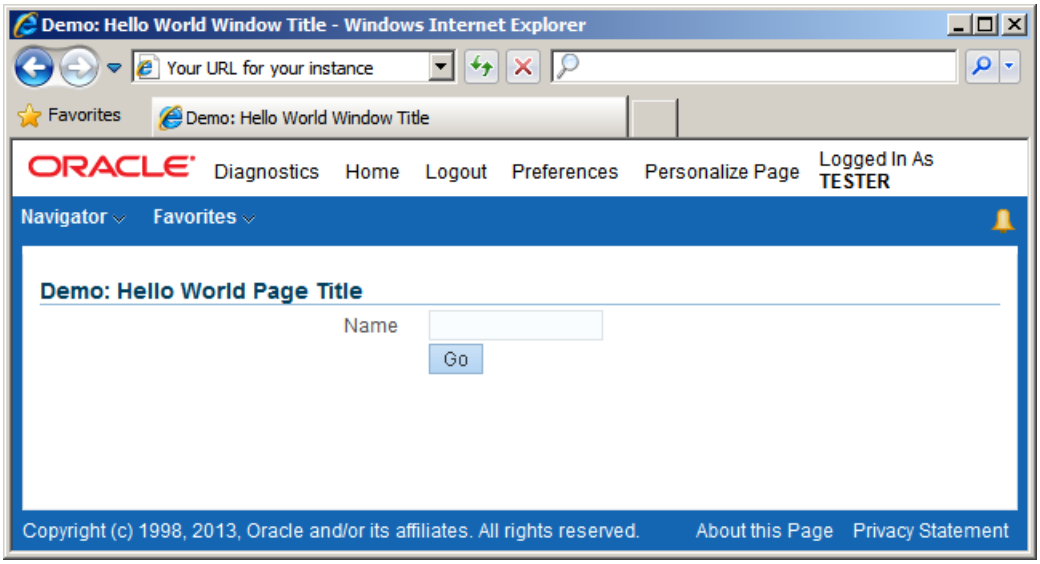
Hello, World Lab Goals

After completing this exercise, you will have learned how to:

- Create an Oracle Applications (OA) JDeveloper workspace and project.
- Use the JDeveloper OA Extension to create a very simple page.
- Create a controller and associate it with a region.
- Handle a submit button press action (an HTTP POST request).
- Run a page in regular and debug modes.

The resulting page has the global links (such as **Preferences**, **Logout**, and **Return to Portal**), a header, a footer, a **Personalize Region** link, one empty field and one **Go** button.

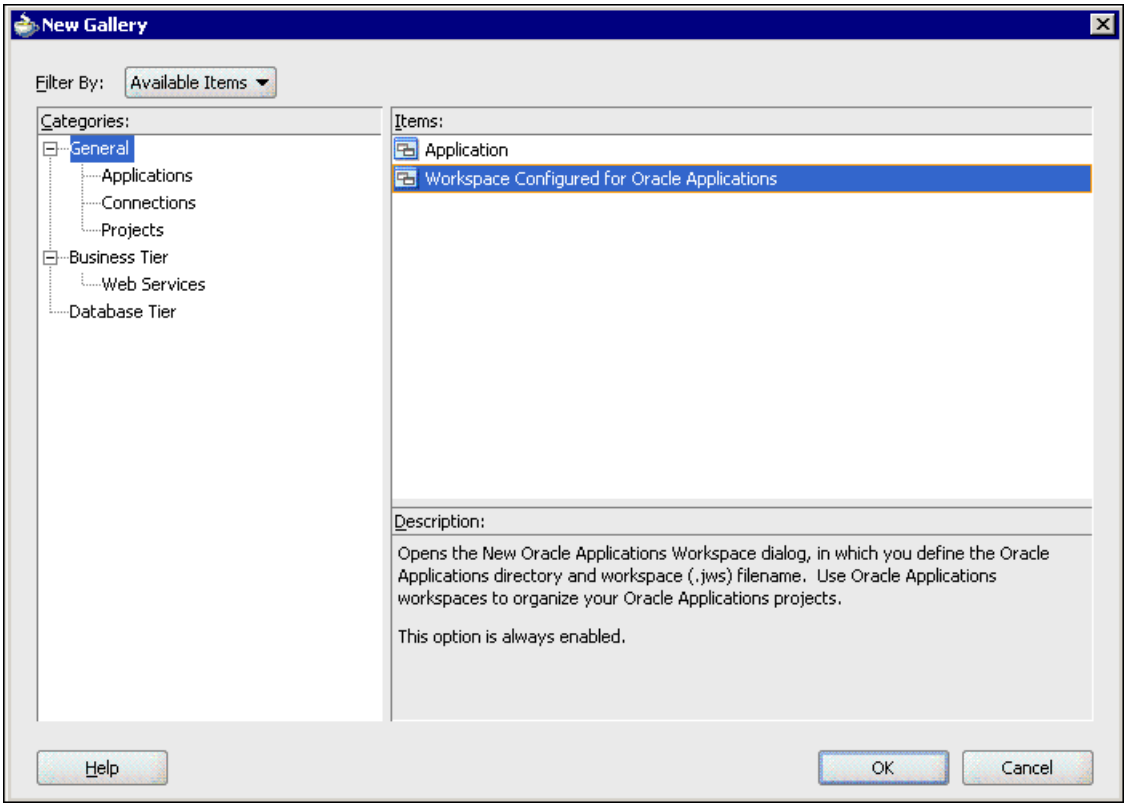
The finished "Hello World" page does nothing other than display a message when you enter a value in the field and click the **Go** button. Your final page and layout will look like the following:



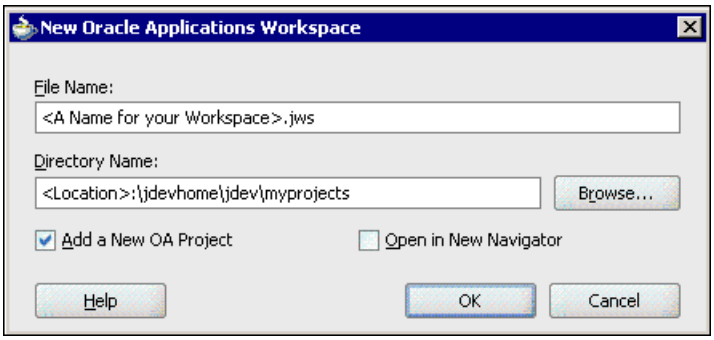
Note that there are several profile options that control features such as personalization, the visual appearance of global links, and other features, so what you see may be different from the picture above. Differences you are most likely to encounter are *not* seeing the **Personalize Region** link, seeing button icons above the corresponding global links, or not seeing certain global links such as **Customize** (if personalization is not enabled for your username).

Step 1. Create a New OA Workspace and Empty OA Project with the New... Dialog.

Select **File > New...** to open the **New...** dialog (shown in the following diagram). This dialog is also called the **New Object Gallery**.



Choose **General > Workspace Configured for Oracle Applications** from the **New...** dialog, or highlight **Workspaces** in the Navigator and choose **New OA Workspace...** from the context menu (right mouse button menu that changes depending on the context). You'll be prompted to create an OA workspace. Verify that the default workspace directory name points to your own <JDEV_USER_HOME>\myprojects directory, as shown in the following diagram. Modify the workspace file name as well (any name is okay for a workspace, such as HelloWorldOAWorkspace.jws). Check the **Add a New OA Project** check box.

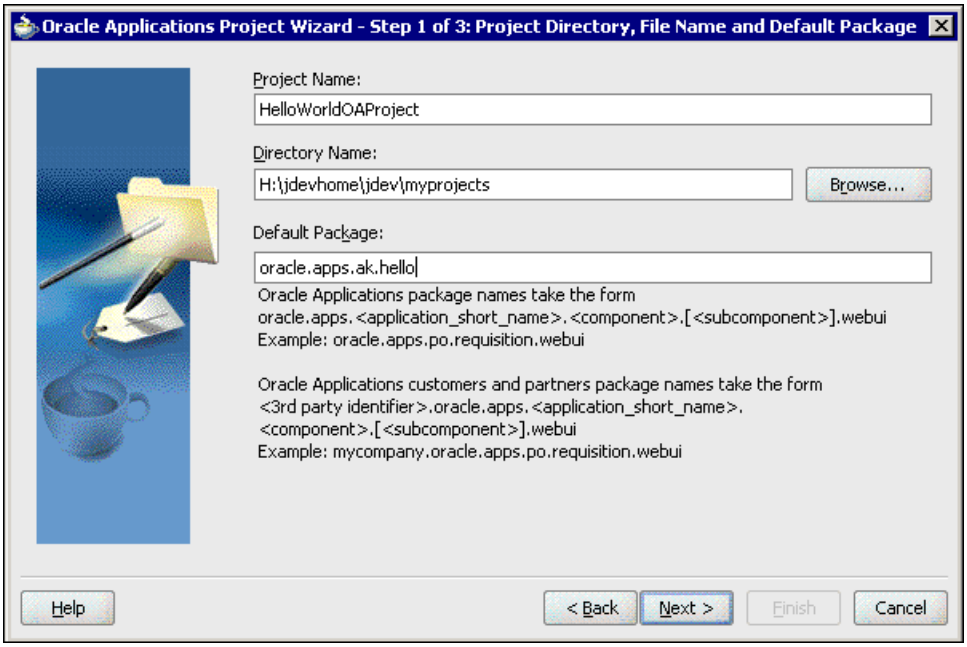


After you click **OK**, you will see the **Oracle Applications Project Wizard** after a Wizard Welcome screen. You can skip the welcome screen and select **Next**.

In Step 1 of the wizard, verify that the default project directory name points to your own `JDEV_USER_HOME\myprojects` directory, as shown in the following diagram. Modify the project file name as well (any name is okay for a project, such as `HelloWorldOAProject.jpr`). Set the default package name to the following (where "hello" is the component):

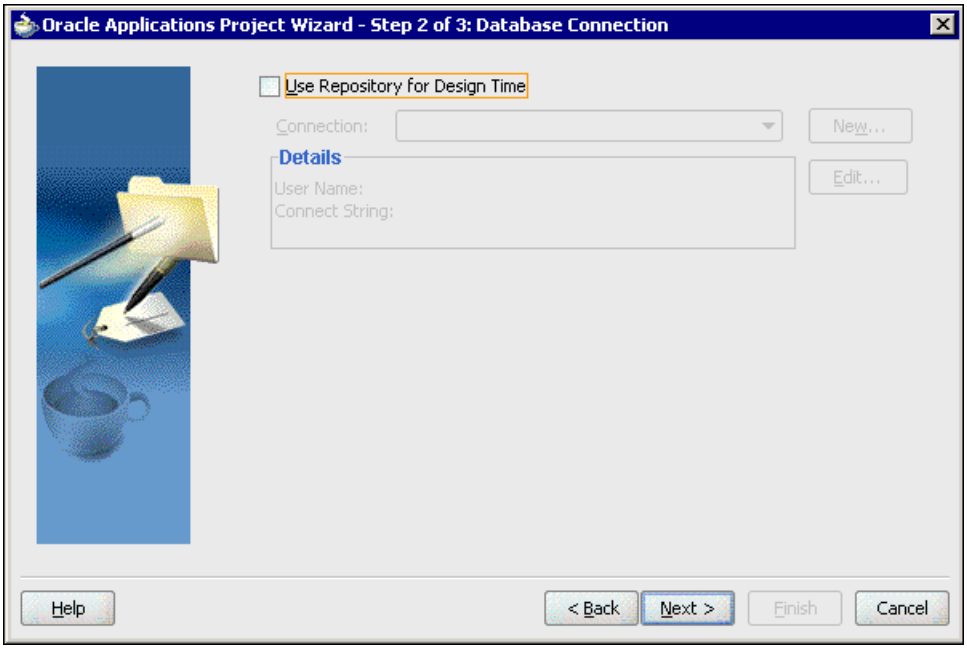
```
oracle.apps.ak.hello
```

Note: For this exercise and for all later lab exercises, ***you must use the exact package, page, region, item and variable names*** specified in the instructions, because the instructions depend on having these names. Specifically, you must use oracle.apps.... in your package names for the labs even if you are an Oracle E-Business Suite customer or partner (though you would use <3rd party identifier>.oracle.apps.... in production objects you create).

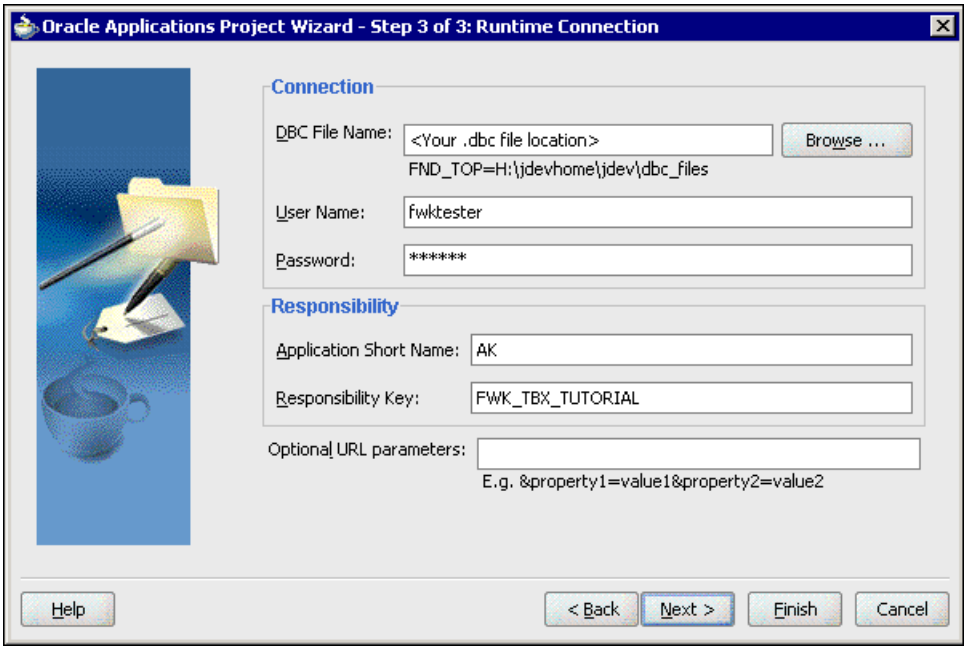


In Step 2 of the wizard, verify that the XML Path points to your own `JDEV_USER_HOME\myprojects` directory, as shown in the following diagram. You can include additional directories in the XML Path field if you have files in your project that do not reside under your myprojects directory.

For your Hello World project, you do not use the Repository for metadata in the database (the Hello World example uses only the XML files). In regular development work, where you use standard components that have been imported into the Repository, you would check the **Use Repository for Design Time** check box and provide connection information in Step 2.



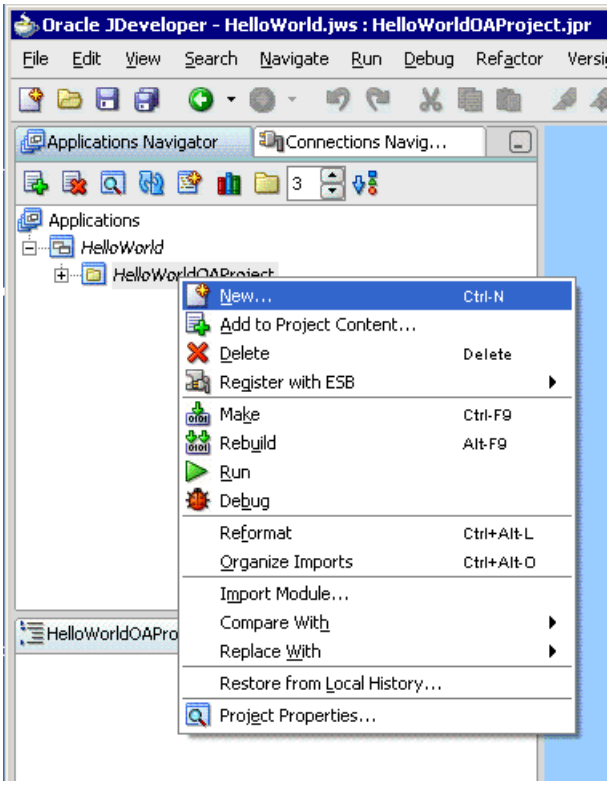
In Step 3 of the wizard, adjust the runtime connection information, if necessary, for the database and Oracle E-Business Suite username, password, and responsibility you are using (it must be a valid user and responsibility for your installation).



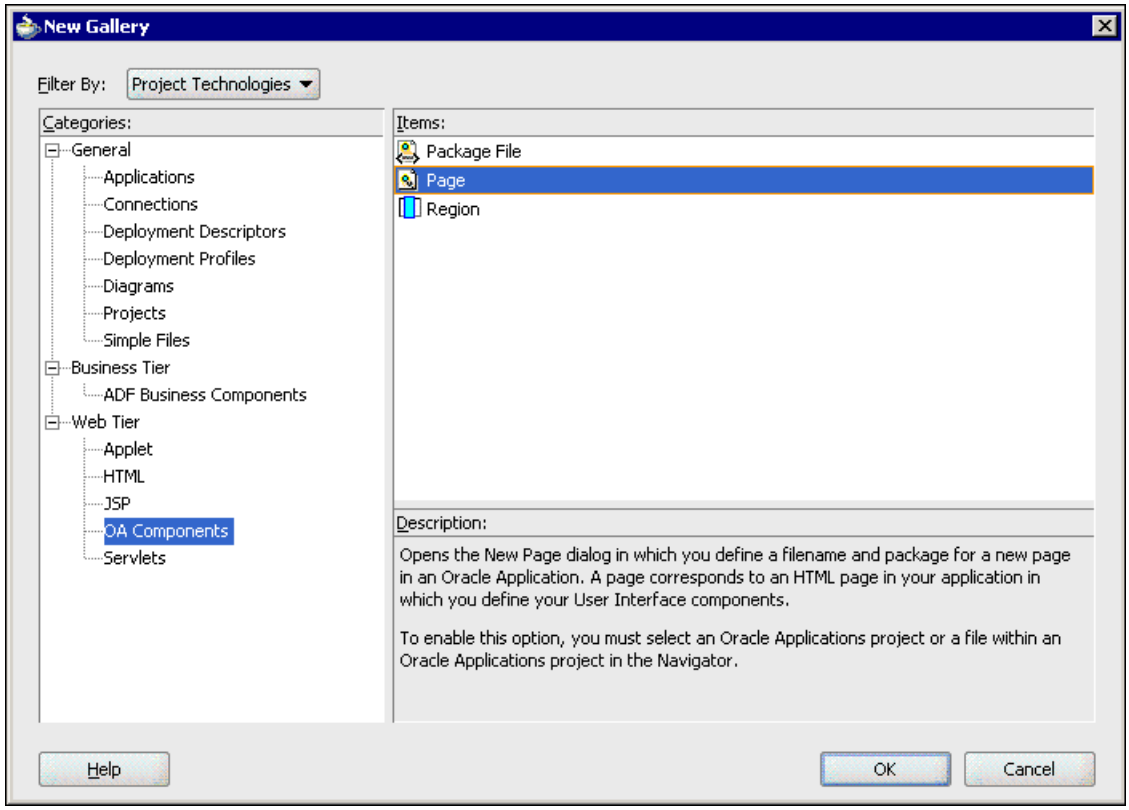
You will then have a screen after the one above that verifies the wizard is done, or you can select **Finish** on this page of the Wizard. When you select Finish, you will notice that JDeveloper will build some new files within your JDev Home directories.

Step 2. Create the OA Components Page File

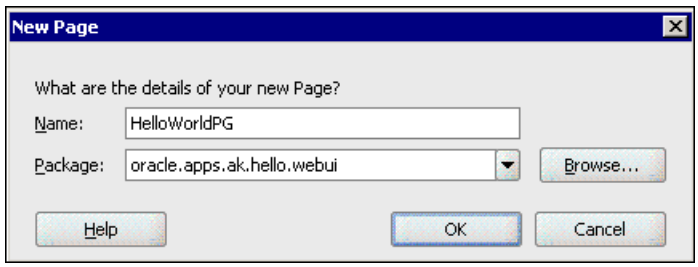
Within your new workspace, select your new project (your .jpr file). To add an OA Components page file to your project, choose **New...** from the context menu or use **File > New...** on the main menu to open the **New...** dialog.



Select **Web Tier > OA Components** in the Categories column. Then select **Page**, and press **OK** as shown in the following diagram:



You will then see a dialog box that asks for the name and package file for your new page. This dialog box is shown in the following diagram:



Name your page HelloWorldPG. Your page name cannot include any spaces.

In the Package field, type the following:

```
oracle.apps.ak.hello.webui
```

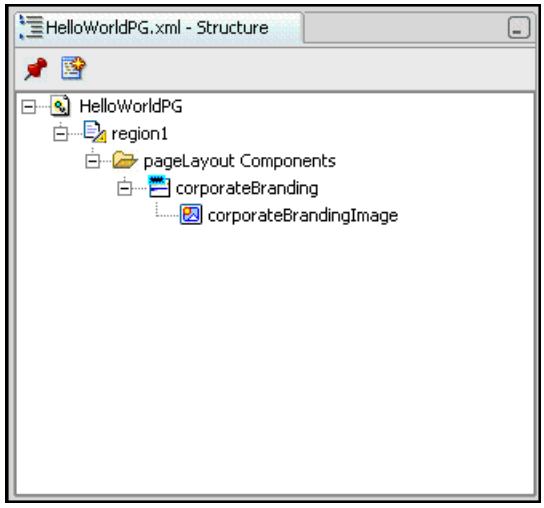
Note: Be aware that the Wizard will populate a package name (the default package for the project) so be sure to check it carefully.

Your package file name (which determines the location of the XML page file in the directory structure) should be set to **oracle.apps.*<application_shortname>*.*<optional_modulename>*.*<optional_subcomponent>*.webui** (to comply with Oracle E-Business Suite directory structure standards), where the application shortname is lowercase and is an existing Oracle E-Business Suite product shortname, such as INV.

Note: Pages migrated from old AK pages may use a different directory structure (pages instead of webui).

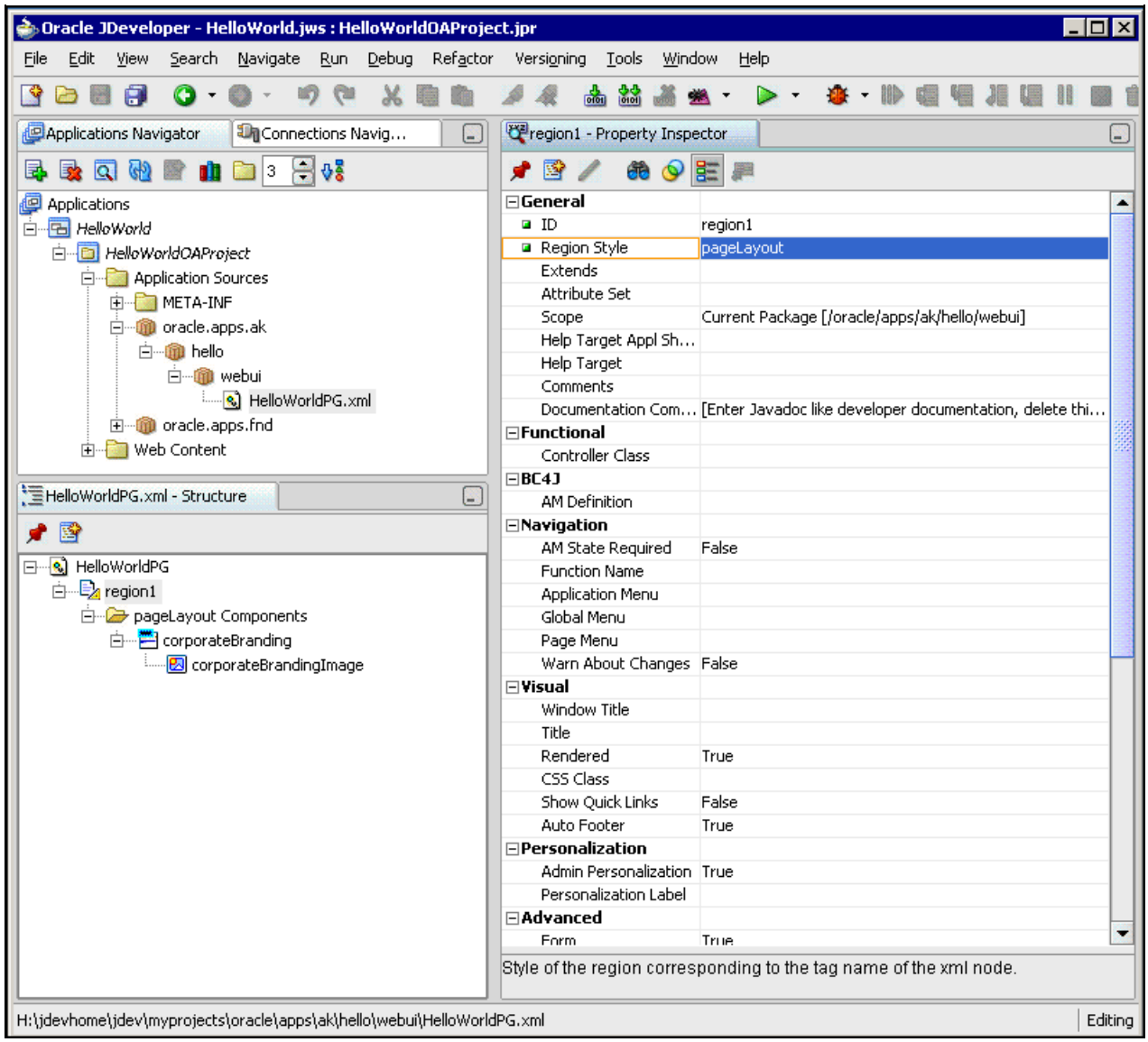
Be sure to follow the package name, directory location and object naming standards in the OA Framework File / Package / Directory Structure standards.

Your initial page structure appears in the Structure window as shown below, with an initial pageLayout region called region1, and a folder called pageLayout Components. The pageLayout Components folder contains a standard corporate branding image ("Oracle") that you can change through personalizations and through profile options. Changing the logo is not part of this exercise.



Step 3. Modify the Page Layout (Top-level) Region

JDeveloper creates your top-level page layout region for you automatically when you create your page.

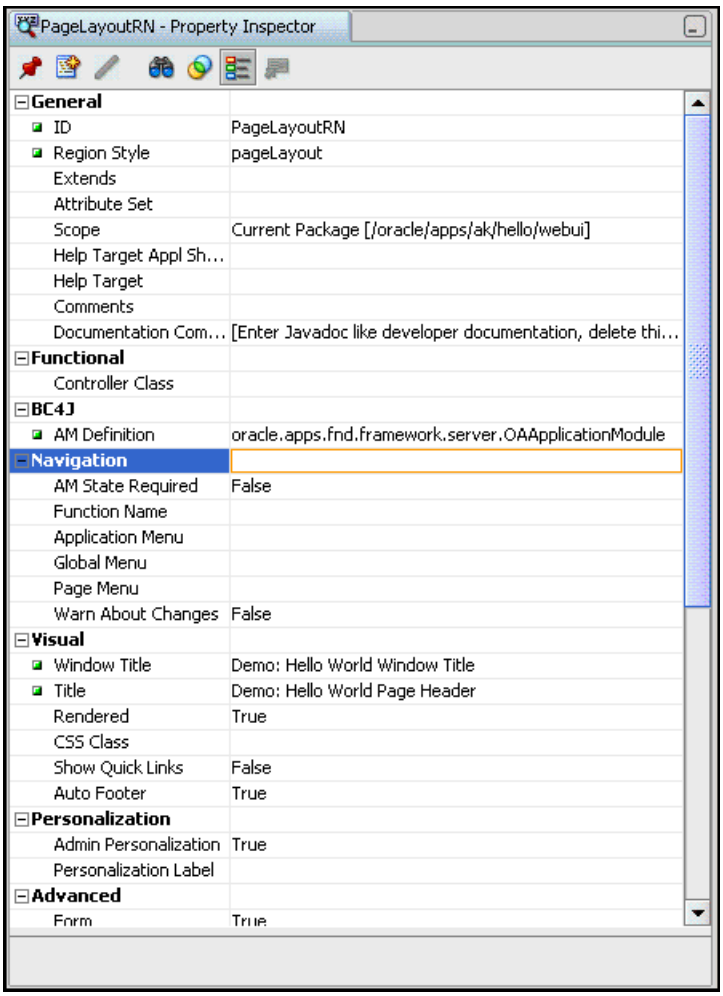


If the Property Inspector is not already open, select **View > Property Inspector** from the main menu. You can alternate between the alphabetical list of properties and the categorized list by clicking on the **Categories** button at the top of the Property Inspector (shown

above with categories enabled).

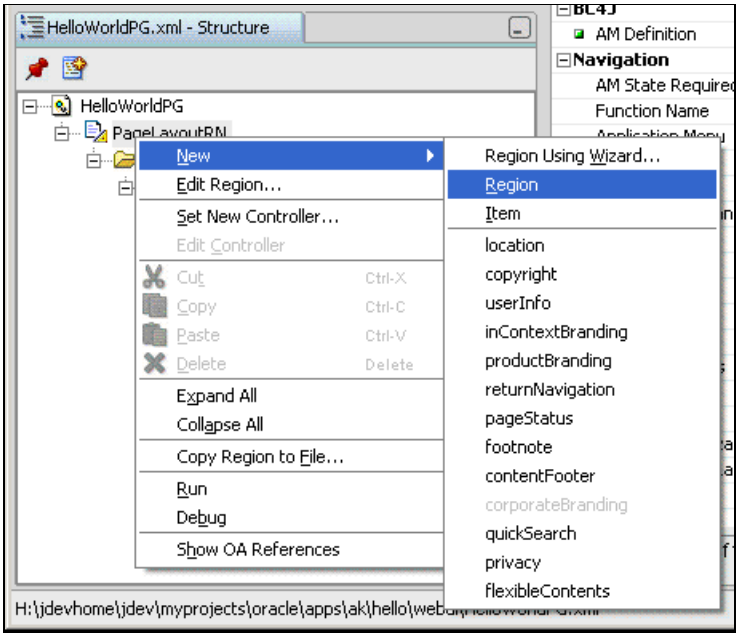
Set the following properties for your page layout region:

- Set the `ID` property to `PageLayoutRN`.
- Verify that the `Region Style` property is set to pageLayout.
- Verify that the `Form` property is set to True.
- Verify that the `Auto Footer` property is set to True.
- Set the `Window Title` property to `<your name>: Hello World Window Title`. This becomes the window title for the page.
- Set the `Title` property to `<your name>: Hello World Page Header`. This becomes the page header for the page (it appears under the blue bar).
- Set the `AM Definition` property to `oracle.apps.fnd.framework.server.OAApplicationModule` (you will have to type in the value). This is a generic application module supplied by the OA Framework.



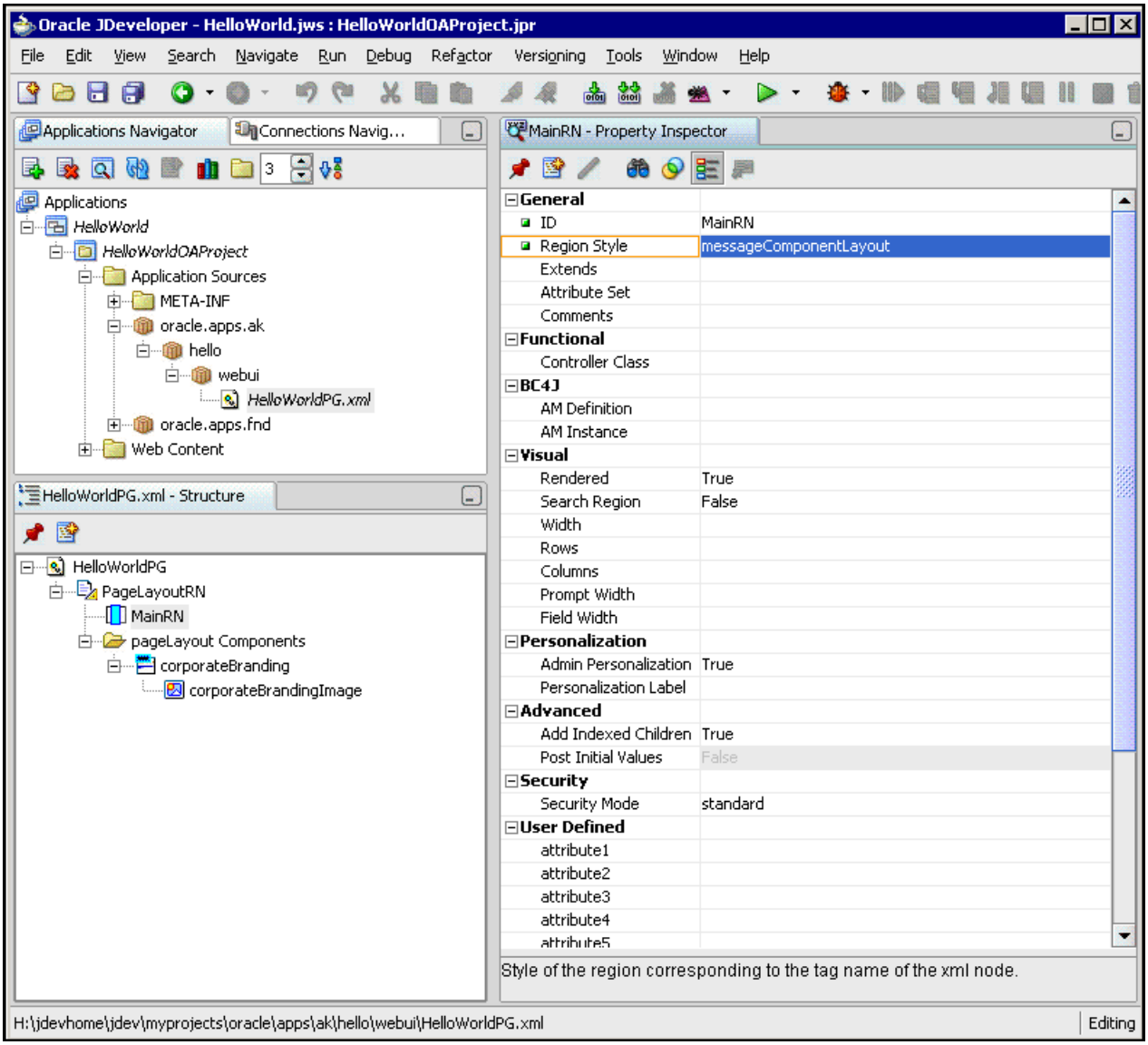
Step 4. Create the Second Region (Main Content Region)

Create your second region under the page layout region by selecting the page layout region in the Structure window and choosing **New > Region** from the context menu.



This region is merely going to provide a container for your items and ensure that the items are properly indented. Set the following properties for your second region:

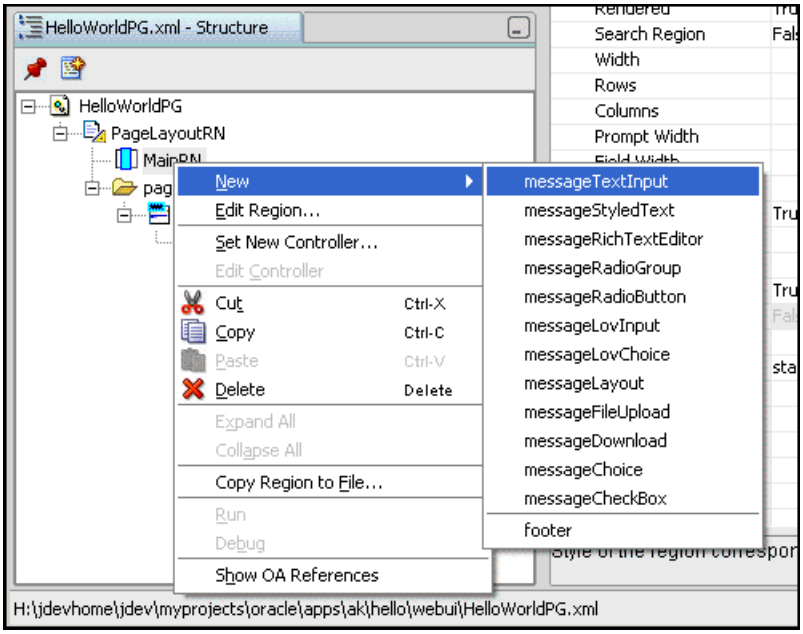
- Replace the default value in the `ID` property with MainRN.
- Set the `Region Style` property to **messageComponentLayout** (this provides an indented single- or multiple-column layout for the child items of the region).



If you want to, you can run your page at this point. You will see the global links, the copyright and privacy footer elements, and your page header text.

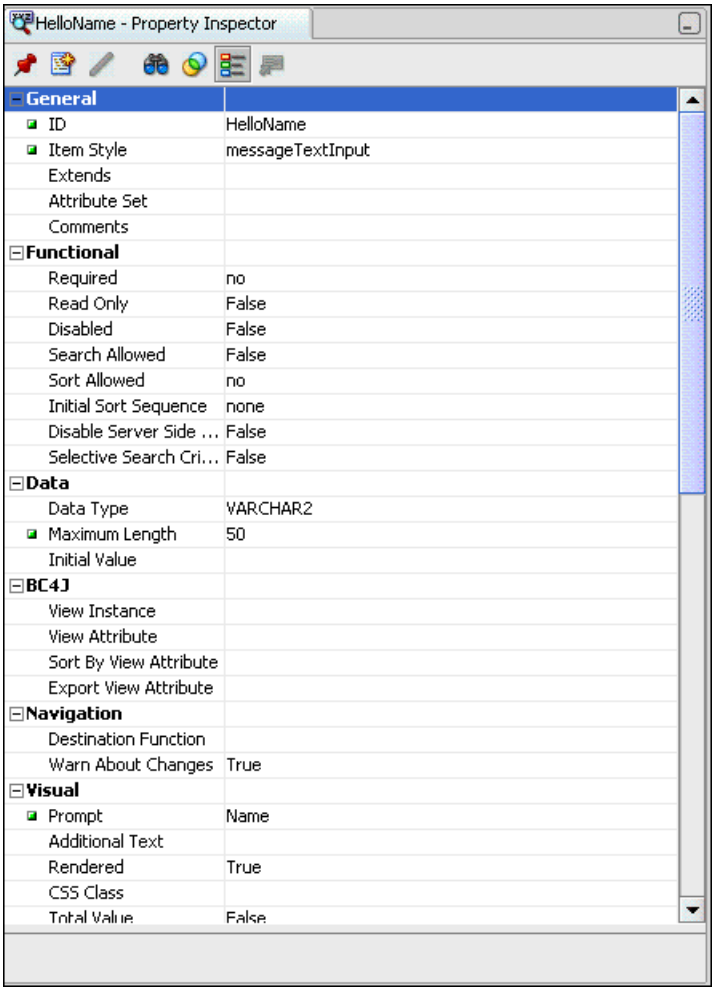
Step 5. Create the First Item (Empty Field)

Create your first item under the second region (main content region) by selecting the second region in the Structure window and choosing **New > messageTextInput** from the context menu.



Set the following properties for your item:

- Set the `ID` property to `HelloName`.
- Verify that your `Item Style` property is set to messageTextInput (this style provides a text label and an input field).
- Set the `Prompt` property to `Name` (in the later labs, you will use an attribute set to set the prompt).
- Set the **`Visual`** `Length` to `20`.
- Set the **`Data`** `Maximum Length` to `50`.

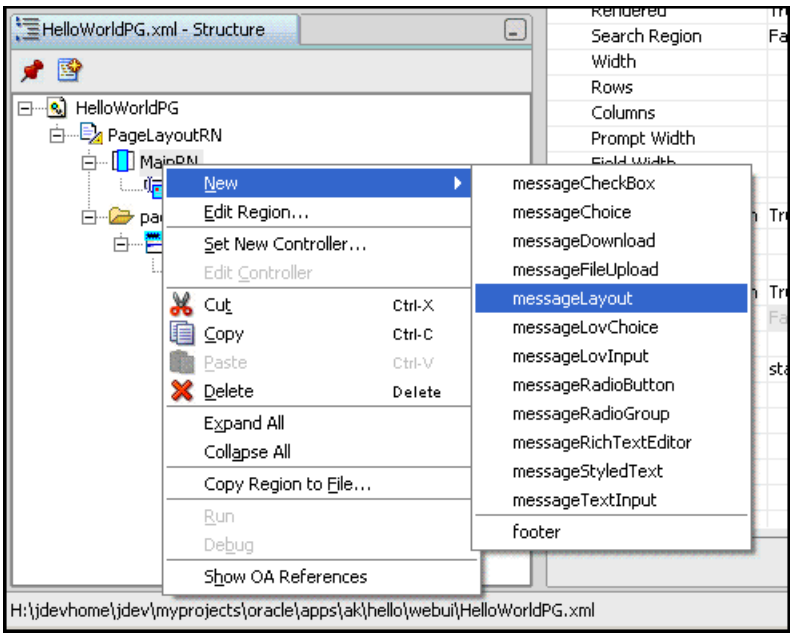


If you want to, you can run your page at this point.

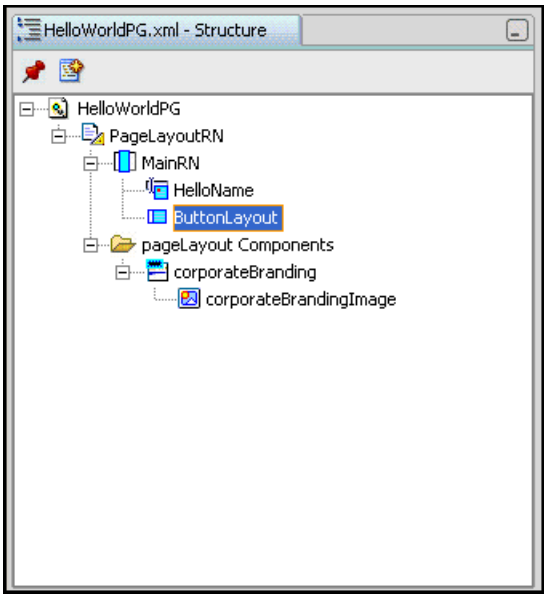
Step 6. Create a Container Region for the Go Button

To add a non-**message***-type bean such as a submitButton to a messageComponentLayout region, you must first add the bean to a **messageLayout** region.

Select the messageComponentLayout region and select **New > messageLayout**.



Name this region **ButtonLayout**.



Step 7. Create the Second Item (Go Button)

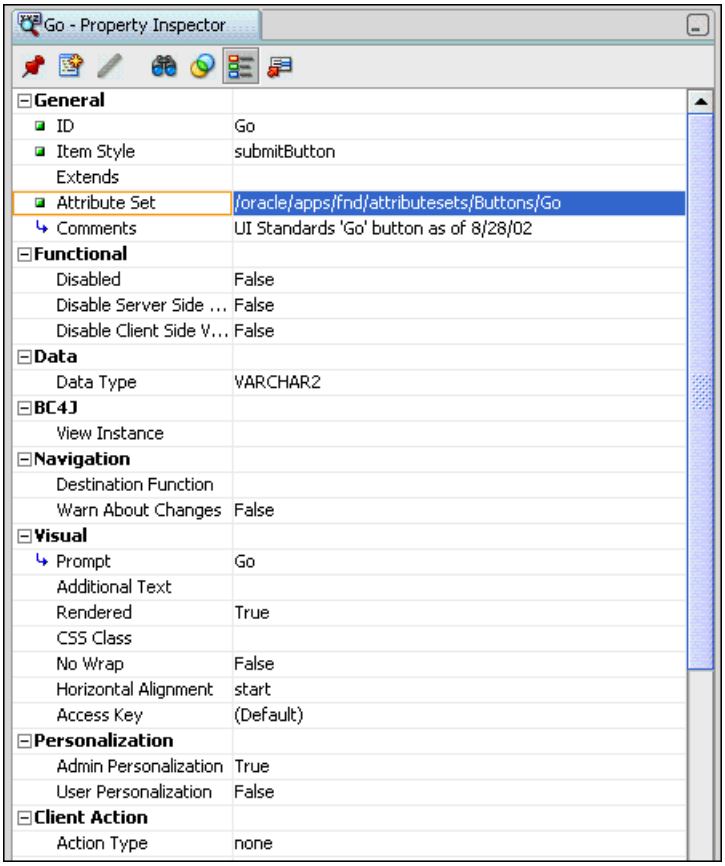
Create your Go button item by selecting the messageLayout region, ButtonLayout, in the Structure window and choosing **New > Item** from the context menu.

Set the following properties for your button item:

- Set the value of the `ID` property to `Go`.
- Set the `Item Style` property to submitButton.
- Set the `Attribute Set` property to `/oracle/apps/fnd/attributesets/Buttons/Go`.

Note: You can search for this attribute set, even though the attribute set file is not part of your project, by choosing the **Search in: Entire MDS XML path** option but not selecting the **Show Components in Same Scope Only** check box. You can use **/oracle/apps/fnd/attributesets/** and **Go%** as criteria for your search.

Verify that the `Prompt` property is now set to Go (this is your button label, inherited from the attribute set).

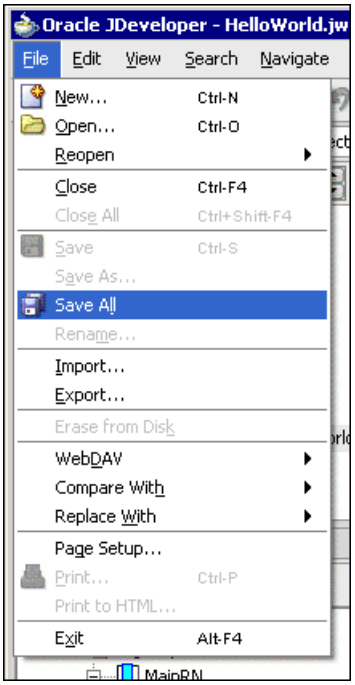


If you want to, you can run your page at this point.

Step 8. Save Your Work

Save your work. Using the menu choice **File > Save All** will save your metadata changes to an XML file as well as save all your other file changes (such as to a .jsp or .java file).

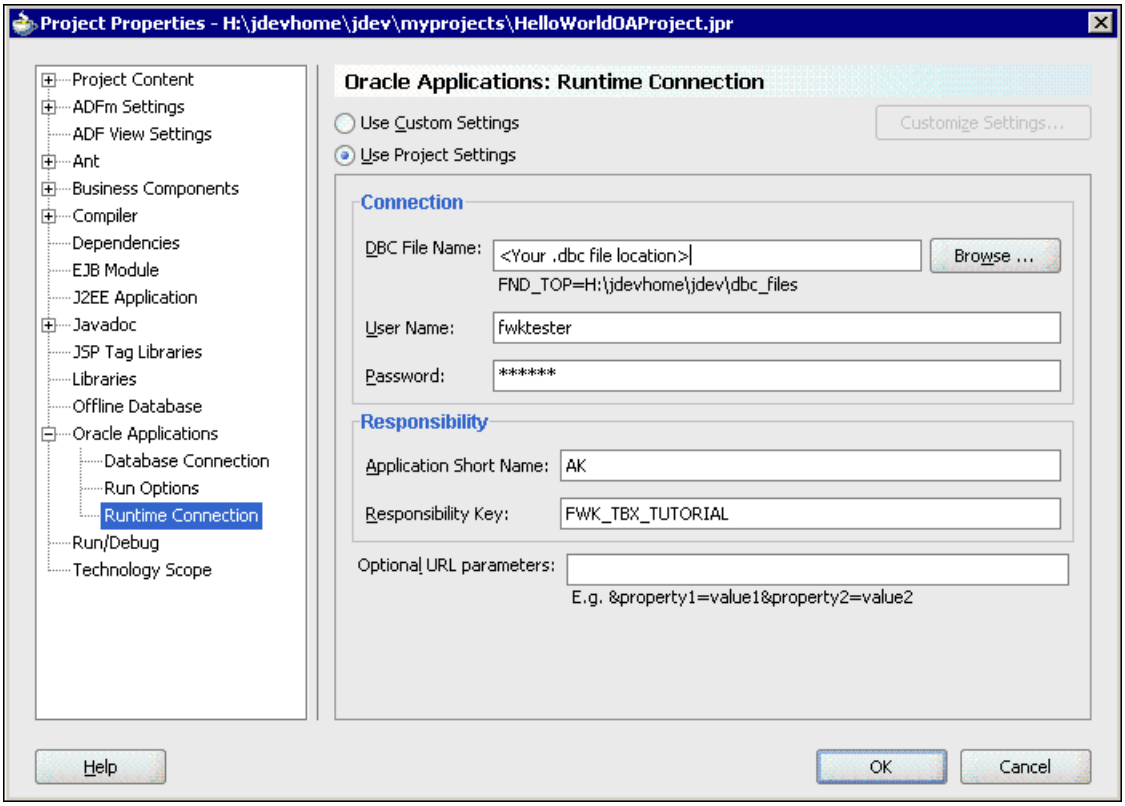
Tip: Though it usually will not be written out as a separate step in the exercises, you should **save your work frequently**.



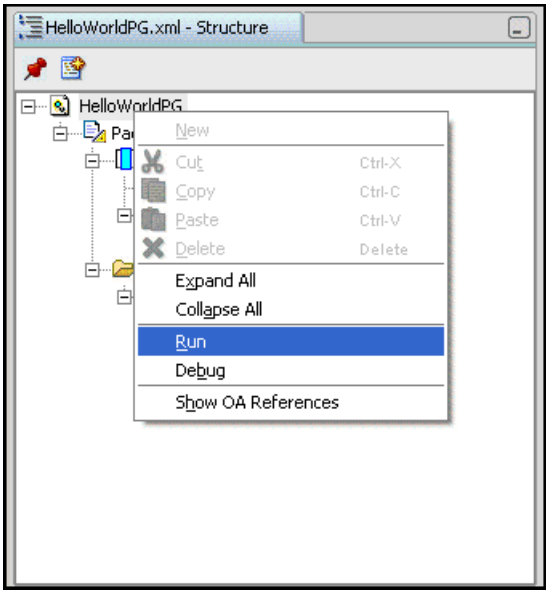
Step 9. Run Your Page Using the Run Option

You can try out your page using the **Run** option on the context menu.

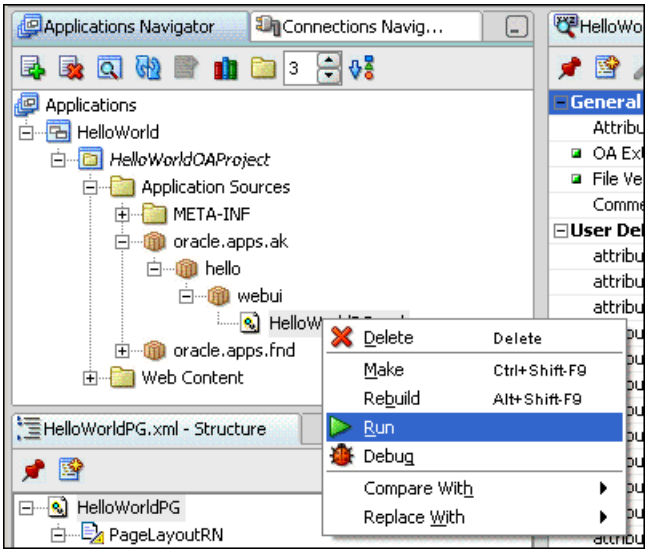
If you are using a database other than what you already have in your project settings, you will need to modify the **Runtime Connection** project settings by selection your project file and choosing **Project Properties...** from the main menu. Specifically, you must use a combination of Username, Password, (Responsibility) Application Short Name and Responsibility Key that is valid for your database to enable your session to log in.



You can use the **Run** option in the context menu to test your page in a standard browser. This option allows you to test your layout as well as functionality such as handling button presses. Select your page or page layout region in the Structure window, and choose **Run** from the context menu.



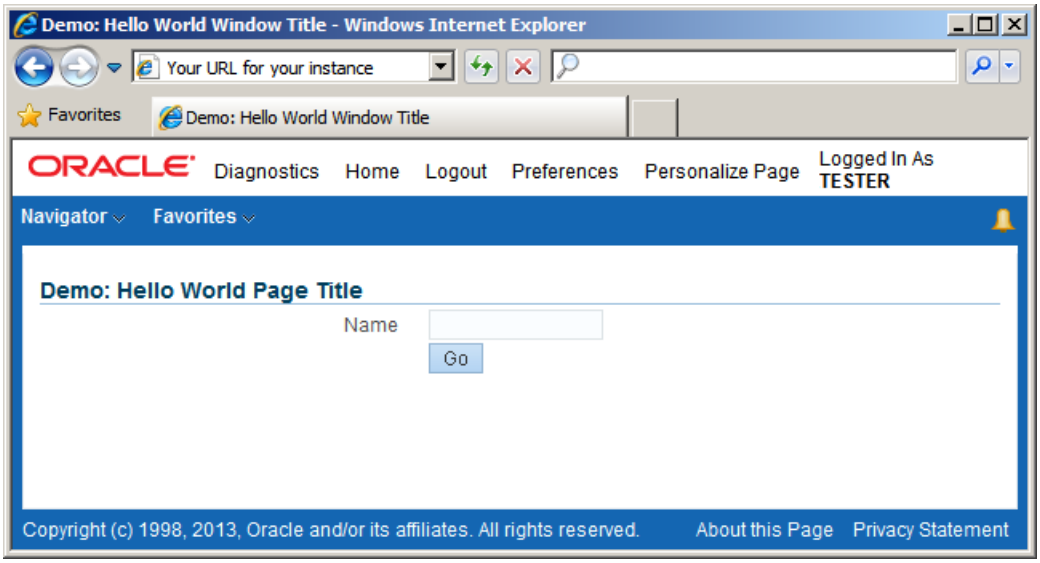
Alternatively, you can select your page in the Navigator window, and choose **Run <*page name*>** from the context menu.



You may have to wait a few minutes or more before you see your page in a separate browser window (it often takes longer the first time).

If your page does not appear after a few minutes, or gives errors, check the messages in the Log window. See the Hello, World! Troubleshooting Guide or the Oracle JDeveloper OA Extension FAQ.

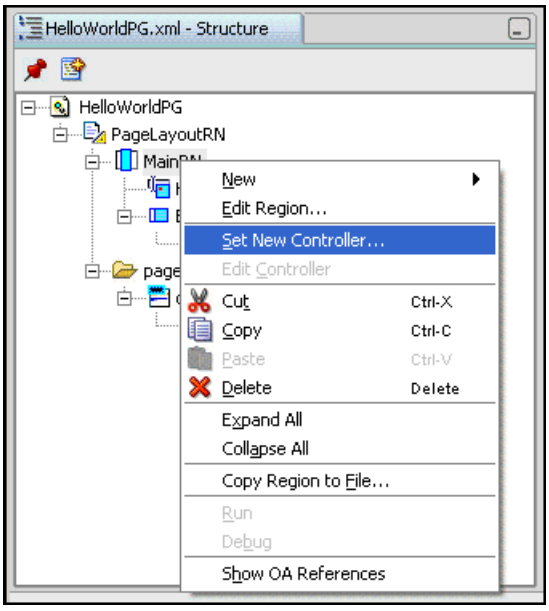
Your page should look like the following picture (with your own name in the page header and window title). You should see your page header, your **Name** field, and your **Go** button, along with global links and buttons (some global buttons may not appear depending on profile option settings). You may or may not see a **Personalize Region** link below your page header, depending on profile option settings. *Do not* personalize this page, as personalizations are data driven and you will affect anyone else building the Hello World page on the same database.



Each time you make changes to your layout (by adding regions or items, modifying properties, or changing code), you must run your page again to see your changes. If the Run process seems to hang for several minutes after you have already run your page previously, you may need to terminate the OC4J server using the **Run > Terminate > Embedded OC4J Server** main menu option, and then run again.

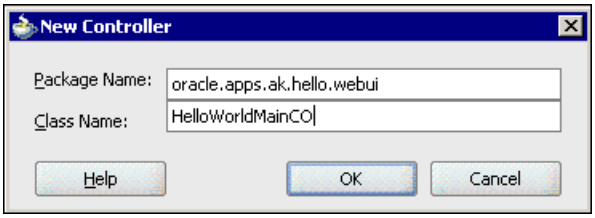
Step 10. Add a Controller

Add a controller to display a message when the user clicks on the Go button. Select your second region (MainRN) and choose **Set New Controller...** from the context menu.



Give your controller the package name **oracle.apps.ak.hello.webui** and an appropriate class name, such as **HelloWorldMainCO.**

Be sure you review the window before you click OK.



Step 11. Edit Your Controller

When you created a new controller, JDeveloper added files to your project in the package you created (oracle.apps.ak.hello.webui). The controller file is called **HelloWorldMainCO.java**.

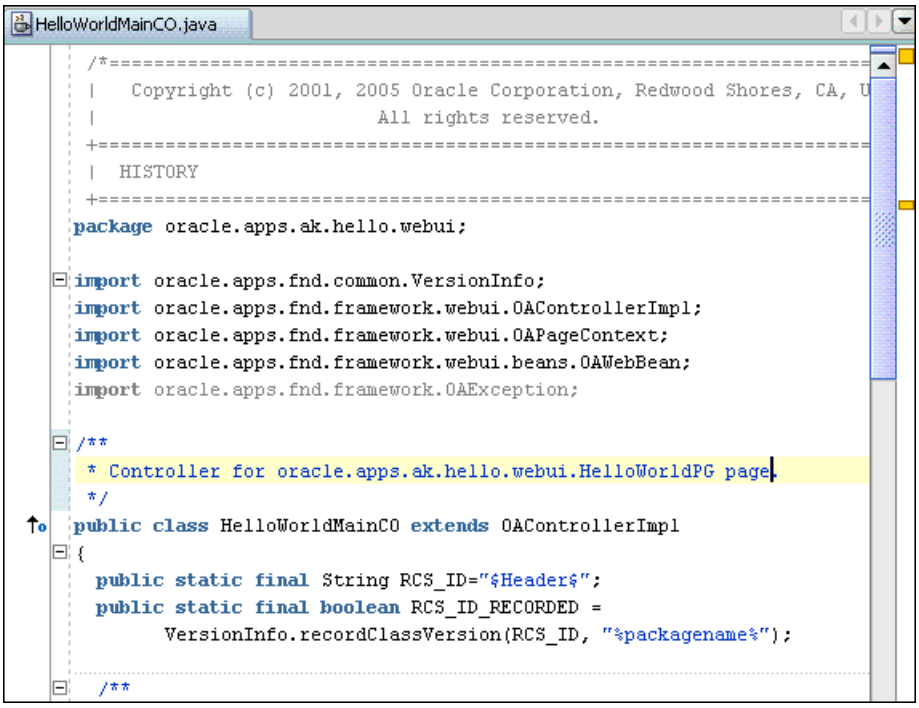
If you double click on **HelloworldMainCO.java**, the file will be editable in an adjacent screen.

Edit your controller code as follows:

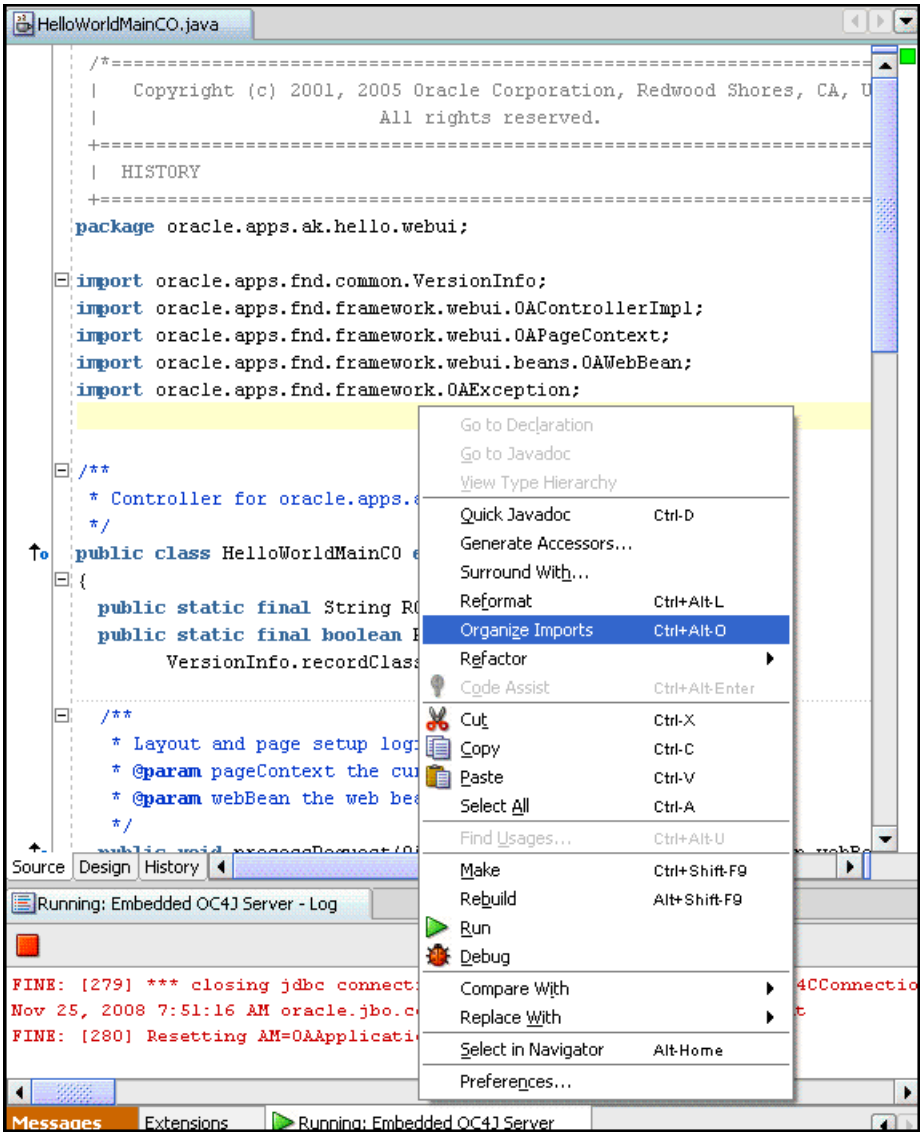
Add the following line as the last line of the import section to make the OA Framework `OAException` routines available:

```
import oracle.apps.fnd.framework.OAException;
```

Within the comments section at the top of the controller source code, you can also fill out what this controller is used for. Doing so makes it easier to understand what a specific controller is used for since there will be potentially many controllers in the same package. In our case, we are going to create a single controller in this exercise.



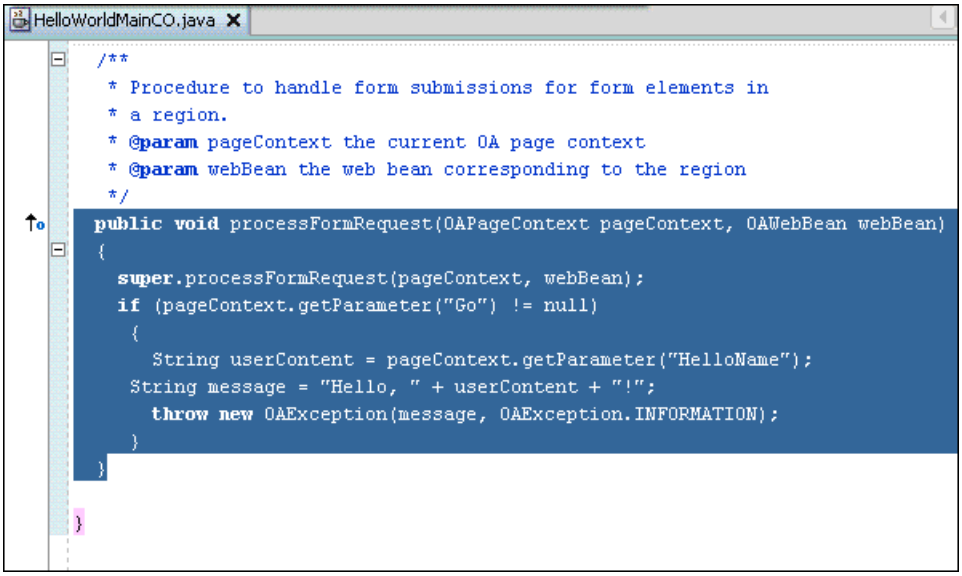
Note: You can sort your imports using the context menu in the code editor (**Organize Imports > Sort Imports**) as shown in the following picture. **However, DO NOT organize the imports now because our import is unused** (grayed out) and will be removed if we run the organize imports now.



Code the `processFormRequest()` method to match the following (making sure to match the item IDs you chose):

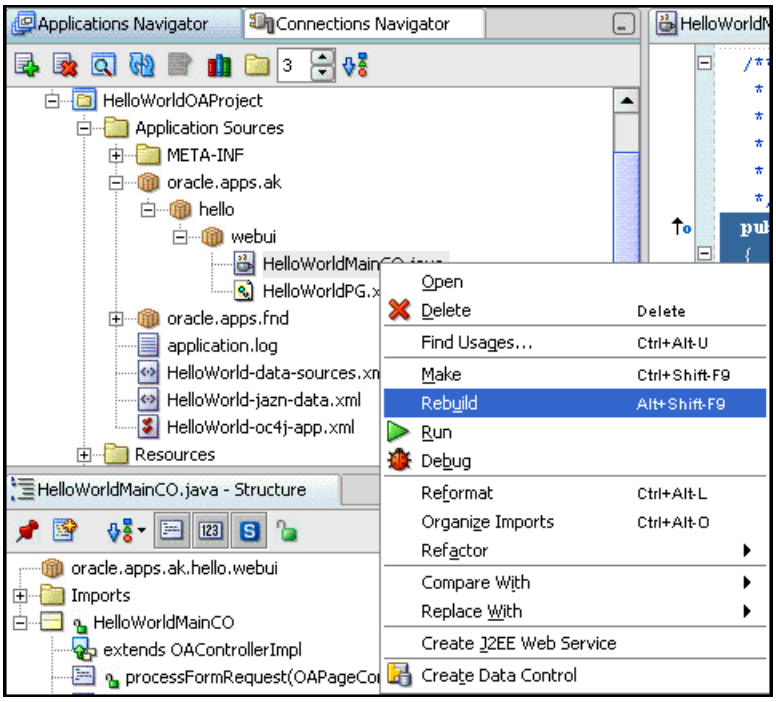
```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormRequest(pageContext, webBean);
  if (pageContext.getParameter("Go") != null)
  {
    String userContent = pageContext.getParameter("HelloName");
        String message = "Hello, " + userContent + "!";
    throw new OAException(message, OAException.INFORMATION);
  }
}
```

Note: Hardcoding a message text string is not translatable and would not be acceptable in a real Oracle E-Business Suite product. Normally you would define your message in Message Dictionary and call it from your code using its message name using the `OAException` routine.



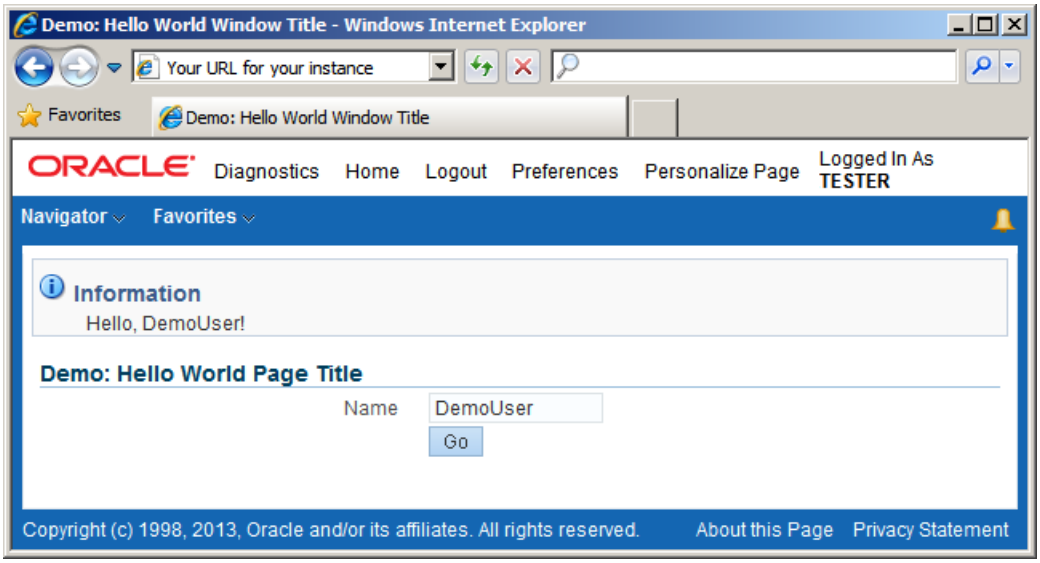
Step 12. Build Your Controller

Build your controller by selecting Rebuild from the context menu within the code editor window.



Step 13. Test Your Work Using the Run Option

Save your work, and then test it using the **Run** option. Type something into your field and then click the **Go** button. You should see the your page with an informational message that contains what you typed into the field, as shown:



Step 14. Test Your Work Using the Debugger

First, go back to your controller window to set up a breakpoint. Click on the line number next to the code line where you want the breakpoint to occur. You may have to toggle on the line numbers in JDeveloper.

The number changes to an icon indicating that it is a breakpoint in the source code. Set a breakpoint at the following line within your processFormRequest code:

```
if (pageContext.getParameter("Go") != null)
```

```
41    public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
42    {
43      super.processFormRequest(pageContext, webBean);
      if (pageContext.getParameter("Go") != null)
45      {
46        String userContent = pageContext.getParameter("HelloName");
47      String message = "Hello, " + userContent + "!";
48        throw new OAException(message, OAException.INFORMATION);
49      }
50    }
51
52  }
53
```

Now test your page by selecting your page name in the Navigator and choosing **Debug HelloWorldPG.xml** from the context menu (you can also do this in the Structure window).

Tip: When you are trying to debug an OA Extension page avoid launching the debugger from the project file. Instead, launch from a specific JSP or XML file (which may itself be a launch page).

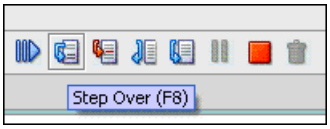
When the debugger starts up, your JDeveloper layout changes to a debugging layout that includes additional windows, including a debugging toolbar, a Breakpoints window, Data and Smart Data windows, and a Stack window (these may appear as tab controls); this is only a partial list.

In debug mode, your code executes as it normally would until you hit a breakpoint. So, in this example, the page renders as usual because the breakpoints are in the code that handles the Go button press. For the breakpoints above, you will need to enter a value in your page's Name field and select the Go button before you reach your breakpoint.

Type something into the Name field on your page and then click the **Go** button. If you go back to the JDeveloper window, you should find your first breakpoint line highlighted; processing has stopped just before that line is executed. This first breakpoint occurs just before your code checks to see if the Go button has been selected.

```
  public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
42 {
43   super.processFormRequest(pageContext, webBean);
     if (pageContext.getParameter("Go") != null)
45    {
46      String userContent = pageContext.getParameter("HelloName");
47    String message = "Hello, " + userContent + "!";
48      throw new OAException(message, OAException.INFORMATION);
49    }
50 }
51
```

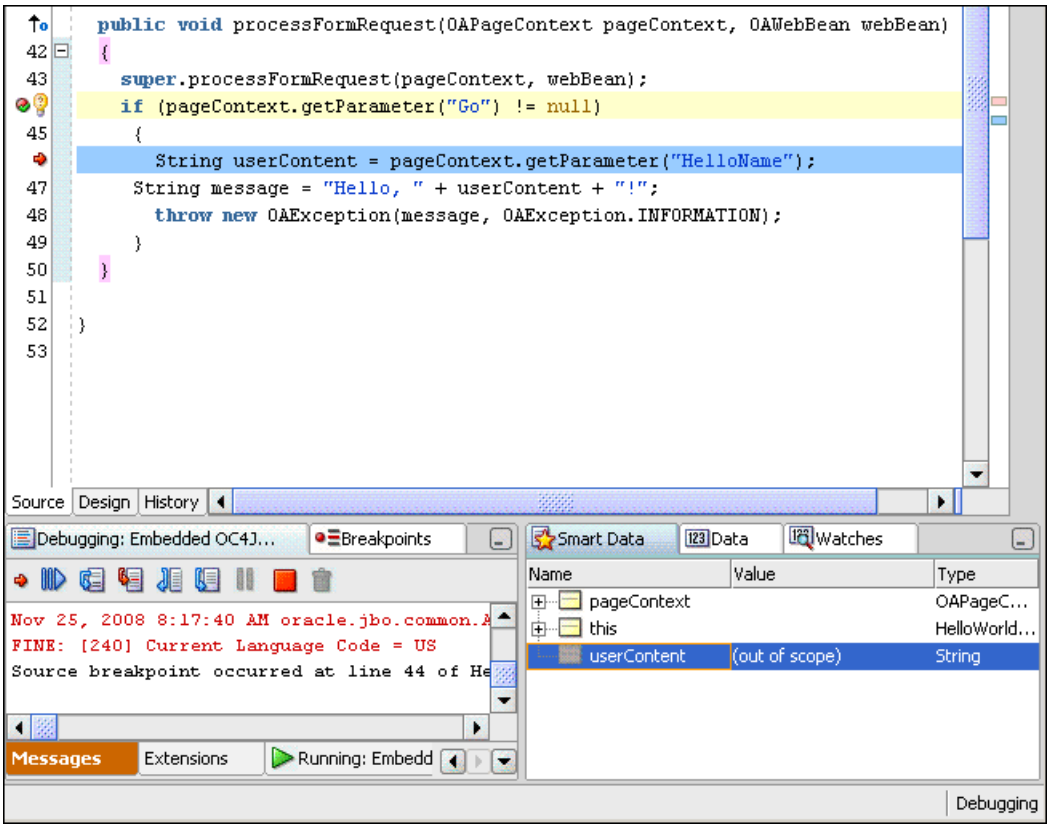
Now select **Debug > Step Over** from the main menu, or select the **Step Over** button, to continue to the next line of code.



The debugger then highlights the following line of code:

```
String userContent = pageContext.getParameter("HelloName");
```

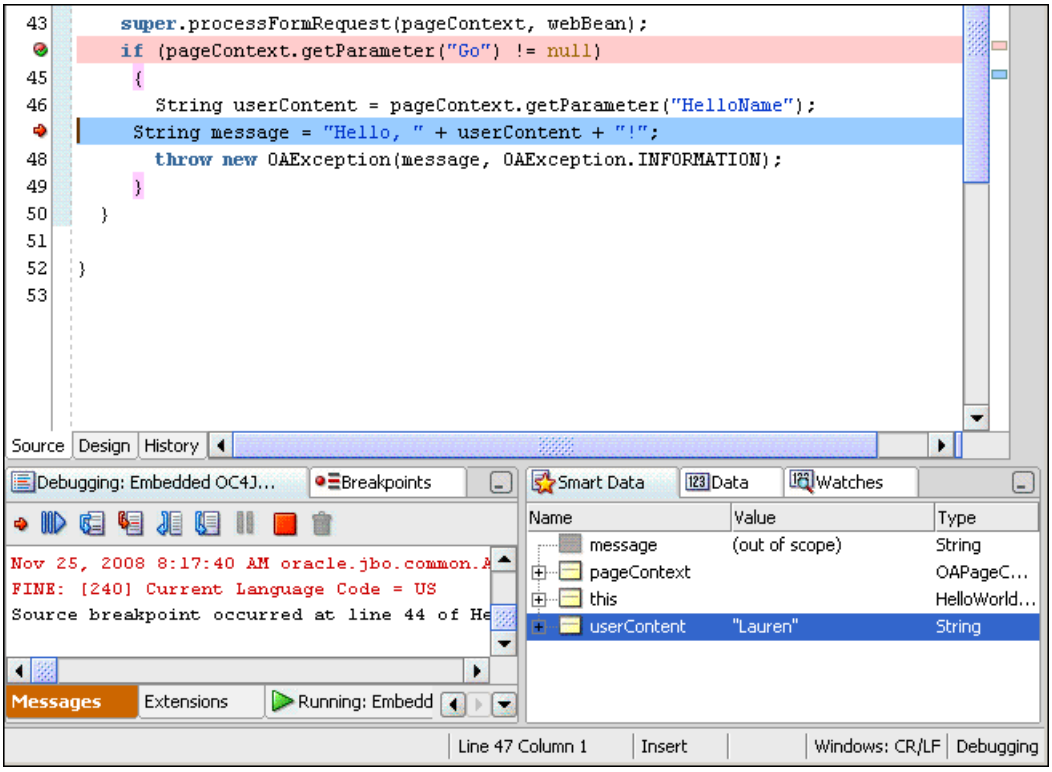
If you hold your mouse hovering above `userContent` on that line, you will see `userContent = (out of scope)` because the line has not yet executed.



Now select **Debug > Step Over** from the main menu, or select the **Step Over** button again, to continue to the next line of code.

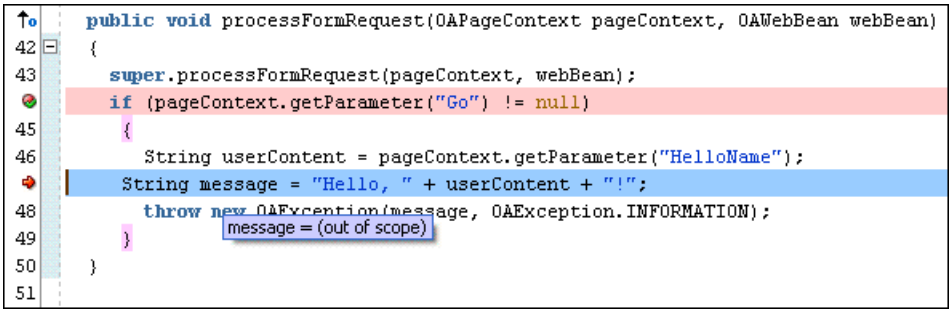
```
String message = "Hello, " + userContent + "!";
```

Now if you hold your mouse hovering above `userContent` on the new line, you will see its value as whatever you typed in (this only works for strings).



In the screen captures above, you can see the value of `userContent` in the **Data** or **Smart Data** window.

Note: If you hold your mouse over the word `message` on the same breakpoint code line, you will see `message = (out of scope)` because the line has not yet executed (and you will not see `message` at all in the data windows).

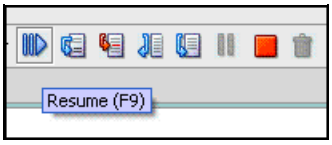


Step over again. You then get to the following line of code:

```
throw new OAException(message, OAException.INFORMATION);
```

At this point, if you examine the message string (either in the code window or in the Data or Smart Data windows), you will see your entire message text.

Select **Debug > Resume** from the main menu, or select the **Resume** button, to resume processing so your code runs to completion. You should see your page with an informational message that contains what you typed into the field, just as it looked when you ran your page using the Run option.



Congratulations! You have finished your first page with Oracle JDeveloper and the OA Framework!

.

OA Framework Development Runtime Configuration

Overview

This document briefly introduces some of ways that you configure your OA Framework runtime for pages in development, and some simple utilities that you might find useful.

Contents

- Page Test Modes
- Profile Options
- Technology Stack Information

Page Test Modes

As briefly mentioned in Building and Running "Hello, World!" there are several test modes that you can leverage at different points in the development/test cycle. See the associated links for each option for additional information about its use.

Option	Description	Recommended Use
OABackButtonTestMode	Tests the page's browser Back button support. See Testing OA Framework Applications.	This should be used only when performing Back button support tests.
OADumpUIXTree	Writes out the component tree for a page. See Debugging OA Framework Applications.	This should be used only when you need to diagnose a problem.

Project Settings

1. Select your project in the JDeveloper System Navigator and select **Project > Project Settings...** from the main menu.
2. In the **Project Settings** dialog, select the **Common > Oracle Applications > Run Options** page.
3. In the **Run Options** page, select the test modes that you want to enable in the **Off Options List** and shuttle them to the **On Options List**.
4. Select **OK** to save your changes.

Test JSP Cookies

The `test_fwktutorial.jsp` that you ran as described in Setting Up Your Development Environment includes the following cookie definitions. If you save this test JSP to create your own (the easiest way to create a test JSP), simply modify the following section as appropriate to enable or disable test modes.

Note: Test JSP cookie values override any values that you set for the project.

```
<SCRIPT LANGUAGE="JavaScript">
   document.cookie = "OABackButtonTestMode=0";

   document.cookie = "OADumpUIXTree=0";
</SCRIPT>
```

Profile Options

There are several profile options that you might want to set during development. See Appendix B: OA Framework Profile Options for a list of all the Oracle E-Business Suite profile options that affect the OA Framework.

Warning: Be conservative when you change profile option values! You should be setting them at the responsibility, user, or application level as appropriate in order to avoid impacting other developers working on the same environment.

Accessibility

- The Self Service Accessibility Features profile options controls whether pages can be used with assistive technologies. If

enabled, the OA Framework Partial Page Rendering (PPR) features are disabled -- which can be surprising if you're not expecting this behavior.

Personalization

- There are series of related profile options that control whether Personalization is enabled and accessible in a page. See the Personalization section in the profile options document (this also points you to additional information about the feature).

Logging

- There are a series of related profile options that control if and how logging is implemented. See the Logging section in the Profile Options document (this also points you to additional information about the feature).

Technology Stack Information

If you need to know what version of the OA Framework and Java you're running (among other things), see the instructions in Discovering Page, Technology Stack and Session Information.

Chapter 2: OA Framework Essentials

JSP Application Overview

If you do not have web application development experience, this document is intended to help you visualize -- *in very simple terms* -- how a generic JSP application is constructed, and what happens at runtime. It also identifies some key vocabulary/concepts used throughout the OA Framework Developer's Guide and ToolBox Tutorials.

Contents

- Key JSP Application Components
- What Happens at Runtime?
- Event Handling: Web UI vs. Classic Client UI
- Page Navigation
- What is a Cookie?
- More about Servlet Sessions

Suggested Reading

For additional information about JSP applications and servlets (none of which is required reading for working with the OA Framework), you might want to review the following for tutorials, documentation and book suggestions:

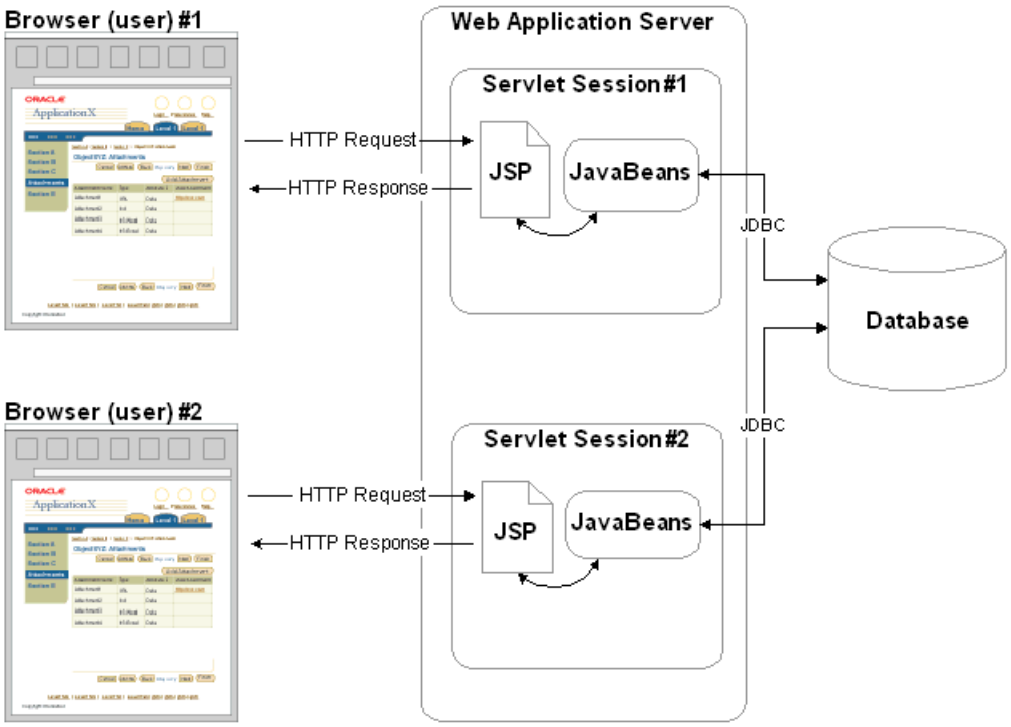
- Oracle Technology Network
- JavaSoft Java Developer Connection

Key JSP Application Components

A typical JSP application involves the following components: a browser for client access, a database for enterprise data and a web application server ("middle tier") where the application objects live.

- The browser communicates with the application tier using **HTTP** (Hyper Text Transfer Protocol), which involves sending a **request** message to which the middle-tier replies with a **response** message.
- A **JSP** is a file with some HTML and Java code that executes top to bottom. At runtime, it is compiled into a Java class, which is actually a **servlet**.
- A **servlet** that implements a standard API.
- A **servlet session** is a mechanism for maintaining state between HTTP requests during a period of continuous interaction between a browser and a web application. A session may be initiated at any time by the application and terminated by the application, by the user closing the browser, or by a period of user inactivity. A session usually corresponds to an application login/logout cycle
- A **JavaBean** (or "bean" for short) is simply a reusable component that implements specific design patterns to make it easy for programmers and development tools to discover the object's properties and behavior.
- Any objects in the middle-tier that communicate with the database use a **JDBC** (Java Database Connectivity) driver.

Figure 1: Key web application components and browser/server communication



What Happens at Runtime?

Step 1

When the user selects a link, a button or an active image, the browser sends an HTTP request to the web application server for processing. For the purposes of this introduction, we will focus on the two primary HTTP request methods (`POST` and `GET`) that are relevant for an OA Framework application.

HTTP GET

Whenever the user clicks a link or an image with an associated URL (such as http://www.example.com) the browser submits a `GET` request.

You can think of a `GET` as a postcard: both the address (URL) and any information the sender wants to convey (URL parameters) are written on the card itself (which is inherently space-constrained; how much can you write on a postcard?). This means that all the information for the communication is visible externally (and in an HTTP `GET`, all the data sent to the server is visible as parameters on the URL).

HTTP POST

Whenever the user clicks a button, image or link that performs a form submit in an OA Framework application (see What is a Form? below), the browser submits a `POST` request to the server (technically, a form can be submitted with a `GET`, but for the purposes of working with the OA Framework, you can assume a form submit is a `POST`).

You can think of a `POST` as an envelope: the address (URL) is written on the outside, but the content within has the information the sender wants to convey. There's no limit to the amount of information that can be stored inside the envelope. Furthermore, the submitted data is not visible on the URL -- just as the contents of an envelope are not externally visible (although the metaphor isn't absolutely accurate: a developer could also add some parameters to the URL when doing a form submit).

What is a "Form?"

In simple terms, a "form" lets you collect data entered by users into "form fields," and send that data to the server for processing.

A form is an HTML construct that groups data entry controls such as fields (both hidden and visible), poplists, and so on with action controls (like buttons) that are capable of "submitting the form." When the user selects a **submit** button, for example, the browser issues a `POST` request, which sends the form's data to the server.

Tip: People often use the terms `POST` and "submit form" interchangeably when talking about the OA Framework.

Step 2

The HTTP listener in the web application server routes the incoming request to the JSP. The developer's code does not know or care whether the browser issued a `POST` or a `GET`. All it does is read request values to determine what to do. So, for example, one of the request values might tell the JSP that a **Go** button had been pressed, which means it must execute a query.

Step 3

As shown in Figure 1 above, the JSP delegates to one or more JavaBeans, which implement various behaviors including database interaction. Once they have completed their work, the JSP prepares the appropriate HTML content to send back to the browser in the response.

Note: We included the JavaBeans in this example just to make the point that in an application of any complexity -- and modularity -- the JSP does not do the application work on its own since you should not combine model, view and controller code in the same file. However, there is no absolute technical requirement for the JSP to work with any other Java classes, and if it does, there is no requirement that these classes be JavaBeans.

Step 4

The browser displays the HTML it received in the response.

Event Handling: Web UI vs. Classic Client UI

In traditional client/server applications, you have the option of handling events ranging in granularity from very low-level mouse movements to field, region and finally, window-level scope. Furthermore, when you communicate from the client to the server, you can send a single value to be validated back to the server while expecting a single validation result back. You can then modify the user interface accordingly, which allows for a highly interactive experience for the user.

In a web application, you essentially handle "page-level" events (unless you are using experience, and since the OA Framework Coding Standards and Oracle Browser Look and Feel (BLAF) UI Guidelines prohibit this, we will not consider it here). In this case, as users navigate from field to field and enter data, there are no events for you as a developer to handle.

Tip: OA Framework provides partial page rendering (PPR) support for some actions, which allows for a more interactive user experience. This is fully described in Chapter 4.

When the browser finally sends a request as described above, *all* the page data is sent in that single communication -- including any user-entered values and information about what actions the user wants to perform. The developer reads request values to determine what happened (if the user pressed a button, which one was it?), does whatever work is required by the selected action, and then transmits a new HTML page back to the browser.

Page Navigation

So far, we've reviewed what happens (in general terms) when the browser communicates to the web server and vice versa, but we have not discussed the primary mechanisms for navigating from one page to another.

Note: In the following generic descriptions, it does not matter whether the request sent by the browser is a `POST` or a `GET`.

Standard Request

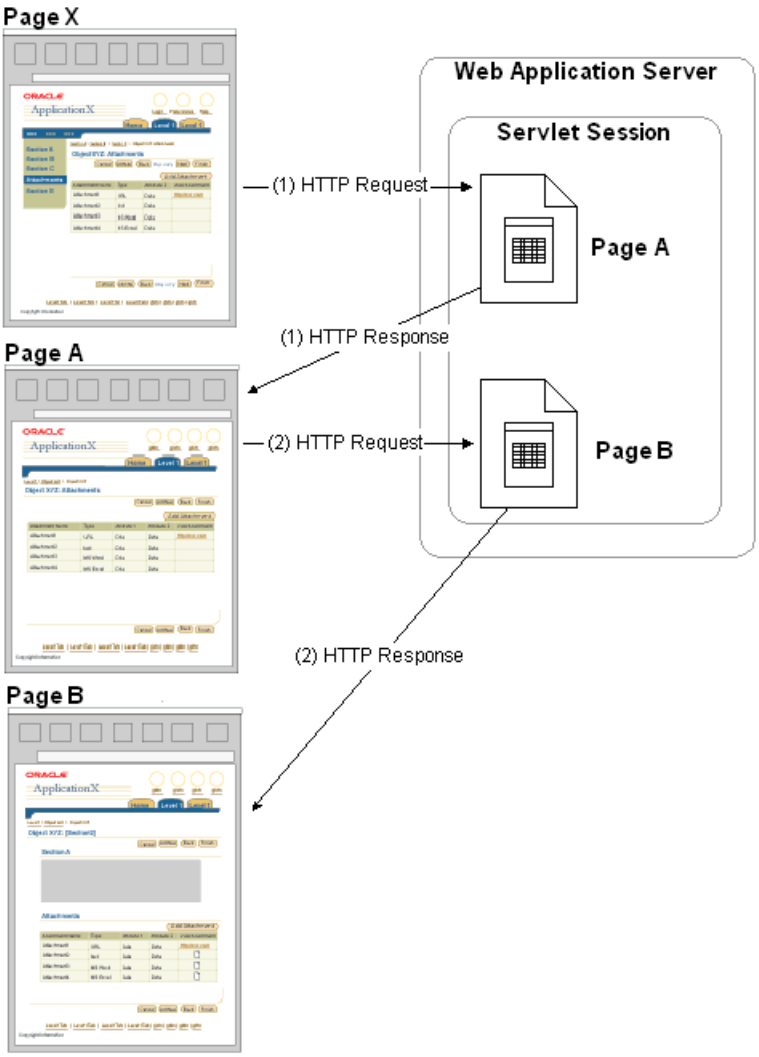
Scenario

A user selects a link on Page X to navigate to Page A. While on Page A, they select a link to navigate to Page B.

Implementation

The browser sends a request to Page A, which does its work and sends a response to the browser including the HTML to display. When the user indicates that they want to see Page B, the browser sends a new request to Page B, which does its work and sends a response so Page B can display.

Figure 2: Standard request illustration



JSP Forward

Tip: You will code many JSP Forwards in your OA Framework application. You must understand this concept.

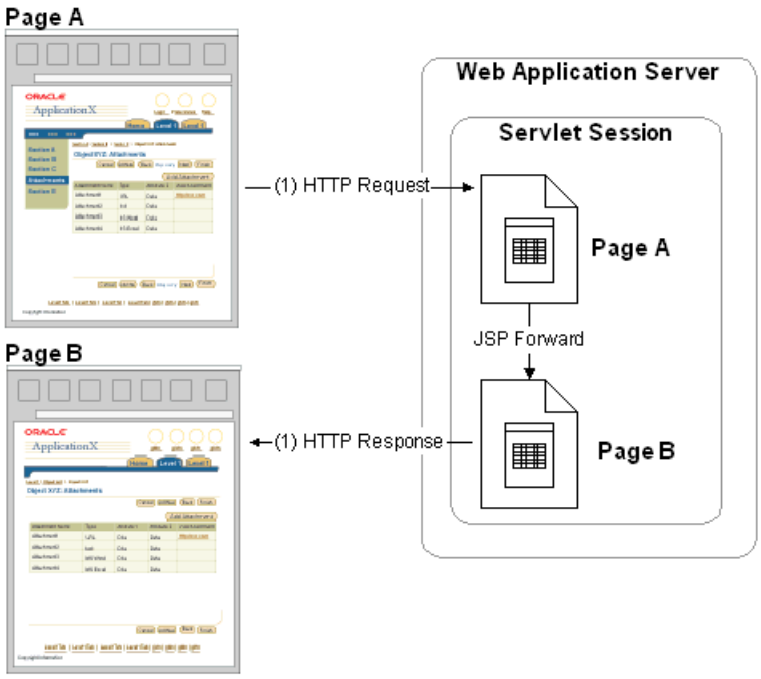
Scenario

While on Page A, a user selects an action from a dynamically defined list. The code in JSP A needs to handle that action to determine what page to display in response.

Implementation

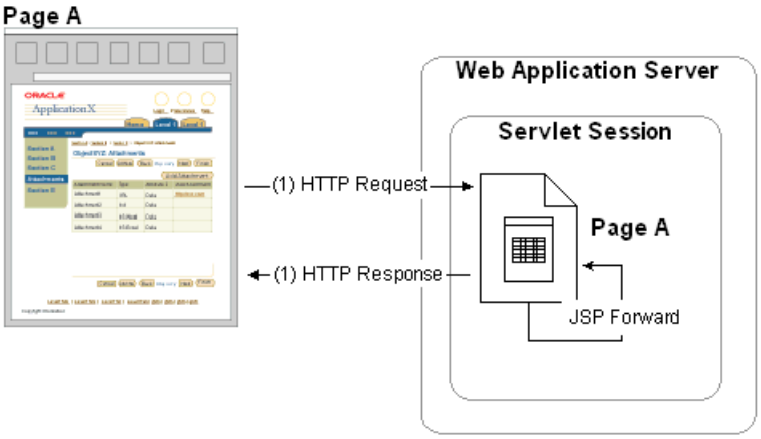
In this case, while handling an incoming request as a result of the user's action selection, JSP A "forwards" to JSP B which does its work and sends a response including the HTML to display itself. Since the "forward" action happens on the server, the browser knows nothing about it and the two pages share the same request.

Figure 3: JSP forward from one page to the next within a single request



In another variation, which is very common in OA Framework pages for reasons, which will be described later in this chapter and the next, Page A could perform a JSP Forward back to itself as shown below.

Figure 4: JSP forward from one page back to itself within a single request



Client Redirect

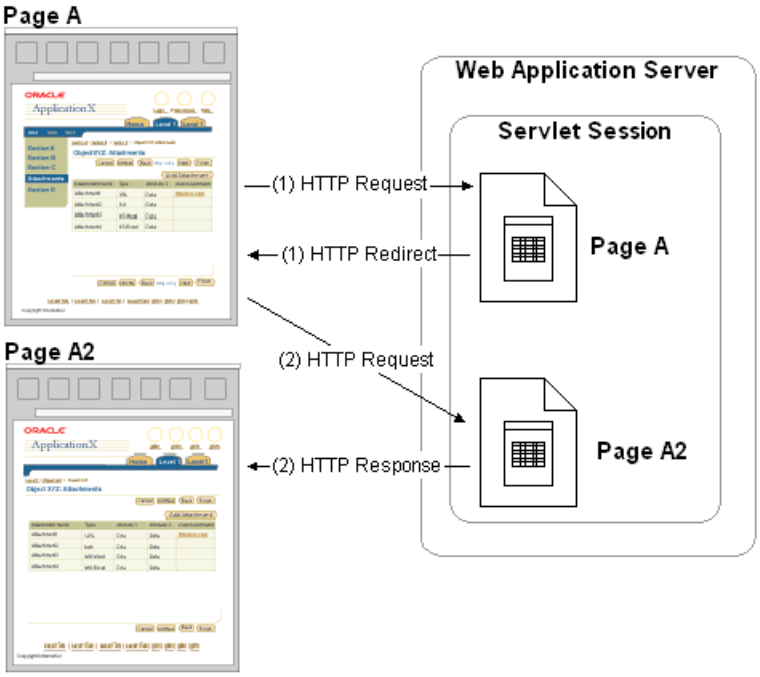
Scenario

A user selects a link on Page X to navigate to Page A, but the link is old so the developer wants to automatically send the user to the new replacement, Page A2.

Implementation

In this case, while handling an incoming request, JSP A sends a special "redirect" message to the browser telling it to immediately access JSP A2. The browser sends a *second* request to JSP A2, which does its work and sends a response including the HTML to display.

Figure 4: A client redirect from one location (page) to the next (the same page in a different location)



What is a Cookie?

To fully understand how the OA Framework maintains application context after a user logs in, you need to understand what a browser "cookie" is.

A "cookie" is a nugget of information that a web application can give to a browser with the understanding that the browser will send it back to the server with each request. In other words, it is a mechanism for holding on to some small amount of state between requests.

Cookies can be persistent or session-based:

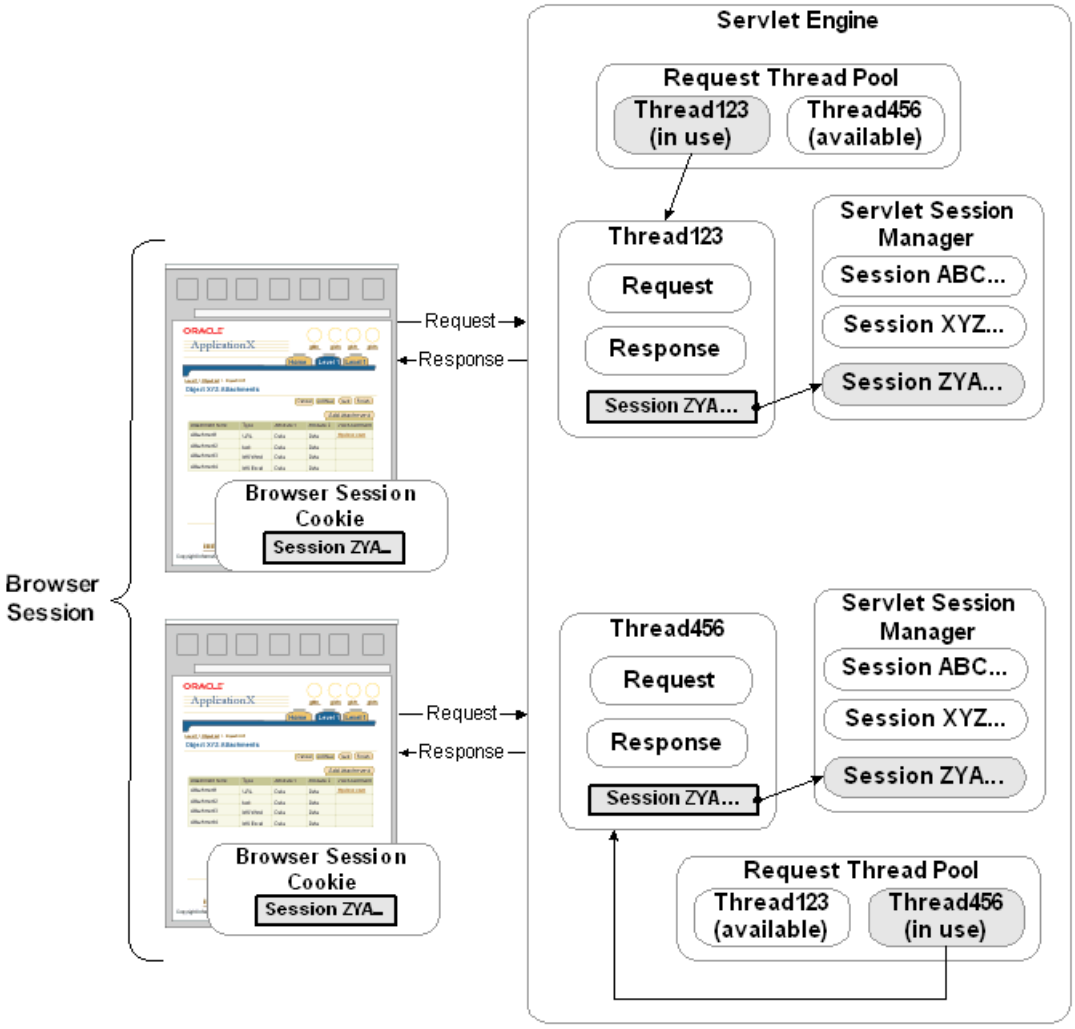
- The browser saves a persistent cookie to a file on the user's computer, and the information endures across browser sessions. Have you ever navigated to a web site that greeted you by name before you logged in? If so, this was accomplished with a persistent cookie.
- Session-based cookies are held in the browser's memory, and when the browser is closed, the cookie is destroyed.

More about Servlet Sessions

In the same way that AOL/J pools JDBC connections because they are a precious resource (you will learn more about connection pooling later in this chapter), the servlet engine pools request-processing threads. As illustrated in Figure 5 below, the servlet engine allocates a thread to process each request it receives. When the request completes, the servlet engine returns the thread to its pool.

Note: The following diagram assumes a user performs two actions resulting in two separate HTTP requests while working in the *same browser window* (the same browser session). It should not be interpreted to mean that two browser windows are open.

Figure 5: Conceptual illustration of servlet engine request processing



Since a single browser session can be served by numerous threads, (a different one for each request), the servlet session provides a resource for maintaining state across requests.

- If a web application wants to establish a servlet session, it calls a method on the request object asking for a session to be created.
- The servlet engine creates the session (specifically, a javax.servlet.http.HttpSession object), along with a special cookie that it returns to the browser with the response. This session cookie holds the servlet session ID.
- When a new request is received with the session ID cookie, the servlet engine uses this ID to locate that particular browser's servlet session object.
- Web application code can then access any data stored on the servlet session during previous requests within the same browser session.

Note: You can track sessions two ways. The most common way, which is what the OA Framework does, is to use a session cookie. Alternatively, you can encode the cookie into request URLs. If you want to learn more about this, or any other concepts presented in this document, see the suggested reading section above.

Anatomy of an OA Framework Page

This document describes the basic elements of a typical OA Framework page.

Contents

- Page Basics
- The Model
- The View
- The Controller
- Web Bean Architecture
- Guide to OA Framework Javadoc

Prerequisite Reading

If you are new to web application development, please read the short JSP Application Primer before proceeding. The OA Framework Developer's Guide assumes you are familiar with the concepts and vocabulary presented in the JSP Primer.

Page Basics

At the browser level, an OA Framework page, like any other web page, renders as standard HTML.

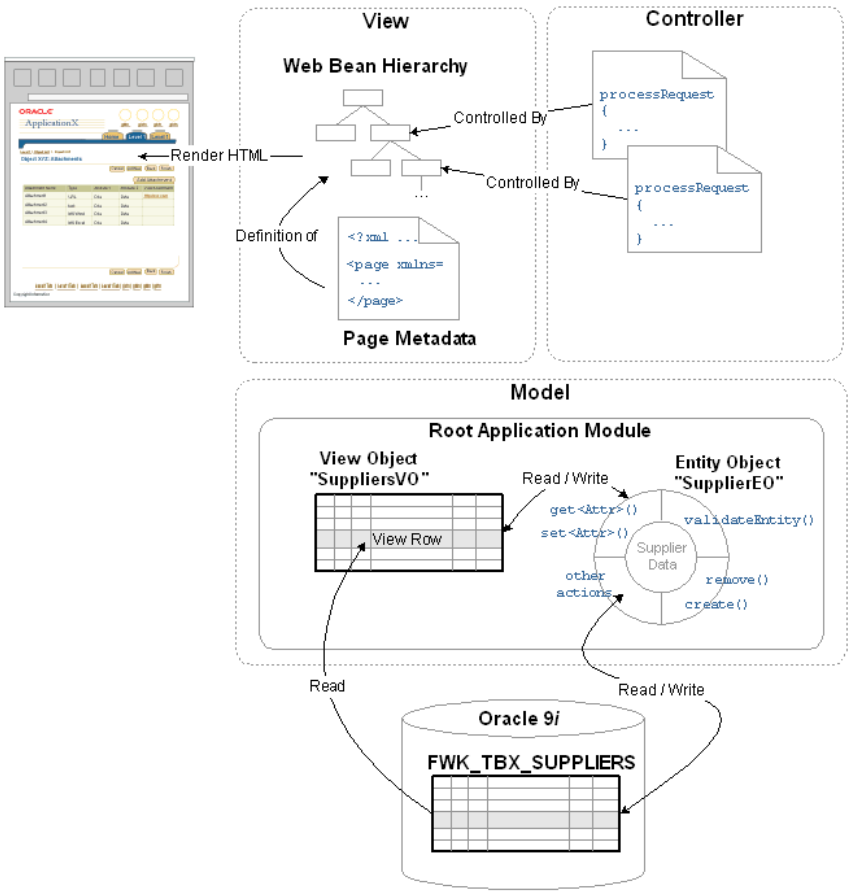
In the middle tier, however, this page is implemented in memory as a hierarchy of Java beans -- very much like a classical Java client UI. Each UI widget, such as buttons, a table, the tabs, the application branding image and so on, that renders in the page corresponds to one or more web beans in the hierarchy.

When the browser issues a request for a new page, OA Framework reads the page's declarative metadata definition to create the web bean hierarchy. For each bean with an associated UI controller, OA Framework calls code that you write to initialize the page. When page processing completes, OA Framework hands the web bean hierarchy to the UIX framework so it can generate and send HTML to the browser.

When the browser issues a form submit (if, for example, the user selects a search region's **Go** button), OA Framework recreates the web bean hierarchy if necessary (the hierarchy is cached between requests, and typically needs to be recreated only in exceptional cases that we'll discuss in detail later), and then calls any event handling code that you've written for the page beans. When page processing completes, the page HTML is generated again and sent to the browser.

The rest of this document introduces the specific model, view and controller code and declarative definitions that you create to implement a page.

Figure 1: A conceptual illustration of the OA Framework model-view-controller architecture



The Model

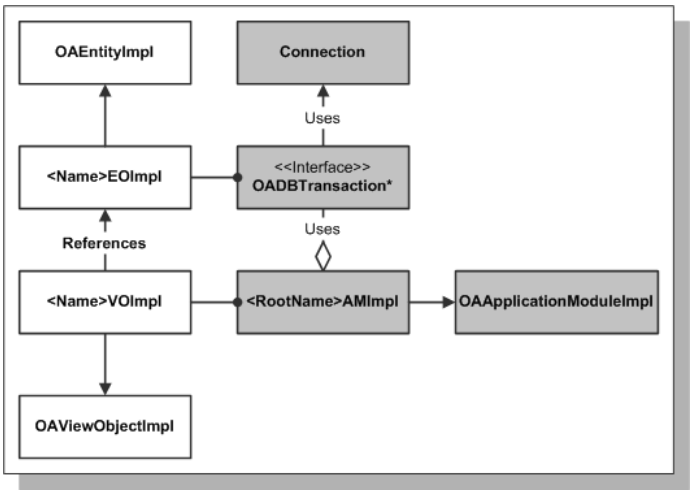
The model encapsulates the underlying data and business logic of the application. It also provides an abstraction of the real-world business object(s) and application service(s) that it provides.

The Implementing the Model document in Chapter 3 discusses all of the following in detail.

Application Modules

A BC4J application module is essentially a container that manages and provides access to "related" BC4J model objects. In this context, objects are "related" by virtue of participating in the same task. For example, all the BC4J objects that comprise a single transaction participate in the same task -- even if the corresponding user interface requires that the user visit multiple pages.

Figure 2: Basic model architecture - Application Module associations



All application modules that you create subclass the `oracle.apps.fnd.framework.server.OAApplicationModuleImpl` class.

Each OA Framework page has a "root" application module, which is associated with the top-level page region (the **pageLayout** region). The root application module provides transaction context and establishes a database connection.

- If multiple pages participate in the same physical or virtual transaction, they should share the same root application module.
- If a page functions independently of any other, it should have its own application module.

The OA Framework State Management document in this chapter discusses the relationship between root application modules and different kinds of pages in detail.

Note: It is also possible for a root application module to contain one or more "nested" application modules, which can themselves nest children to any arbitrary level. In this scenario, the root application module has access to all the data/objects held by its children, and all children participate in the same transaction established by the root. You will use this feature whenever you want to create a reusable UI region that interacts with the database.

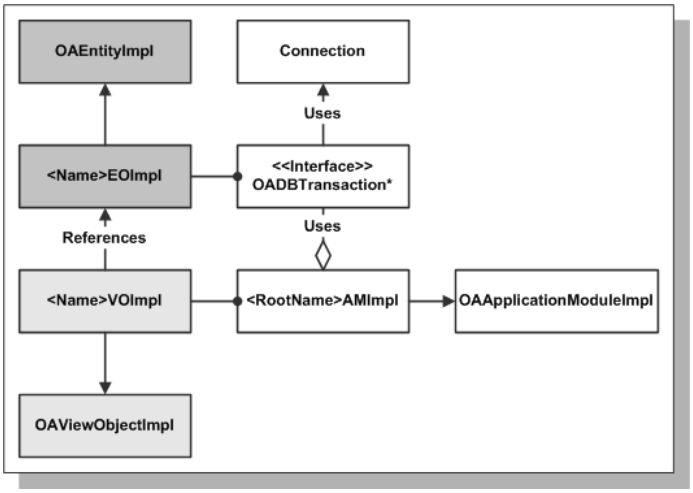
Entity Objects (and Association Objects)

BC4J entity objects encapsulate the business rules (validations, actions and so on) associated with a row in a database table. For example, the OA Framework ToolBox Sample Library includes a `FWK_TBX_SUPPLIERS` table for storing supplier definitions. We also defined an entity object for this table (`SupplierEO`) that implements all the business rules for inserting, updating and deleting a supplier.

Note: Entity objects can also be based on views, synonyms or snapshots.

OA Framework supports both Java and PL/SQL entity objects (Chapter 5 discusses business logic design and implementation in detail, including advice on choosing Java versus PL/SQL entity objects).

Figure 3: Basic model architecture - Entity Objects associations



Most entity objects that you create subclass the `oracle.apps.fnd.framework.server.OAEntityImpl` class (you will see a bit later that the PL/SQL entity objects extend specialized versions of `OAEntityImpl`).

There is a one-to-one mapping between a table and an entity object, and all Oracle E-Business Suite entity objects should include all columns in their associated

tables. Entity objects use a declarative mapping between their attributes and underlying database columns to automatically implement queries, inserts, updates and deletes. In most cases, all you need to do is add the validation logic.

An entity object is intended to be used by *any* program (not just an OA Framework client) that needs to interact with a given table. As such, it should consolidate all the validation logic for the entity object so business rules are consistently implemented regardless of which client exercises them.

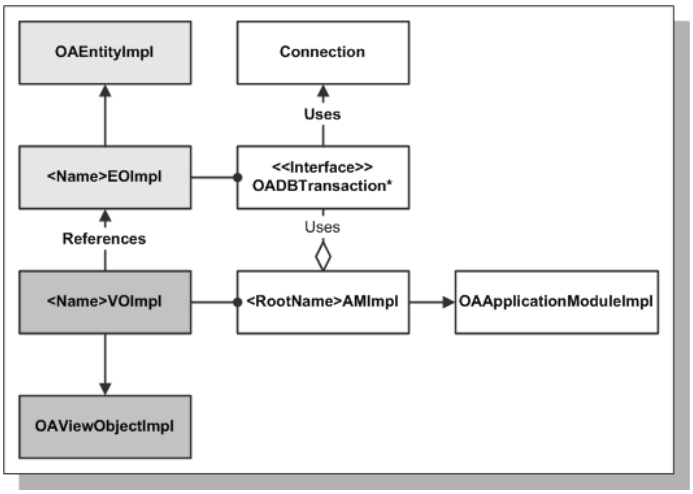
Association Objects

If you have complex objects (like a 3-level purchase order with a 1:many relationship between headers, lines and shipments) you can also define relationships between the corresponding entity objects by creating *association objects*. You can create weak associations (a purchase order header "references" a supplier which exists independently of any given purchase order) and strong composition associations (a purchase order header "owns" its lines, which cannot exist outside the context of their header).

View Objects (and View Links)

In the simplest terms, a BC4J view object encapsulates a database query. After a query is executed, a view object provides iteration over and access to its result set. The result set contains one or more view rows, where a view row comprised of individual attributes corresponds to a row returned by a database query.

Figure 4: Basic model architecture - View Objects associations



All view objects that you create subclass the `oracle.apps.fnd.framework.server.OAViewObjectImpl` class.

Each view object can be configured to query data using one of the following strategies:

- Its attributes map to columns in a simple SQL statement (commonly used for small, read-only view objects)
- Its attributes map to entity object attributes (used to insert, update and delete entity objects)
- Some attributes map to entity objects, and some are populated directly using SQL (used to augment the entity object data with transient columns that cannot be queried via an entity object -- a calculated value used exclusively for UI display purposes is a common example)

In an OA Framework application, you will use view objects in each of the following scenarios (all of which are fully described in later topics):

- Present data that is optimized for a particular user interface. If the user interface supports entity object inserts, updates and deletes, you will interact with the view object to perform these tasks.
- Create simple queries for poplists, lists of values and other supporting UI components
- Create efficient "validation queries" that can be used in your business logic. For example, in a purchase order header entity object you might use a validation view object to get the current maximum purchase order line number from the database so it can cache and increment this value as new lines are created.

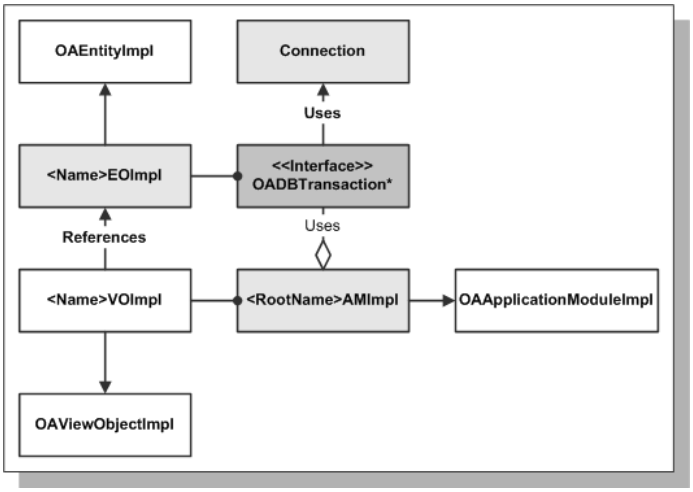
Finally, you will not only define view objects declaratively, but you will write code for them. In a typical case, the code will implement data bindings for complex queries and execute the query (so a view object knows how to "query" itself).

View Links

Just as you can relate entity objects, you can also create view object associations called *view links*. For example, you can create a view link between a purchase order header view object and a purchase order lines view object. This can be used at runtime to automatically query the lines when a header is accessed.

OADBTransaction

Figure 5: Basic architecture model - OADBTransaction



*** Note:** To be completely accurate and consistent, this diagram should include the implementation `oracle.apps.fnd.framework.server.OADBTransactionImpl` instead of the `oracle.apps.fnd.framework.OADBTransaction` interface, however, we have chosen to include latter since you will exclusively use the interface in your code.

As shown in the diagram above, the `OADBTransaction` plays a central role in your model code since it encapsulates the JDBC connection/database session associated with a root application module, and directly owns any entity objects that you create (your view objects, owned by the root application module, hold references to their entity objects in their view rows). You will also make regular use of the `OADBTransaction` in your model code for the following common actions:

- Creating a callable statement for executing PL/SQL functions and procedures
- Accessing session-level Applications context information like the user's name, id, current responsibility and so on
- Accessing an `oracle.apps.fnd.framework.OANLSServices` object if you need to perform NLS operations like converting server date/time into user date/time and so on

Access to the `OADBTransaction` is provided by the root application module.

The View

The view formats and presents model data to the user.

The Implementing the View document in Chapter 3 discusses all of the following in detail.

Defining the Page

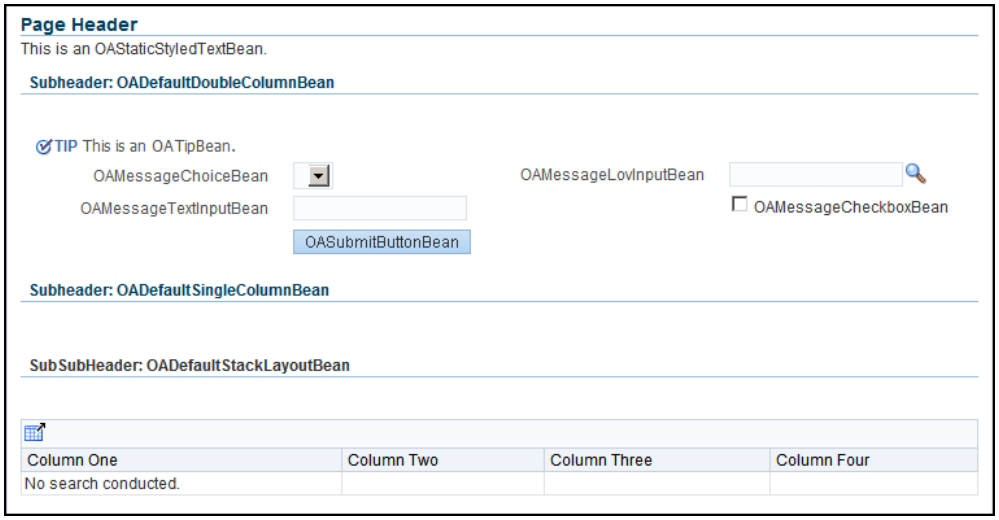
At development time, you specify the bean hierarchy for every page using the declarative JDeveloper tool that we introduced in Building "Hello, World!". In Oracle E-Business Suite development, you will work with (and source control) XML file page definitions. When your product is deployed at a customer's site, the OA Framework runs the page definitions out of a database repository.

To quickly recap, you use JDeveloper to define *pages* comprised of *regions* and *items*.

- Items are simple widgets like buttons, fields, images and so on which contain no children.
- Regions are container objects that can hold items and other regions. Examples of regions include headers, tables, and special layout components.
- Each region and item that you define has a *style* property that tells the OA Framework what web bean object to instantiate for it at runtime (and this in turn dictates what HTML is generated for the bean). For example, if you define a region whose style property is "table," the OA Framework will instantiate an `oracle.apps.fnd.framework.webui.beans.table.OATableBean`.
- All pages must have a single top-level region (often called the "root region") whose style is **pageLayout**. This is instantiated as an `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`.
- The sequence in which regions and items appear in JDeveloper page tree (and the corresponding XML file) tells the Framework where to add these objects to the runtime bean hierarchy.

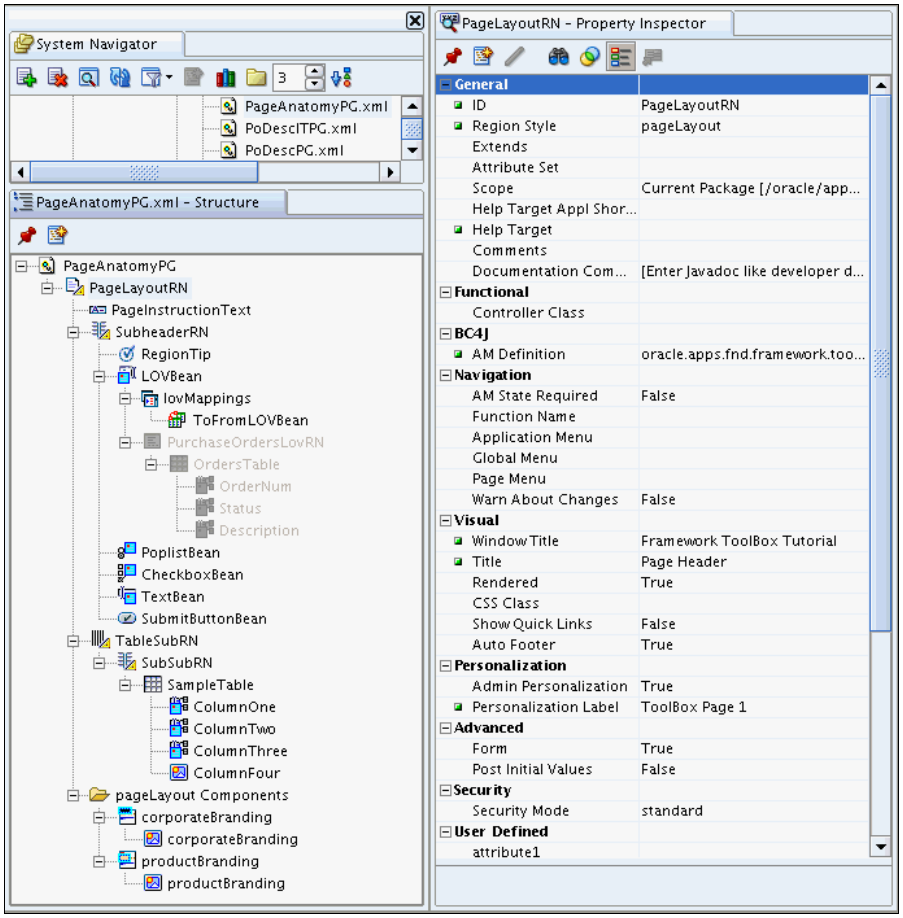
Figure 6 below gives a behind-the-scenes look at the kinds of web beans that are instantiated for a simple page. The labels that you see name the underlying web beans. For example, a poplist is instantiated as an `oracle.apps.fnd.framework.webui.beans.message.OAMessageChoiceBean`, and a submit button is instantiated as an `oracle.apps.fnd.framework.webui.beans.form.OASubmitButtonBean`. Figure 7 shows the corresponding page definition.

Figure 6: Page with UI components showing names of corresponding web beans



Note: The region and item names shown below do NOT comply with the Oracle E-Business Suite naming standards; instead, they are intended to help you translate from the structure to the corresponding web beans.

Figure 7: Page structure in JDeveloper



Attribute Sets

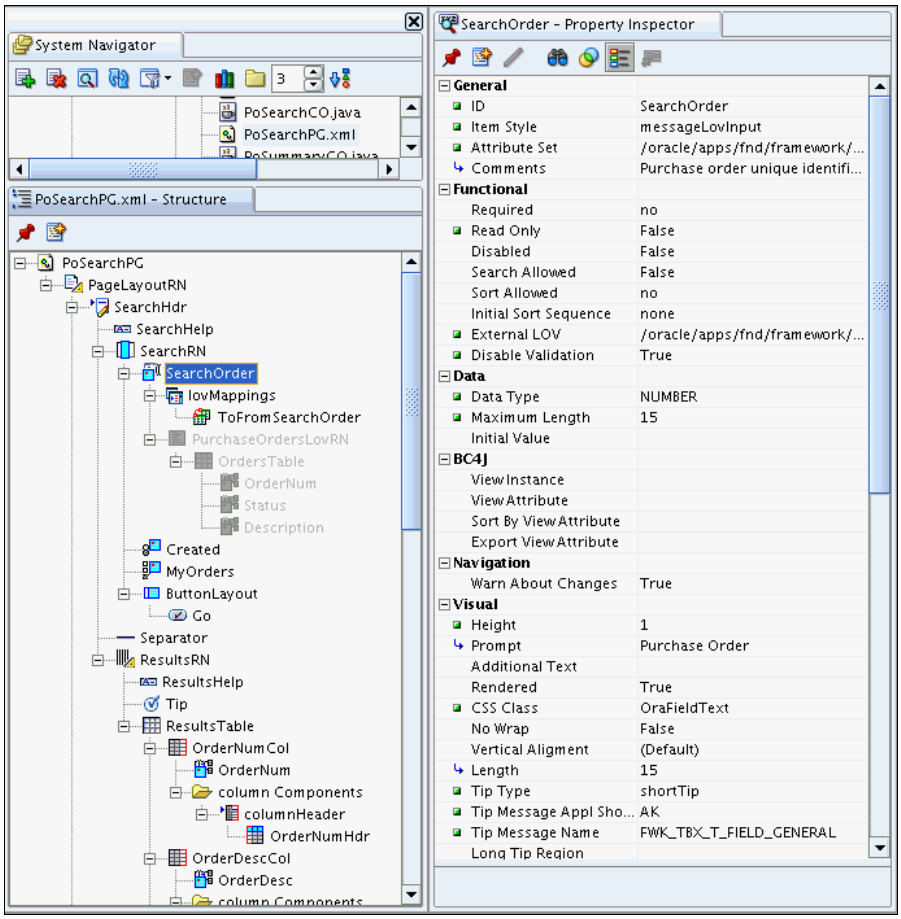
Each region or item can inherit groups of property settings by using attribute sets. An *attribute set* is a named, reusable collection of properties that can be used by any type of UI object, including regions, items, and other attribute sets. Whenever you create a UI that uses attribute sets, you can override the inherited properties (although this is discouraged in the OA Framework coding standards).

To illustrate this concept, in Applications development, each table must have associated attribute sets for each displayable column. These attribute sets include properties like prompt, display width, and so on.

- In the OA Framework ToolBox Sample Library/Tutorial, we have a purchase orders table (`FWK_TBX_PO_HEADERS`) with a primary key column `HEADER_ID` of type `NUMBER` that is also displayed to users as the purchase order number.
- This table has an associated attribute sets XML package file called **FwkTbxPoHeaders** that includes all the attribute sets for the table's displayable columns (one attribute set per column). One of the attribute sets is called **HeaderId**.

- The **HeaderId** attribute set has the `Prompt` property set to **Order Number** and the `Display Length` set to something reasonable like **15**.
- When we create a page that includes the purchase order number item, we would specify the Attribute Set property to the fully qualified attribute set name /oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxPoheaders/Headerid

Figure 8: Using an attribute set in JDeveloper



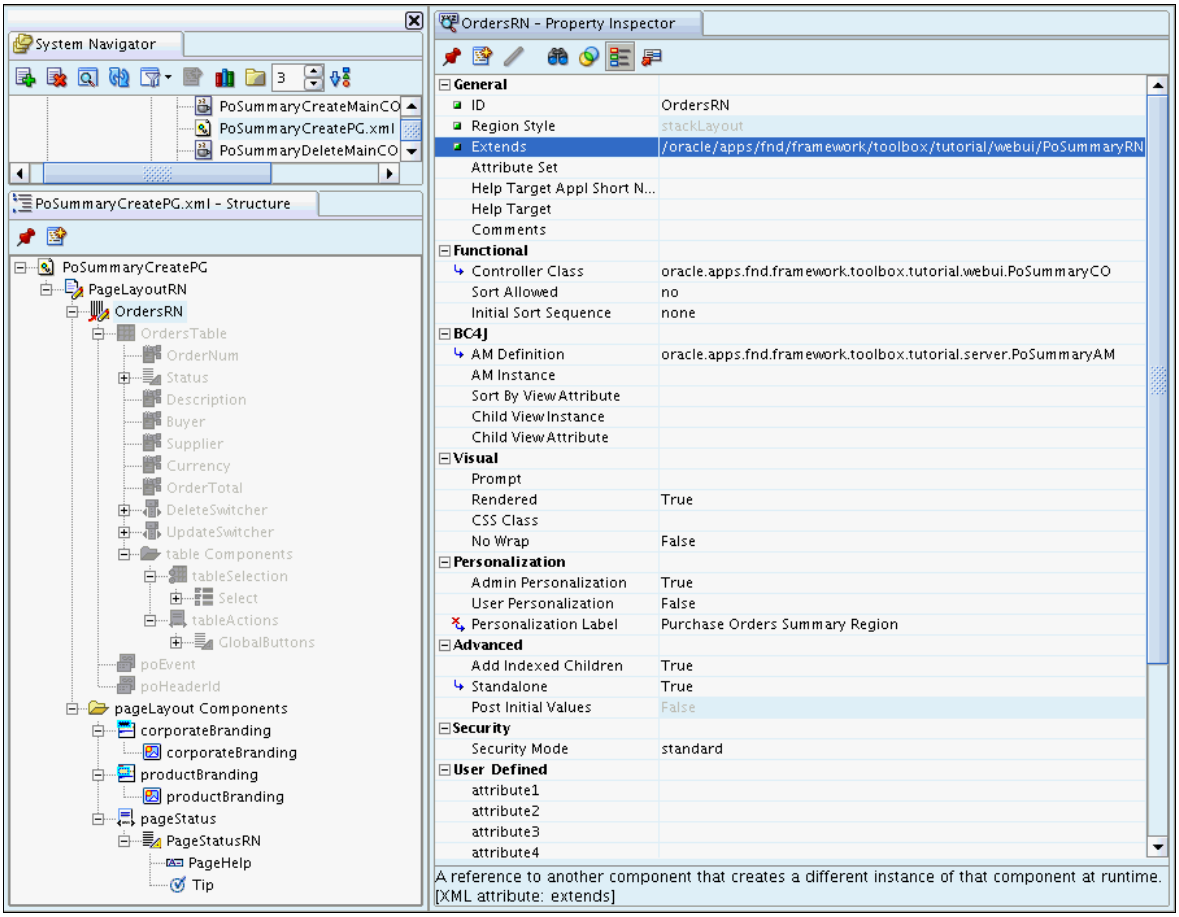
Component Reuse

If you want to incorporate shared objects into your page, you can simply *extend* them.

For example, in the OA Framework ToolBox Sample Library/Tutorial we created a common region (named **PoSummaryRN**) so the same content could be included in multiple pages without recoding. To add this shared region to a page, we simply created a new region, and then set its `Extends` property to the fully qualified name of the shared region: /oracle/apps/fnd/framework/toolbox/tutorial/webui/PoSummaryRN

Note: The shared region is not editable in the referencing page, so its items are grayed out in the JDeveloper Structure pane.

Figure 9: Extending a region JDeveloper



Data Source Binding

For beans with any database interaction (query, insert, update and/or delete), you also specify a data source binding to a `View Instance Name` and associated `View Attribute Name`. This binding is crucial because the OA Framework uses it to get queried data from, and write user-entered data to, the underlying view object.

- The `View Instance Name` references the underlying view object within the context of its containing application module (all view objects "live" within an application module and are identified by an instance name within its container). For example, if a `SuppliersVO` view object is identified by the instance name "MySupVO" within your page's root application module, "MySupVO" is the name you would specify here.
- The `View Attribute Name` references the underlying view object attribute that maps to a column. For example, if your `SuppliersVO` has an attribute "SupplierId" (which maps to the underlying column `SUPPLIER_ID`), "SupplierId" is the name you would specify here.

Defining the Menu

All OA Framework applications include menus as described in the Oracle Browser Look and Feel (BLAF) UI Guideline: Tabs/Navigation. You define these menu structures declaratively using Oracle E-Business Suite menu and function definition forms. We'll discuss this in detail later in the Developer's Guide.

Just as the OA Framework translates your declarative UI layout into the runtime bean hierarchy, it also includes web beans for the declarative menu definition.

Defining Page Flow

When dealing with multipage transaction flows, the OA Framework provides a declarative (thereby customizable) alternative to complex, hard-coded controller logic. See Chapter 4: Declarative Pageflow Using Workflow for additional information about this feature.

Personalizing Pages

The OA Framework also includes a declarative customization infrastructure called the OA Personalization Framework. This is intended to support the customization needs of end users and the product delivery chain (changes for localization, verticalization and so on).

Note: As you'll see throughout the Developer's Guide, creating regions and items declaratively is always preferable to creating them programmatically. In fact, you should create components programmatically ONLY if you cannot create them declaratively so customers can personalize your work.

The Controller

The controller responds to user actions and directs application flow.

The Implementing the Controller document in Chapter 3 discusses all of the following in detail.

Controllers can be associated with the view at the region level (in more general terms, any OA Framework web beans that implement the `oracle.apps.fnd.framework.webui.beans.OAWebBeanContainer` interface can have associated controllers).

All controllers that you create subclass `oracle.apps.fnd.framework.webui.OAControllerImpl` as shown in Figure 10 below.

The controller class is where you define how the web beans behave. Specifically, you write controller code to:

- Manipulate/initialize the UI at runtime (including any programmatic layout that you are unable to do declaratively) and
- Intercept and handle user events like button presses

Request Handling

When the browser issues an `OA.jsp` request for one of your pages:

1. The `oracle.apps.fnd.framework.webui.OAPageBean` (the main OA Framework page processing class) uses the page name to determine which root application module it needs so it can check it out from the application module pool. This application module also checks out a JDBC connection from the connection pool, and the transaction context for the page is established.
2. The user session is validated; if invalid, a login page is displayed (note that this is a simplification; additional details are provided later in the Developer's Guide).
3. Assuming the user is valid, the `OAPageBean` evaluates request parameters to figure out if it is dealing with an HTTP `POST` or a `GET`.

Handling a GET Request

When the browser issues a `GET` request to the server for a page (or you manually forward to it), the OA Framework uses the declarative UI definition to build the web bean hierarchy:

1. The `OAPageBean` calls `processRequest()` on the page's top-level **pageLayout** bean, and the entire web bean hierarchy is processed recursively as follows to initialize the web beans (including any associated model components):
    1. Each web bean instantiates its controller -- if it has one -- and calls processRequest(OAPageContext pageContext, OAWebBean webBean) on the controller. This is the method you use to construct/modify your page layout, set web bean properties and do any manual data initialization (if, for example, you need to perform an autoquery when you navigate to the page).
    2. Some complicated web beans (like the `oracle.apps.fnd.framework.webui.beans.table.OATableBean` and `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`) perform post-controller processing by calling their prepareForRendering() methods (this method is described in the corresponding bean Javadoc).
    3. Each web bean calls processRequest() on its children.
2. The `oracle.apps.fnd.framework.webui.OAPageBean` gives the web bean hierarchy to UIX to render and send to the browser.

Handling a POST Request

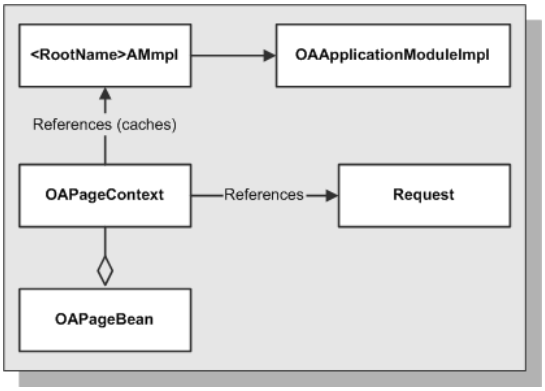
When the browser issues a `POST` request to the server for a page:

1. The `OAPageBean` checks to see if the web bean hierarchy is in memory. If not (because resources were reclaimed, the user navigated with the browser Back button, or a `POST` is issued to the main page from a dialog message page), it recreates the hierarchy as described in the `GET` processing above.
2. The `OAPageBean` calls processFormData(OAPageContext pageContext, OAWebBean webBean)on all the beans in the hierarchy to write the form data to the model (specifically, it calls processFormData() on the **pageLayout** region, and then each web bean recursively calls processFormData() on its children). Writing the form data to the underlying model automatically invokes attribute and entity-level validations, and if you throw any validation exceptions, processing stops and error messages are displayed to the user.
3. If no exceptions are thrown during the processFormData() phase, `OAPageBean` calls processFormRequest(OAPageContext pageContext, OAWebBean webBean) on all the beans in the hierarchy using the same approach described above. This pass gives your controller code the opportunity to respond to user actions.
4. If no JSP forwards or page redirects were issued -- or exceptions were thrown in processFormRequest() -- then the page is refreshed.

OAPageContext

When the OA Framework receives an `OA.jsp` request, the `OAPageBean` creates an `oracle.apps.fnd.framework.webui.OAPageContext`, a class that exists only for the duration of page processing. Each of the three key methods described above (`processRequest()`, `processFormData()` and `processFormRequest()`) takes an `OAPageContext` as a parameter, and any controller code that you write will invariably make use of this crucial class.

Figure 10: Relationship between the OAPageContext and other key classes



As illustrated in the diagram above, the `OAPageContext` has a reference to both the request and the root application module. Given these relationships, and the fact that an `OAPageContext` is passed to each of your controller response-processing methods, you can see how you would use the `OAPageContext` for the following list of common tasks:

Accessing Request Parameters

Perhaps most importantly, this is the class that you use to read request values by calling a simple getParameter(String name) method (remember that the request includes any URL parameters plus -- if it is a `POST` -- any form field values plus the names and events associated with any action/control widgets selected by the user).

Tip: For individual web beans on your page (buttons, fields, and so on) the `name` value passed to getParameter() is the corresponding unique `ID` that you assign when defining your page. So, for example, you can tell if the user pressed a button that you named "GoButton" in JDeveloper by writing the following code in a controller:

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  if (pageContext.getParameter("GoButton") != null)
  {
    // The user pressed the "Go" button, do something...
  }
}
```

Accessing the Root Application Module

The `OAPageContext` caches a reference to the root application module, which in turn provides access to its view objects and the transaction. If you need access to an application module, ask the `OAPageContext`:

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  OAApplicationModule am =
    (OAApplicationModule)pageContext.getRootApplicationModule();
}
```

Issuing Navigation Instructions

You use methods on this class to tell the OA Framework to perform a JSP forward or a client redirect. For example (we'll review this method in greater detail later in the Developer's Guide):

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  if (pageContext.getParameter("CreateButton") != null)
  {
    // The user pressed the "Create Supplier" button, now perform a JSP forward to
    // the "Create Supplier" page.

    pageContext.setForwardURL("OA.jsp?page=/oracle/apps/dem/employee/webui/EmpDetailsPG",
                              null,
                                                  OAWebBeanConstants.KEEP_MENU_CONTEXT,
                                                  null,
                                                  null,
                                                  true, // Retain AM
                                                  OAWebBeanConstants.ADD_BREAD_CRUMB_YES, // Show breadcrumbs
                                                  OAWebBeanConstants.IGNORE_MESSAGES);
  }
}
```

Accessing Application Context Information

Like the `OADBTransaction` in your model code, the `OAPageContext` provides access to servlet session-level Oracle E-Business Suite context information like the user's name, id, current responsibility and so on. For example, the following code snippet shows how to get the user's name:

```
processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  String userName = pageContext.getUserName();
}
```

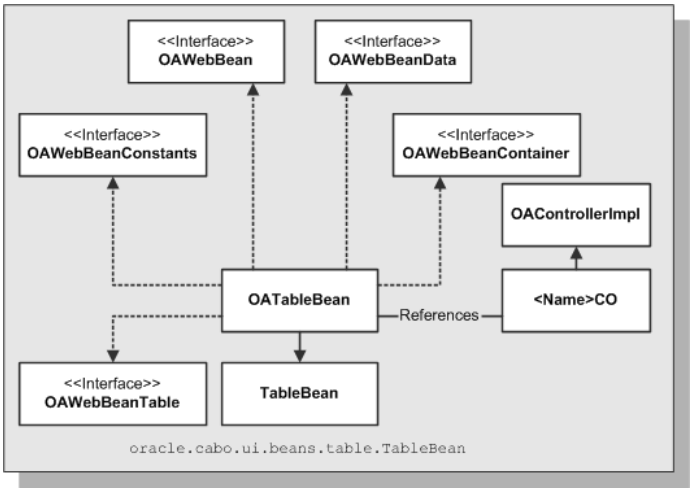
Web Bean Architecture

First and foremost, all OA Framework web beans subclass corresponding beans in the UIX framework. For example, an OATableBean extends an oracle.cabo.ui.beans.table.TableBean ("cabo" was an earlier name for the UIX framework, and the package definitions still use this old name).

Each OA Framework web bean also implements a group of interfaces whose implementations collectively define the behaviors that the OA Framework adds to the base UIX beans.

- `oracle.appps.fnd.framework.webui.beans.OAWebBean` - defines core behavior common to all web beans (for example, among other key behaviors, this defines the `processRequest`, `processFormData`, and `processFormRequest` methods that individual beans implement for themselves).
- `oracle.apps.fnd.framework.webui.OAWebBeanConstants` - a collection of constants used in the view/controller modules
- `oracle.apps.fnd.framework.webui.beans.OAWebBeanData` - defines common personalization definition and data source management behavior
- `oracle.apps.fnd.framework.webui.beans.OAWebBeanContainer` - defines the characteristics of all web beans that can act as containers for other web beans. For instance, all the layout web beans implement this interface. Only beans which implement this interface can have associated controllers.
- **OAWebBean<Type>** - defines a bean's inherent behaviors within the context of the OA Framework. For example, the `OATableBean` implements the `oracle.apps.fnd.framework.webui.beans.OAWebBeanTable` interface.

Figure 11: Example of a container web bean (OATableBean)

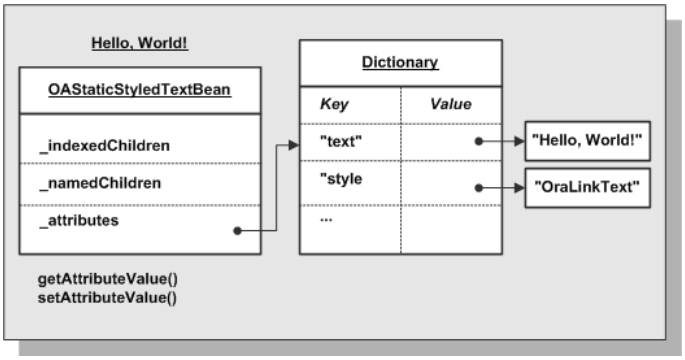


Internal Bean Structure

Each web bean maintains the following information about itself:

- _indexedChildren - child web beans
- _namedChildren - child web beans that UIX tags for special behavior
- _attributes - web bean characteristics (descriptive properties) as illustrated in Figure 12 below

Figure 12: Illustration of web bean use of a Dictionary to track key/value pairs of its attributes



Data Bound Values

Instead of literal values as illustrated in Figure 9 above, OA Framework web bean attributes are actually implemented as data bound values, meaning that the value is provided by an underlying data source that is resolved to the component at rendering time. You will see a bit later how to define and use custom bound values in your code.

Rendering

At page rendering time, the UIX framework processes the web bean hierarchy to generate the page HTML.

For each web bean attribute, UIX calls its `getAttributeValue()` method while passing it a *rendering context* (a rendering context is basically all the information that UIX needs to resolve bound values). For a given attribute, for example, the rendering context knows what the underlying view object instance, view attribute and current row are. The data bound value uses the information supplied by the rendering context to interrogate its data source, and returns the actual value to UIX so it can generate the corresponding HTML.

Guide to OA Framework Javadoc

You've probably already noticed that the Developer's Guide links directly to the OA Framework Javadoc for individual classes. Given that you now have a high-level understanding of the components that you'll be using when you build a page, this section briefly summarizes the purpose of each OA Framework package and describes when you're likely to use the Javadoc for the corresponding classes/interfaces.

oracle.apps.fnd.framework

Contains classes and interfaces which can be safely accessed from model (server) and user interface controller or view (client) code. For example, if you need to access a root application module in your page, you will use the `oracle.apps.fnd.framework.OAApplicationModule` interface (you will never access an implementation on the client).

Among other things, this package also includes:

- All OA Framework exceptions that you might have occasion to throw
- The `OANLSServices` class that you will use to perform internationalization operations

oracle.apps.fnd.framework.server

Contains classes and interfaces for implementing the model in an OA Framework Model-View-Controller application.

These classes are intended to be used with any client user interface (not just OA Framework HTML pages), and as such should have no direct references to any classes and interfaces published in the oracle.apps.fnd.framework.webui package and subpackages, or in any application-specific webui packages and subpackages.

When building an OA Framework application model, you should always work with the classes in this package instead of the BC4J classes they extend.

Warning: Never call any classes in this package from controller or view code.

oracle.apps.fnd.framwork.webui

Contains core classes for building and manipulating OA Framework HTML user interfaces.

Some of the most commonly used classes/interfaces in this package include:

- OAController
- OAPageContext
- Any class in the beans subpackages described below

Warning: Never call any classes in this package from model code.

oracle.apps.fnd.framework.webui.beans

Contains web bean classes for user interface components that don't fit neatly into the various bean subpackages (for example: image, switcher, static instruction text, key flexfield, descriptive flexfield and more). You should use these classes when writing a user interface controller that needs to programmatically manipulate the web beans.

This package also contains core OA Framework interfaces implemented by all web beans.

The classes in this package and its subpackages correspond to the UIX components they extend as shown below. When building OA Framework application pages, you should always work with the OA Framework classes unless a new feature that you want to use has been introduced in UIX, and is not yet supported by the framework.

Note: OA Framework classes are always instantiated for MDS pages that you build declaratively in JDeveloper.

UIX Package	OA Package
oracle.cabo.ui.beans	oracle.apps.fnd.framework.webui.beans
oracle.cabo.ui.beans.form	oracle.apps.fnd.framework.webui.beans.form
oracle.cabo.ui.beans.include	oracle.apps.fnd.framework.webui.beans.include
oracle.cabo.ui.beans.layout	oracle.apps.fnd.framework.webui.beans.layout
oracle.cabo.ui.beans.message	oracle.apps.fnd.framework.webui.beans.message
oracle.cabo.ui.beans.nav	oracle.apps.fnd.framework.webui.beans.nav
oracle.cabo.ui.beans.table	oracle.apps.fnd.framework.webui.beans.table

Warning: Never call any classes in this package from model code.

oracle.apps.fnd.framework.webui.beans.form

Contains web bean classes for HTML form components including a submit button and various data entry/specification controls (checkbox, radio group, shuttle, text input field and more). You should use these classes when writing a user interface controller that needs to programmatically manipulate the web beans.

For many of the web beans in this package there are variants in the oracle.apps.fnd.framework.webui.beans.message package (the message web beans have the ability to display error, information, and warning icons with an explanatory message whereas corresponding data entry/specification web beans in this package do not). When you create your pages declaratively in JDeveloper, the OA Framework automatically instantiates message beans for any of those components that exist in both packages. You should use the classes in this package only in the following cases:

- The class doesn't have a message bean alternative (for example, the `OASubmitButtonBean` exists only in this package)
- You cannot use the message bean alternative

Warning: Never call any classes in this package from model code.

oracle.apps.fnd.framework.webui.beans.include

Contains web bean classes for including user interface fragments from external sources (servlets, JSP pages, and plain HTML) in OA Framework application pages. You should use these classes when writing a user interface controller that needs to programmatically manipulate the web beans.

Warning: Never call any classes in this package from model code.

oracle.apps.fnd.framework.webui.beans.layout

Contains web bean classes for laying out content in an OA Framework application page, including special layout components like hide/show, content containers, bulleted lists, headers, standardized templates for single/double column layouts and more. You should use these classes when writing a user interface controller that needs to programmatically manipulate the web beans.

Warning: Never call any classes in this package from model code.

oracle.apps.fnd.framework.webui.beans.message

Contains web bean classes for HTML form data entry/specification components that are capable of displaying associated error, warning or information icon(s) with an explanatory message (for example, if a user enters the wrong value in a text input field an error icon renders next to its prompt). You should use these classes when writing a user interface controller that needs to programmatically manipulate the web beans.

Many of the web beans in this package are also included in the oracle.apps.fnd.framework.webui.beans.form package without the ability to display the supplemental message icons and text. When you create your pages declaratively in JDeveloper, the OA Framework automatically instantiates message beans for any of those

components that exist in both packages. You should use the classes without the message capability only if you cannot include message beans in your page.

Warning: Never call any classes in this package from model code.

oracle.apps.fnd.framework.webui.beans.nav

Contains web bean classes for HTML user interface navigation components (links, trees, menu elements, quick links, breadcrumbs and more). You should use these classes when writing a user interface controller that needs to programmatically manipulate the web beans.

Warning: Never call any classes in this package from model code.

oracle.apps.fnd.framework.webui.beans.table

Contains web bean classes for tables (which present data to the user in a tabular format) and HGrid components (a hybrid of tabular data display with treelike hierarchical structure). You should use these classes when writing a user interface controller that needs to programmatically manipulate the web beans.

Warning: Never call any classes in this package from model code.

oracle.apps.fnd.framework.webui.laf

Contains utilities that can be used for controlling HTML rendering characteristics including the page's look-and-feel and context-specific behavior (for example, content can be optimized for printing versus display in a regular browser or in an e-mail).

Warning: Never call any classes in this package from model code.

OA Framework State Management

Overview

This document describes the OA Framework state management architecture, including the mechanisms used for caching application custom data and communicating values from one page to the next.

Contents

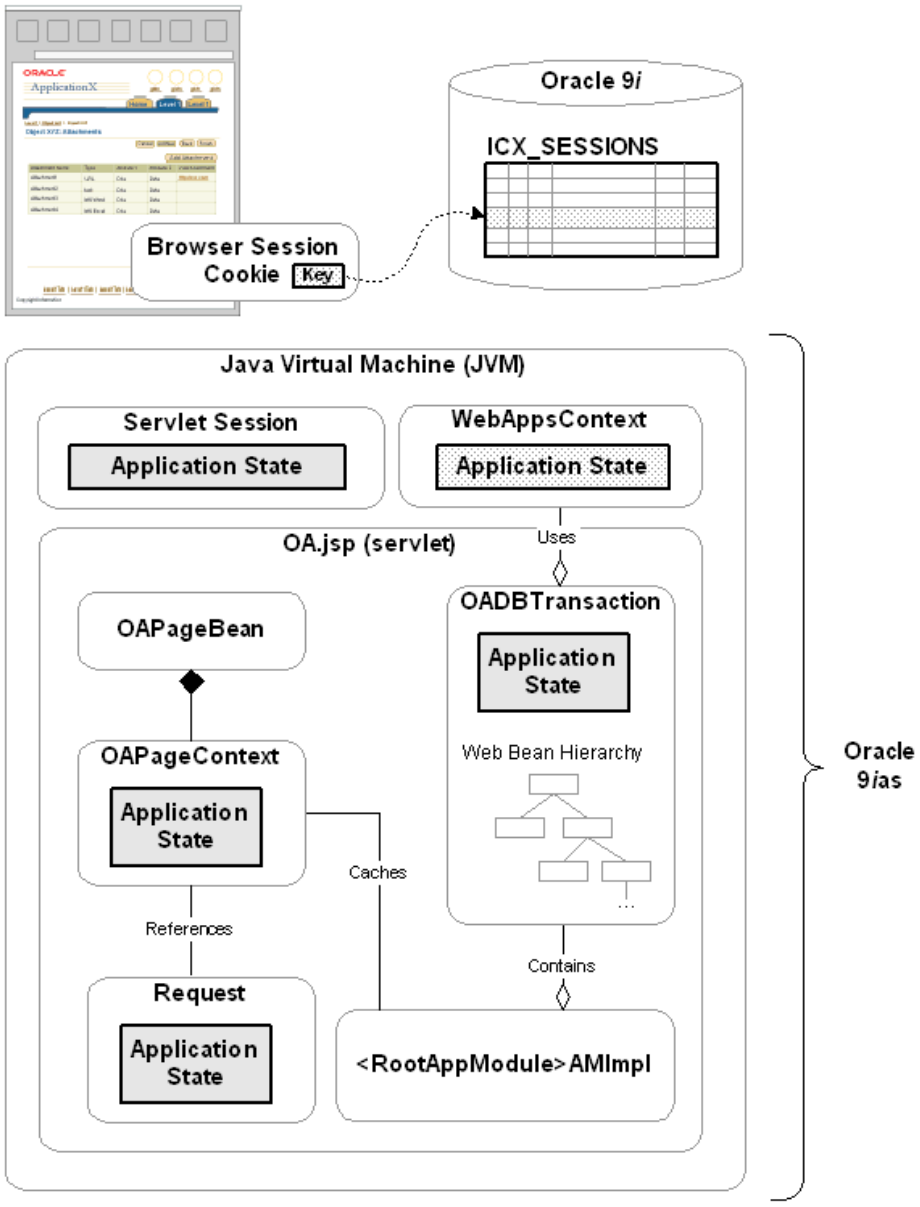
- Architectural Overview
- Root Application Modules (Database Session and Transaction State)
- Servlet Session
- Oracle E-Business Suite User Session
- Page Context
- Request
- Application Module Pooling

Architectural Overview

Figure 1 provides a conceptual application developer's view of the OA Framework state management components. Each component shown in the illustration is discussed below.

Note: It is not intended to reflect all internal OA Framework details.

Figure 1: OA Framework primary state management components



Root Application Modules (Database Session and Transaction State)

As described in the OA Framework Page Anatomy document, each OA Framework page is associated with a root application module that provides its transaction context and JDBC database connection.

Note: In terms of OA Framework, a database session is associated with a JDBC connection.

The root application module is the backbone of any OA Framework module because the core application data (as stored in BC4J view objects, entity objects, and so on) and the page's web bean hierarchy are automatically cached on the root application module's `oracle.apps.fnd.framework.OADBTransaction` object.

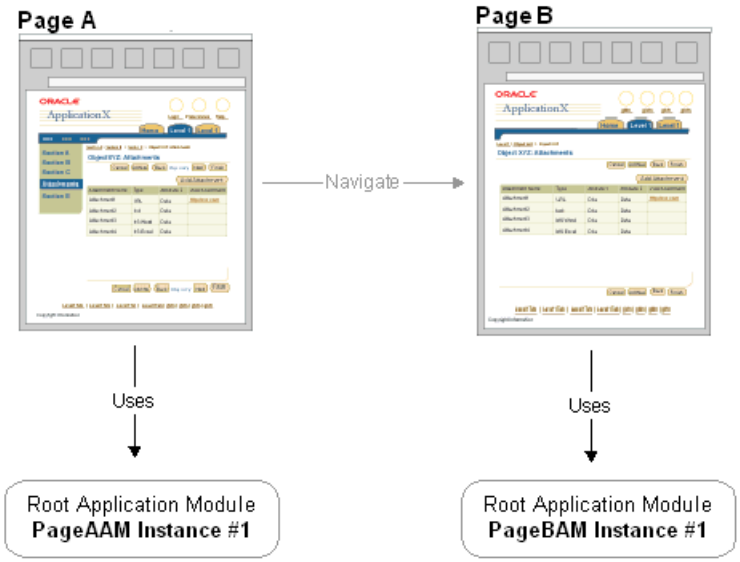
Warning: The use of the browser Back button can cause the loss of application module state. Be sure to review the advanced topic Supporting the Browser Back Button before you start coding. The OA Framework coding standards published in Chapter 8 also include specific recommendations for dealing with this.

Any data stored on the transaction is accessible to all pages that share the same root application module instance (assuming that navigation between them involves retaining this application module as described below). OA Framework provides methods that you can use to store, retrieve and remove custom application values to/from a transaction. Since a single transaction can be accessed from both controller (client) and model (server) code, these utilities are provided in both the `oracle.apps.fnd.framework.webui.OAPageContext` (for the controller) and `OADBTransaction` (for the model) classes.

Root Application Module Retention

By default , when the user navigates from one page to the next (such as with a GET request or a JSP forward), and OA Framework renders the new page, the application module instance associated with the previous page is "released," and a new instance is requested from an application module pool. (See Application Module Pooling below).

Figure 2: Conceptual illustration of default root application module release when navigating to a new page

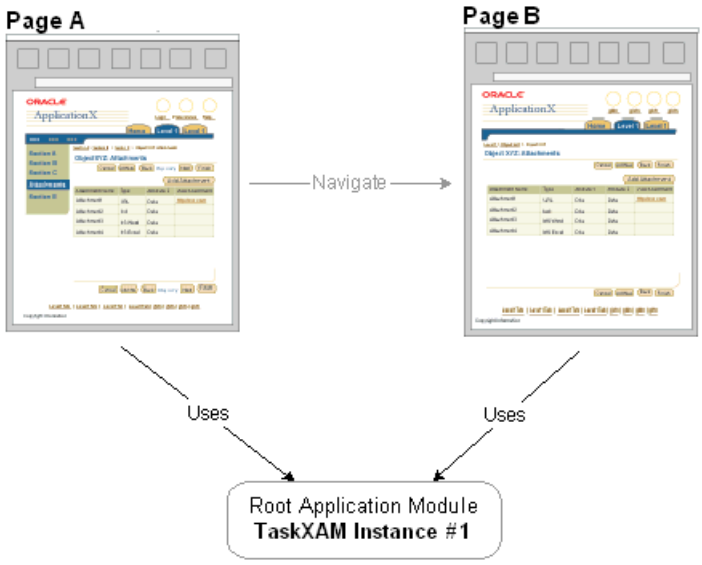


Note: OA Framework never releases application modules during form submit (POST) requests unless you explicitly release the application module in a controller. For example, if a user sorts a table or navigates the result set within a table -- two actions that implicitly submit the page form -- the page's root application module instance is automatically retained.

Retaining the Application Module Across Pages

The default behavior as described above is desirable for individual pages that comprise an isolated, complete task. However, it is not appropriate for a multi-page flow that implements a single task, or a series of related pages participating in a virtual transaction. In these cases, the different pages should be associated with the same root application module instance.

Figure 3: Conceptual illustration of related pages sharing a root application module (and transaction)



To achieve this, you must do the following:

- Declaratively associate the same root application module type with each page in the multi-page flow. (See Implementing the View for additional details on specifying this page property in JDeveloper)
- Set the application module retention flag for a page by specifying the URL parameter `retainAM=Y`. For GET requests. This

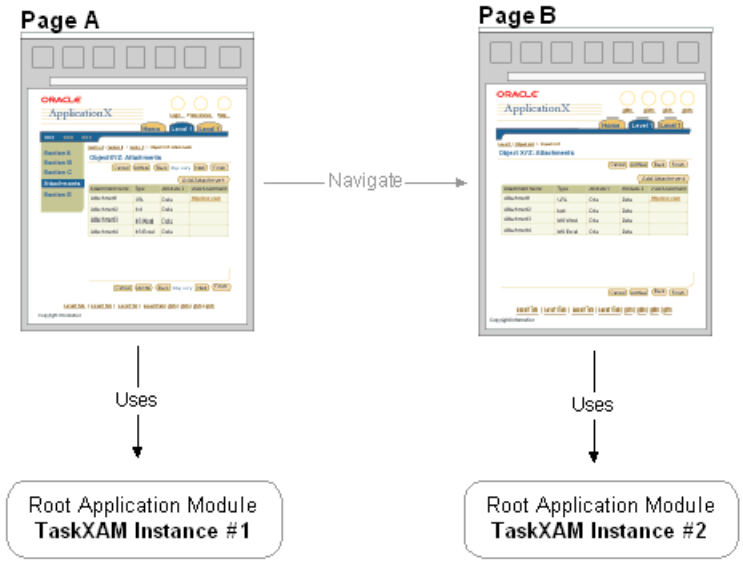
flag is evaluated when a new page is rendered (as mentioned above, OA Framework always retains the application module for POST requests regardless of the `retainAM` parameter value). If set to "Y," the *previous page's* application module instance will be retained. If set to "N" (or not specified, which implies "N"), OA Framework releases all application modules -- including any that might have been explicitly retained before reaching this point.

You also set this parameter when calling JSP forward `OAPageContext` methods. See Implementing the Controller for additional details on controlling this programmatically.

Warning: It is not enough to simply associate the same root application module with each page. If you forget to set the `retainAM` flag, each page will have a *different* application module instance -- and transaction -- even though they are associated with the same application module type.

Note: Technically, depending on the state of the application module pool, Page B could get a reference to the same physical application module instance that Page A used. However, the object's state will be completely reset as if created anew. For the purposes of this discussion, consider it a "new instance."

Figure 4: Conceptual illustration of two pages referencing the same root application module type, but the Retain AM flag is not properly set



Similarly, setting the `retainAM` flag to "Y" -- but associating a *different* root application module type with each of the page -- accumulates several different application module instances (one for each page), each with its own transaction.

Conditionally Retaining/Releasing an Application Module

In some situations, you need to make a conditional determination about whether an application module should be released or not. In these cases, you can implement the `oracle.apps.fnd.framework.webui.OAReleaseListener` interface for individual application modules as described in the Javadoc.

Warning: Oracle E-Business Suite developers should not use this interface without first alerting the OA Framework development team. The incorrect use of this interface can lead to inadvertent memory leaks, and the OA Framework team is currently tracking all implementations.

Explicitly Releasing an Application Module

There are also times when you will want to explicitly release a root application module *before* OA Framework would normally do this. Specifically, when you call the `OAPageContext.releaseRootApplicationModule()` method in one of your page's controllers, OA Framework releases the page's root application module as soon as it finishes rendering the page, instead of waiting until the next application module request.

Root Application Module Retention Use Case Scenarios

The following use cases illustrate recommended approaches to application module retention/release.

Use Case	Recommendation

Unrelated, Discrete Tasks	When navigating between unrelated pages that present complete, discrete tasks do *not* retain the application module. For example, a series of unrelated administration tasks that are typically performed in isolation (even if they are associated with the same menu item) are unlikely to benefit from application module retention.
Multi-page Flow	When navigating between related pages that cooperatively comprise a complete task within a single transaction, retain the application module.
Related Pages (Virtual Transaction)	When navigating between related pages that perform different tasks associated with the *same business object* (even if the pages present different commit points to the user), retain the application module if the pages are closely associated in the UI. For example, a module that lets you query, view, update, delete and print purchase orders benefits from application module retention.
Multi-page Flow with Branch Transaction	Having a multi-page flow with a branch transaction such as, creating a supplier while creating a purchase order, retain the application modules in the main purchase order flow and use the `OAPageContext.releaseRootApplicationModule` method in the Create Supplier page.

Servlet Session

As mentioned in the JSP Application Primer, a servlet session is a mechanism for maintaining state between HTTP requests during a period of continuous interaction between a browser and a web application. A session may be initiated at any time by the application, and terminated by the application by the user closing the browser or by a period of user inactivity. A session usually corresponds to an application login/logout cycle, but that is not strictly true in the case of OA Framework applications. (See the Oracle E-Business Suite User Session below).

You have the option of caching small, serializable objects (the OA Framework restricts this to Strings, Numbers and Dates) on the servlet session; any data that you cache is accessible to all pages rendered within the session. For example, use this approach if you want to use information that is too expensive to retrieve from the database each time it is required.

Note: Use the session cache only when you need to set and access simple values in many pages, which have different root application modules. (The transaction cache discussed above isn't an option). Since the values in the session cache are stored at a transaction level, the values can only be retrieved properly within the same session *and* transaction. Be aware that servlet session values are *not* cleared automatically unless the user logs out, returns to the global Home page to start a new transaction flow, or the servlet session times out. Therefore, you *must* explicitly remove the session values that are no longer needed. For this reason, consider the session a last choice for caching since there is no good event point for freeing memory if the user simply abandons the session without logging out.

Tip: Experienced JSP developers may wonder why hidden form fields aren't recommended instead. Due to the way that OA Framework currently implements menus (some menu selections issue a GET instead of a POST), it is not always possible to add values to the corresponding request when a user navigates by making a menu selection, and root application module boundaries are crossed.

If you want to store, retrieve and remove values from the servlet session, see the `OAPageContext put*()`, `get*()` and `remove*()` methods for session values.

Oracle E-Business Suite User Session

When the user logs in to an OA Framework application, the OA Framework creates an AOL/J `oracle.apps.fnd.common.WebAppsContext` object and a browser session-based cookie that together keep track of key Oracle E-Business Suite context information like the current responsibility, organization id and various user attributes such as user name, user id, employee id and so on.

- The cookie contains an encrypted key identifier for a session row stored in the Applications database. Specifically, this is the servlet session ID which, in its decrypted form, serves as the primary key in the ICX_SESSIONS table.
- The `WebAppsContext` retrieves this key value after each request and uses it to query the current session state.
- The Oracle E-Business Suite user session is associated with a servlet session, however, it has its own life cycle and time-out characteristics. (See Appendix B: Profile Options for additional information about configuring the user session characteristics):
    - Generally, the Oracle E-Business Suite user session has a longer life span than the servlet session. The servlet session should time-out sooner.
    - An Oracle E-Business Suite user session might be associated with multiple servlet sessions. For example, the servlet session times out while the user takes a phone call in the middle of creating an OA Framework expense report, then resumes work before the Oracle E-Business Suite user session times out.
- If the Oracle E-Business Suite user session times out, as long as the user does not close the browser window (so the browser

session-based cookie isn't lost) and no one deletes the corresponding session row in the ICX_SESSIONS table, the user can resume her transaction at the point where she stopped working after being prompted to log back in.

If you need access to any of the information stored with the Oracle E-Business Suite user session, obtain it from `OAPageContext` (in your controller code) or `OADBTransaction` (in your model code).

Applications Context State

You can also use the Applications context to store some state when you don't have access to an OAPageContext (this is the case for Java server tier code or PL/SQL). To do this, use the `WebAppsContext.`setSessionAttribute(java.lang.String pName, java.lang.String pValue) method. For additional information, see the `WebAppsContext` Javadoc.

Page Context

Each time a request is received for a page, OA Framework creates an `OAPageContext` that persists until a new page finishes processing. Specifically, the `OAPageBean` -- which is the primary force behind page processing -- creates the `OAPageContext`.

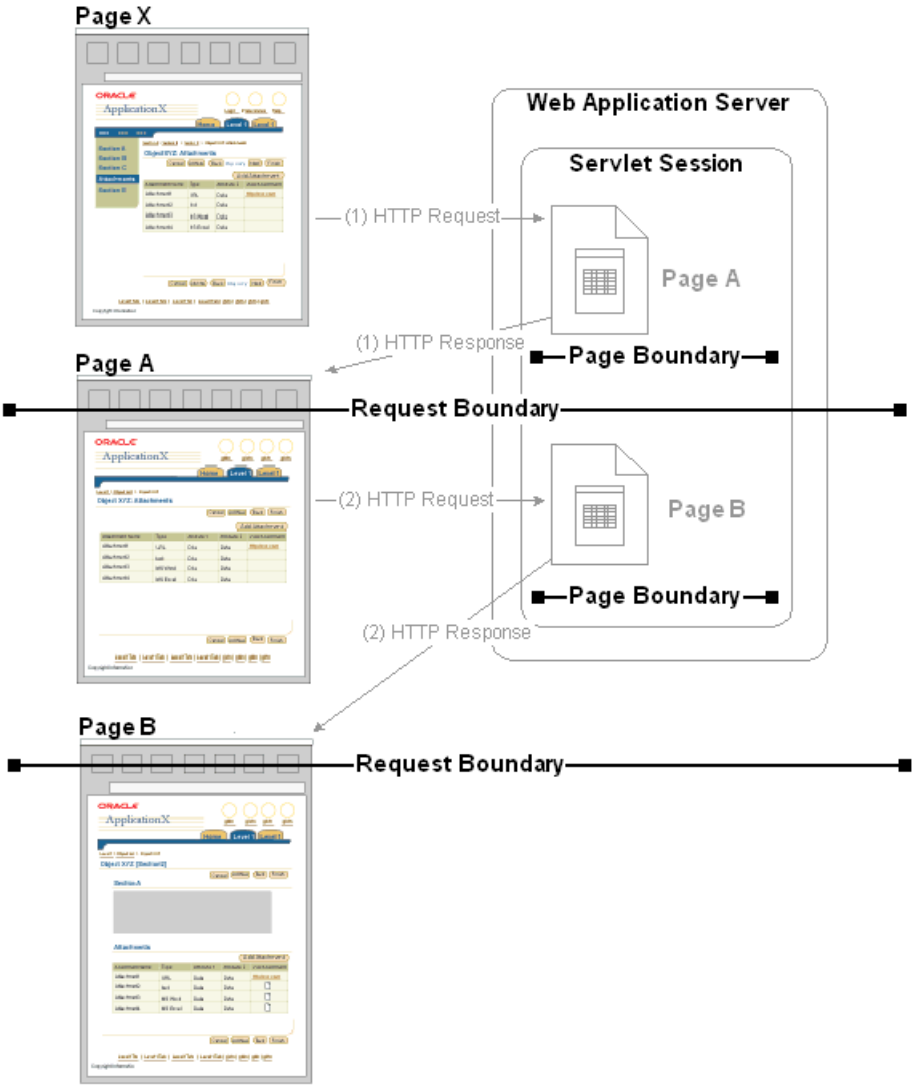
Request and Page Boundaries

As described in the JSP Application Primer document, a web application's unit of work is a request/response pair: the browser submits a request, the servlet processes the request and returns a response. The transmission of a response signifies the end of a single request, or the "boundary" between the completed request and a new one.

Similarly, when the `OAPageBean` finishes processing a page, this is the "boundary" between the current page and a new one.

So, in the following simple scenario where a user navigates from Page X to Page A and then to Page B, there are two request boundaries: the first is between Page X and Page A, the second is between Page A and Page B. There are also two page boundaries in the same conceptual location between Page X and Page A, and Page A and Page B.

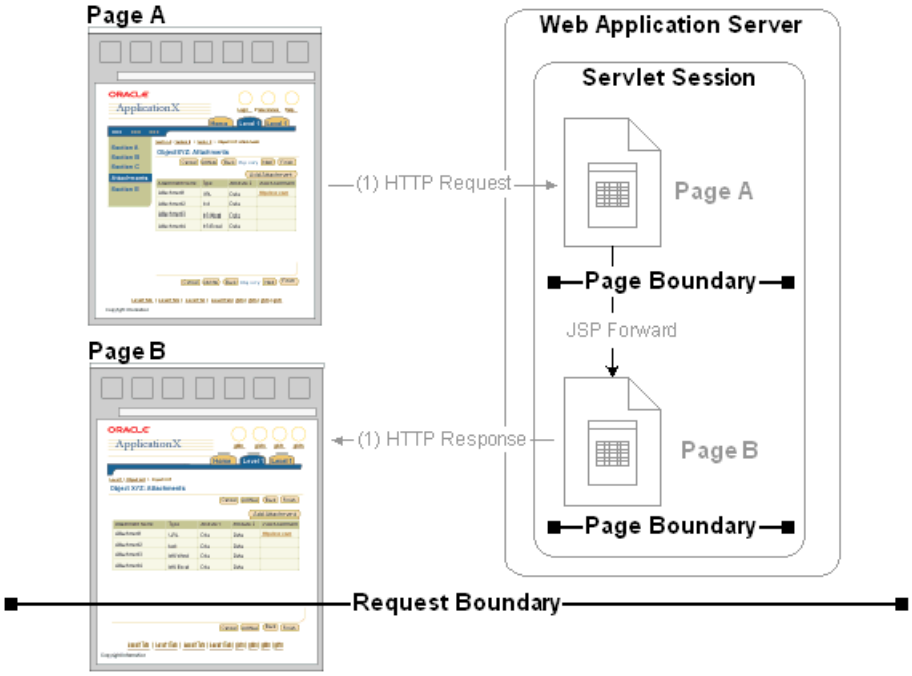
Figure 5: Conceptual illustration of request and page boundaries being the same



In some situations, however, the request and page boundaries are *not* the same. Consider the following JSP Forward case:

- The user navigates from Page X to Page A as illustrated in Figure 5 above.
- While on Page A, the user selects a control that the Page A code must evaluate before deciding which page to display in response. The browser issues a request to Page A, which OA Framework processes, including creating an `OAPageContext` for the page. Once Page A finishes processing, the first page boundary is reached as illustrated in Figure 6 below.
- Within the Page A code, the developer evaluates which control the user selected and issues a JSP Forward to Page B. Instead of providing an HTTP response at this point since we don't want to redisplay Page A, OA Framework begins processing for Page B, including creating a new `OAPageContext` for this page. Once Page B finishes processing, the second page boundary is reached.
- Since Page B must now be displayed to the user, an HTTP response it sent to the browser. The request boundary is now reached.

Figure 6: Conceptual illustration of request and page boundaries differing in the JSP Forward case



It is important to understand this distinction for several reasons:

- Request parameters exist throughout the life span of the request -- which can span multiple page boundaries. This can be somewhat surprising for new OA Framework developers who simply assume that a request and a page are the same thing, and therefore do not account for request parameters "hanging around" after performing a JSP Forward. Consider the following example:
    - A user selects a link in Page X that navigates to Page A. The Page A URL includes the parameter foo=bar.
    - Page A issues a JSP Forward to Page B. Now, even though we are in a new page, the request still includes the value foo=bar.
    - If you don't want a parameter value on the request after doing a JSP Forward, you *must* explicitly replace it. For example, in this case, simply reset the value to something like foo=X when you call the `OAPageContext's setForward*()` method.

      Note: You cannot actually *remove* parameters from a request.

      Tip: It is preferable to replace the unwanted parameter value with a new one that your code can use as an "ignore" value. Do not simply set the value to "".
- Since there isn't a one-to-one mapping between the page context and the request, some people find it a bit confusing that you access request parameters from the `OAPageContext`. Just remember that each page is a distinct entity, and from its "point of view," the `OAPageContext` represents the request.

Request

Although short-lived, an object is created for each HTTP request. This object contains the following application state:

- Any URL parameters, regardless of whether the browser issued a POST or a GET request
- Assuming the browser issues a POST request: any form field data values such as, the data a user enters into a text field or the data a developer stores in a hidden field.
- Assuming the browser issues a POST request: the web bean and event names associated with a user's selection of action/control components. For example, if the user selects a "Go" button to execute a query, the request includes the web bean name of this button so you can ascertain that it was pressed and respond accordingly.

To access any of these request values, use `OAPageContext` getParameter*() methods. You will not interact directly with the request itself.

To put values on the request (the preferred way of communicating between pages) you can do any of the following. See

Implementing the View and Implementing the Controller for additional information about working with request parameters.

Note: The following is a general description of the request-passing mechanisms; there are browser Back button considerations related to each of these techniques that you should fully understand before building your pages. See Supporting the Browser Back Button in Advanced OA Framework Development Topics.

Use Hidden Fields

A "hidden" field is a tool for developers to get/set values on a form that can't be accessed by the user. Just as the user's field values are added to the request during a form submit, so are the developer's field values -- assuming your page issues a POST.

You can create hidden fields declaratively in JDeveloper by selecting the **formValue** item style. At runtime, these are instantiated as an `oracle.apps.fnd.framework.webui.beans.form.OAFormValueBean`.

Specify Values During JSP Forward/Client Redirect

When you explicitly forward to a new page using the `OAPageContext` setForward*() methods or issue a client redirect by calling OAPageContext.sendRedirect(), you can optionally set request parameter values.

For example, Page A includes a submit button. When this button is selected, the user navigates to Page B using a JSP Forward. Page A needs to pass a "mode" value to Page B, which can be accessed several different ways so it knows how to behave.

1. The user selects the submit button.
2. In the Page A controller that handles this button press, we call OAPageContext.setForwardURL() in the `processFormRequest()` method. As part of this method call, we pass a request parameter named `queryMode` with a value of `automatic`.
3. In a Page B controller we check for the `queryMode` parameter value in the `processRequest()` method by calling getParameter("queryMode").
4. Page B's controller then responds to the fact that the `queryMode` value is `automatic` by immediately querying the data the page should display.

Specify Values by Calling OAPageContext.putParameter()

`OAPageContext` includes a putParameter() method that is used for passing values down the web bean hierarchy during page processing. Values specified with a call to `putParameter()` are not technically added to the request, but are stored in a special page cache.

Tip: For those familiar with the HttpServletRequest.setAttribute() method in the Java servlet 2.1 API, which is simply a way of stashing some information on the HTTP request, consider this its equivalent.

Set URL Parameters Declaratively

Specify request parameter values when defining URLs declaratively in JDeveloper or by setting the URL programmatically on web beans that have associated URLs.

Warning: The URL is space-constrained; be cautious about adding numerous URL parameters, particularly if they are lengthy. Since the URL is visible to users, encrypt any sensitive values as described later in Chapter 3.

Application Module Pooling

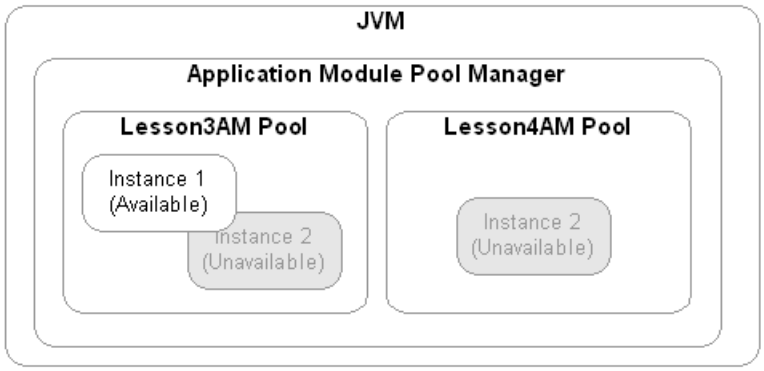
To improve performance and scalability, OA Framework pools (caches and reuses) application modules. Reuse is much more efficient than recreation. In simple terms:

- Each Java Virtual Machine (JVM) has an application module pool manager that contains and manages individual application module pools.
- Each application module pool contains multiple instances of the same application module. For example, if an application uses two root application modules (`oracle.apps.fnd.framework.toolbox.tutorial.server.Lesson3AM` and `oracle.apps.fnd.framework.toolbox.tutorial.server.Lesson4AM`), there would be two application module pools as shown in Figure 7 below. In other words, a pool is created for each root application module type in your product.
- Application module instances within the pool are designated as being available for use, or unavailable (currently "checked out").
- Only root application modules are pooled; nested application modules are pooled as children of the root application module.

To learn more about how this works for both application modules and connections, including how to configure, monitor and optimize pool performance in an OA Framework application deployment, see the Chapter 6 advanced topic Application Module and

Connection Pooling. In particular, focus on the sections titled Architectural Overview and Application Module Pooling in this topic.

Figure 7: Conceptual illustration of application module pool



Chaper 3: Basics of Building an OA Framework Application

Implementing the Model

This document describes how to implement your model objects in generic terms.

Contents

- Designing Model Objects
- Recommended Build Approach
- Business Components Packages
- Entity Objects
- Entity Associations (Association Objects)
- View Objects and View Rows
- View Links
- Application Modules
- Entity Objects, Entity Experts, 'Validation' Application Modules and 'Validation' View Objects
    - Validation View Objects (VVOs)
    - Validation Application Modules (VAMs)
    - Entity Experts
- Reusing Business Objects

Prerequisite Reading

This document assumes that you have read the following in the OA Framework Developer Guide:

- Building 'Hello, World!'
- JSP Application Primer
- Anatomy of an OA Framework Page
- OA Framework State Management

Designing Model Objects

'Client' / 'Server' Code Separation

Within the Model-View-Controller architecture, OA Framework draws a clear distinction between "client" and "server" classes, a distinction that on the surface may seem to conflict with JSP application architecture as described in the JSP Application Primer. In Chapter 2, we say that a *typical* JSP application has 3 physical tiers:

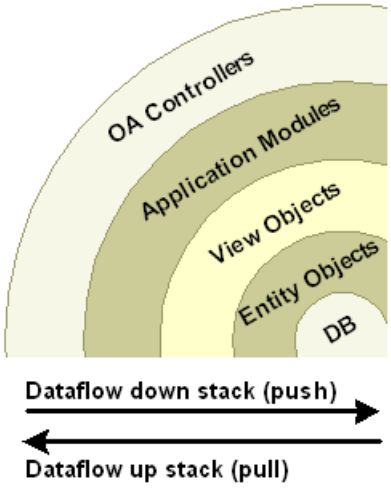
- The browser (the client where our HTML renders and users interact with the UI)
- The web application server (the middle tier where our UI web bean hierarchy is constructed and our application business logic executes)
- The database server

Within the middle tier, OA Framework draws a further distinction between "client" and "server" classes as follows:

- Client classes (View and Controller code) drive the HTML user interface
- Server classes (Model code) supports any client (not just OA Framework) user interfaces

This distinction is important because it preserves the ability to use server code with different clients. Consider the following image which illustrates the relationships between a page's UI controllers, application modules, UI-specific view objects, and the underlying entity objects:

Figure 1: OA Framework "onion" showing code layers and boundaries



In general, to enable reuse at the layer boundaries, objects reference down the dataflow stack, but not up.

- Model code should never reference controller code directly. For example, view objects and application modules should *not* call methods in your UI controllers, and entity objects should *not* reference UI application modules and view objects, as discussed later. However, entity objects can and do make use of server-side validation application modules and view objects.
- Never reference/import any server-side implementation classes or interfaces on the client-side (classes/interfaces that are in the `oracle.apps.fnd.framework.server` package). For example, do *not* call methods on an `oracle.apps.fnd.framework.OADBTransaction` in your UI controller.
- If you need the server code to do some work for you, always route your calls through the root application module using the generic "remoteable" `invokeMethod()` method on the `oracle.apps.fnd.framework.OAApplicationModule` interface, or create an interface for your application modules so you can call typed methods, whose invocation can be checked at compile time. The application module can delegate or implement the logic as required.

  Note: The `OAApplicationModule` interface is in the `oracle.apps.fnd.framework` package and therefore, does not violate the rule stated in the first sentence of this bullet-point. All classes, interfaces, and exceptions in the `oracle.apps.fnd.framework` package can safely be used in both client and server code.

  If you opt to create an interface for your application module instead of using `invokeMethod()`, create this interface in the package directory immediately above your implementation. For example, the `EmployeeAM` interface for the `oracle.apps.fnd.framework.toolbox.labsolutions.server.EmployeeAMImpl` application module should be created in the `oracle.apps.fnd.framework.toolbox.labsolutions` package.
- Never include JDBC or other server-side processing directly in your client code. Again, if the UI client needs information from this server it should ask the application module, which can then delegate or implement the request appropriately.

Coding Standards Compliance

Before defining any model objects or writing supporting code, read the following documents carefully. While this topic mentions several key model standards, it is not intended to be a comprehensive checklist. For any OA Framework code that you write, the documents in Chapter 8 should be considered the *single source of truth* for coding standards.

- Chapter 8: Oracle E-Business Suite Java Coding Standards
- Chapter 8: OA Framework Naming / File / Package / Directory Structure Standards
- Chapter 8: OA Framework Model Coding Standards

Recommended Build Approach

If you are preparing to build your first OA Framework application , you might find it easiest to proceed as follows for a small module (a single page, or a few simple, related pages that participate in the same transaction).

Note: This assumes you have completed your design work, and are ready for implementation. It also assumes that you are building a complete module, not just the UI, or just the business logic.

1. Create any business components packages that you need for your BC4J model objects.
2. Implement declarative BC4J application module, entity object, entity association, view object and view link definitions needed for your page(s). Add view objects to your root application module(s) as appropriate. Don't worry about writing code at this stage.

3. Create the menu definition for your application (discussed in Implementing the View).
4. Create the OA user interface components for your page(s) (discussed in Implementing the View).
5. Create and implement controller code (discussed in Implementing the Controller).
6. Implement UI application module code supporting your pages.
7. Implement entity object business logic.

Business Components Packages

All BC4J model components must belong to a Business Components (BC4J) package.

Note: This section assumes that you have set up your development environment, created an OA Workspace, and defined a working datasource as described in Building and Running "Hello, World!".

1. In the JDeveloper Navigator, select the OA Project where you want to create your package.
2. From the main menu, choose **File > New** to open the **New Gallery**.
3. In the **Categories** tree, expand the **Business Tier** node, and select **Business Components (BC4J)**.
4. In the **Items** list, select **Business Components Package** to open the **Business Components Package Wizard**. You can also right-click on the OA Project and select **New Business Components Package** to navigate directly to the **Business Components Package Wizard**.
5. In Step 1 of 3, enter a **Package Name** that complies with the OA Framework File / Directory / Package Structure standards. Also select **Entity Objects mapped to database schema objects** in the business entity modeling section. Select the **Next** button.
6. In Step 2 of 3, verify your **Connection Name** (it should point to the database where you want to work; JDeveloper uses this information in all of the BC4J component wizards). Set the **SQL Type** and the **Type Map** to "Oracle."
7. Select the **Finish** button to save your changes.

To change the database connection associated with an existing BC4J package, which you need to do if you change your development environment:

1. Select the OA Project with the business components package you want to edit.
2. Right-click and select **Edit Business Components Project**.
3. In the Business Components Project Wizard, select **Connection**.
4. Specify your new database.
5. Select **OK** to save your changes.

Entity Objects

This introduces the basics of entity object creation/usage. See Java Entity Objects and PL/SQL Entity Objects in Chapter 5 for additional information about working with these objects.

As described in Anatomy of an OA Framework Page, an entity object implements the business rules for a given table, view, or snapshot. Entity objects are intended to be used in many clients (not just an OA Framework UI), and as such, should consolidate all of the relevant validation/behavior for the table.

- Each table should have at most one entity object.
- Entity objects should include attributes for all columns in its table.

  Note: For Release 12.2, make sure the entity object omits the `ZdEditionName` attribute, as it is only for internal use by Online Patching.

- You can subclass your own common entity objects (for example, see the Advanced Model Development Topics in Chapter 6 to learn how to create polymorphic entity objects that extend a common base class).
- You will commonly add object initialization, attribute validation, entity-level validation, and other functional behaviors to your entity objects.
- You can also create "entity expert" singletons to perform work shared by multiple related entity objects in a composite business object, such as a purchase order which has a header, lines and shipments. Other referencing entity objects can also use the entity expert to perform lightweight validations (for example, a purchase order might ask a supplier's entity expert if a supplier id that it wants to reference is valid). Entity experts are discussed later.
- Finally, you can create server "helper" objects and interfaces as needed to create modular code. For example, as illustrated in the OA Framework ToolBox Tutorial, you can create one or more helper objects to perform processing on multiple entity object types.

Declarative Implementation

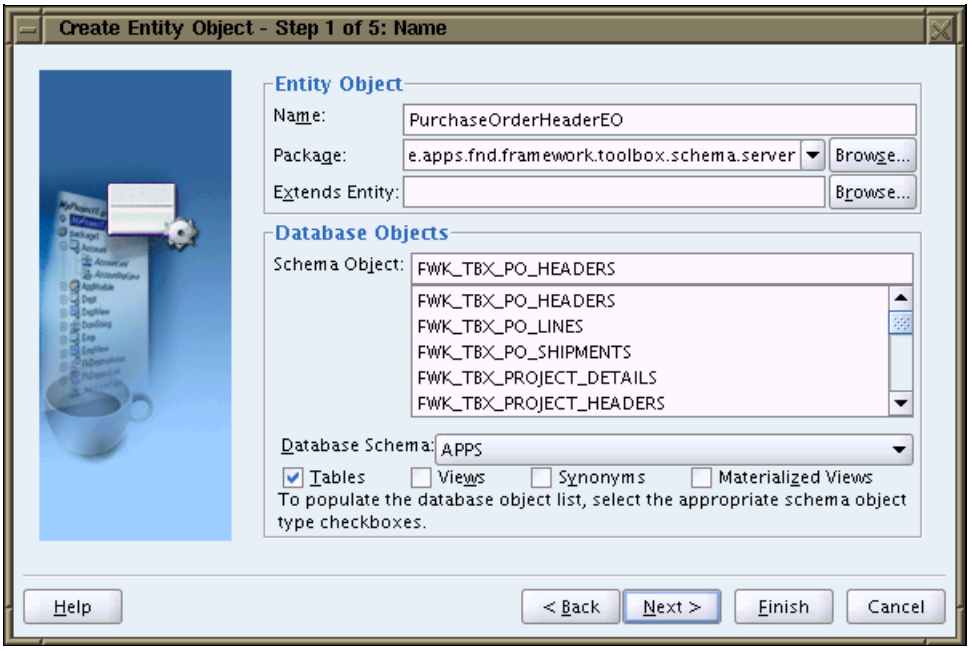
For additional information about Entity Object Wizard properties not specifically described here, see the JDeveloper documentation.

Note: You can access context-sensitive Help while in any of the BC4J wizards by selecting the **F1** key on your keyboard.

To create a new entity object in a Business Components (BC4J) package:

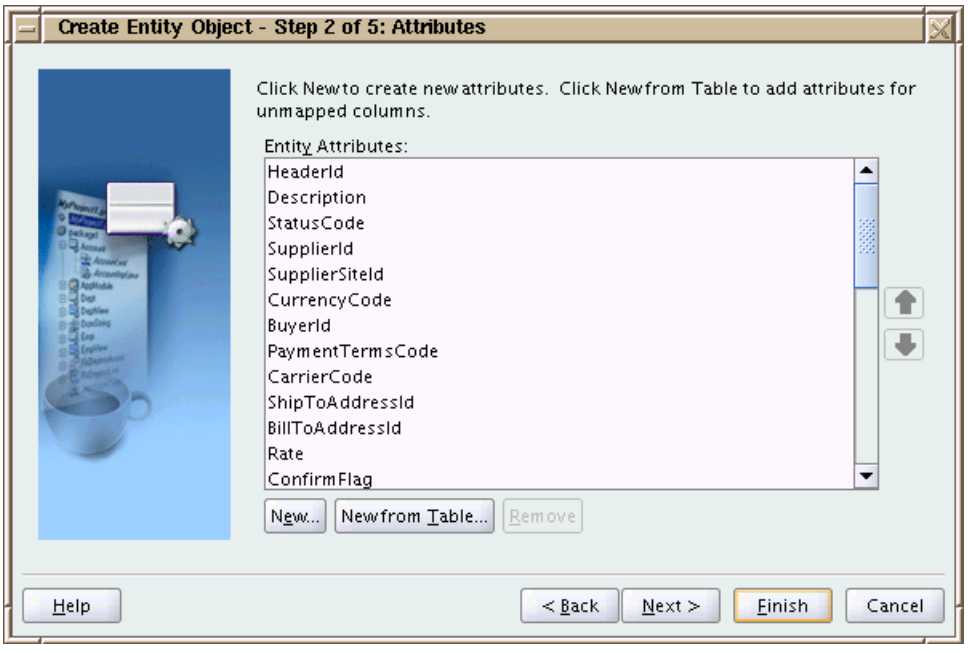
1. In the JDeveloper Navigator, select the BC4J package where you want to create your entity object.
2. From the main menu, choose **File > New** to open the **New Gallery**.
3. In the **Categories** tree, expand the **Business Tier** node, and select **Business Components (BC4J)**.
4. In the **Items** list, select **Entity Object** to open the **Entity Object Wizard**. You can also right-click on the BC4J package and select **New Entity Object** to navigate directly to the **Entity Object Wizard**.
5. In the **Name** page (Step 1 of 5):

Figure 2: Entity Object Wizard Name Page (Step 1 of 5)



   - Specify the entity object's **Name** in accordance with the OA Framework File / Directory / Package Structure standards.
   - Verify that you have selected the right BC4J **Package**.
   - Do *not* enter a value in the **Extends Entity Object** field unless you are deliberately subclassing one of your own entity objects.
   - Specify a **Schema Object** (the exact name of the table for the entity object) as shown in Figure 2, to improve the wizard's performance. You do *not* need to check the **Synonyms** or **Tables** checkboxes.
   - Select **Next** to proceed.
6. In the **Attributes** page (Step 2 of 5), you should see all the columns in the table that you specified in the **Name** page.

Figure 3: Entity Object Wizard Attributes Page (Step 2 of 5)



   - Do *not* remove any entity attributes so as to include all of the table's columns.

Note: For Release 12.2, remove the `ZdEditionName` attribute, as it is only for internal use by Online Patching.

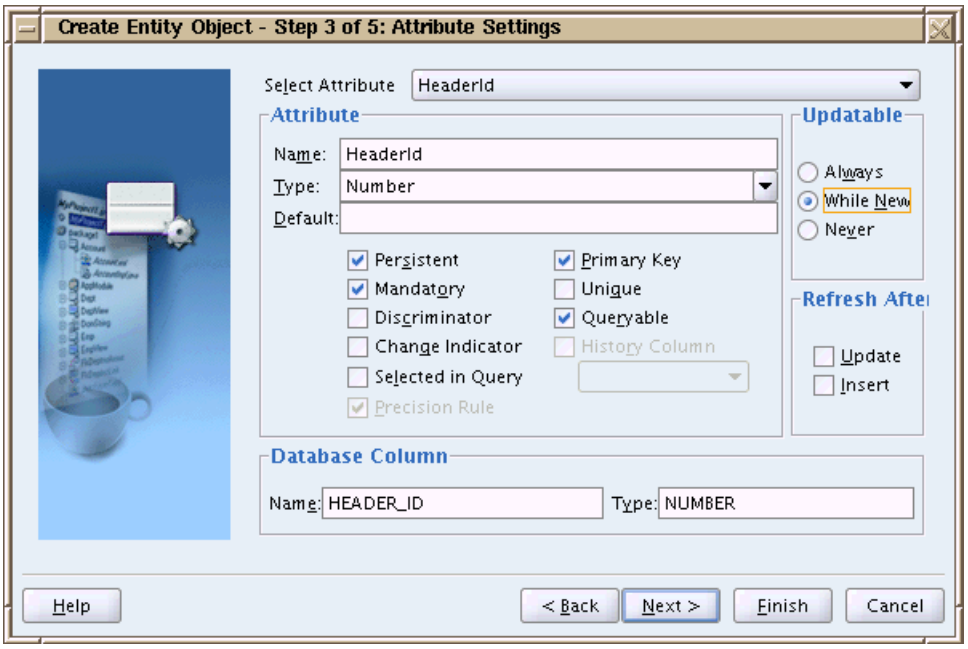
- Select **New...** to create a transient attribute that is used in the business logic, such as a calculated OrderTotal in a purchase order that is used for approval checking.

  Note: To display a calculated value for an entity object that isn't really relevant to the business logic itself (it is very UI-specific), you can always create an attribute for it in the view object as described below.

- Select **Next** to proceed.

7. In the **Attribute Settings** page (Step 3 of 5), verify or set the following information for each of the entity object's attributes:

Figure 4: Entity Object Wizard Attribute Settings Page (Step 3 of 5)



- The **Attribute** and **Database Column Name** and **Type** properties default correctly from the table definition.
- All database column attributes have the **Persistent** and **Queriable** checkboxes selected as shown.
- For primary key columns, ensure that the **Primary Key** and **Mandatory** checkboxes are selected.

  Warning: If you fail to define your primary keys, BC4J will generate a ROWID key for you. *You should explicitly define your primary key instead of relying on ROWID as your primary key because this can lead to problems*. For example, in a Real Application Cluster with database partitions, the same database row can have a different ROWID in different partitions. Similarly, if a table is partitioned, the ROWID can change inside a transaction if the row is migrated.

- For columns that are never updateable, or updateable only when new, select the appropriate **Updateable** radio button.
- For columns whose values change after database triggers execute, select the **Refresh After** update or insert as appropriate.
- Never select the **Unique** checkbox; uniqueness checking should always be performed programmatically (the state of this checkbox has no impact on runtime behavior; BC4J uses it to create a database constraint if you generate a table from the EO wizard).

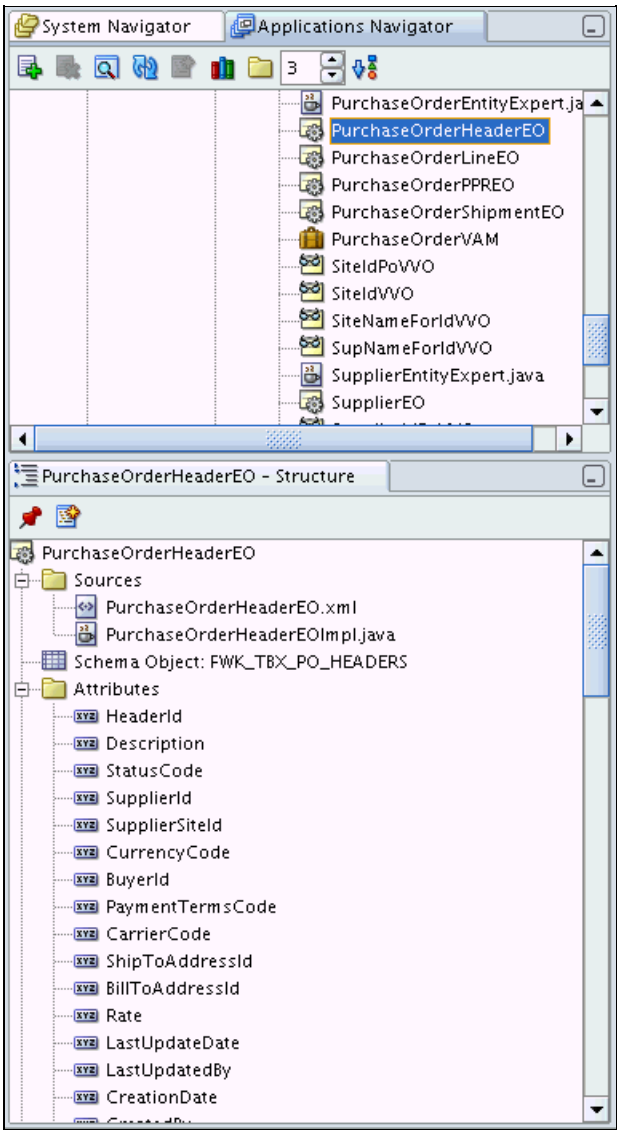
  Note: The **Unique** property has no effect if the EO is created on an already existing database table.

- The Discriminator column is used for polymorphic entity objects as described in Chapter 6 Advanced Model Development Topics.
- If you are using an Object Version Number column, select the **Change Indicator** checkbox for it. See Chapter 5 for information about this.
- Select **Next** to proceed.

8. In the **Java** page (Step 4 of 5) page:
   - Check the option for generating an **Entity Object Class**. In the **Generate Methods** box, opt to generate **Accessors,** a **Create Method** and a **Delete Method**.
   - Select the **Extends...** button to verify that you will be subclassing oracle.apps.fnd.framework.server.OAEntityImpl for the **Row** (the entity object).
9. Select **Finish** to save your entity object definition and implementation. BC4J will create an XML definition file and a Java implementation file for your entity object as shown in Figure 5. Note that you can quickly view the underlying table and attributes included in the entity object by simply selecting it in the Applications Navigator.
10. Select your entity object, right-click and select **Edit<*EntityName*>**... Navigate to the **Tuning** page and check the **Use Update Batching** property. Set the **Threshold** value to **100**. See the OA Framework Model Coding Standard M68 for additional

information about batch DML updates, including a list of situations in which this feature cannot be used.

Figure 5: JDeveloper Applications Navigator and Structure pane showing a selected entity object



Multilanguage "_TL" Entity Objects

To create a multilanguage "_TL" entity object, follow the special instructions published in Java Entity Objects for Translatable Tables.

PL/SQL Entity Objects

To create a PL/SQL entity object, follow the special instructions published in PL/SQL Entity Objects.

Programmatic Control

For detailed information about coding entity object business logic, see Java Entity Objects and PL/SQL Entity Objects.

Entity Associations (Association Objects)

Associations let you create declarative relationships between entity objects. At runtime, BC4J uses these relationships to coordinate the related objects. There are two basic types of associations:

- **Composition** - A strong association where the source entity object *owns* the destination entity object. In other words, the destination cannot exist independent of its source. For example, a purchase order header is comprised of purchase order lines, which have no meaning or life span outside the context of their header.

- **Reference** - A weak association where the source entity object only references the destination entity object. For example, a purchase order header references a supplier, but the supplier can still exist regardless of whether a purchase order references it or not.

Create composition associations as appropriate for all your entity objects and ensure that they are properly created, initialized and managed at runtime. BC4J automatically treats compositions as a logical unit, so for example, a purchase order header is automatically locked even if you make changes only to its lines).

Create reference associations for any entity objects that you're likely to update or instantiate at runtime. For example, create associations between a purchase order header and its supplier if you can update a supplier while editing a purchase order, but don't create associations between purchase orders and a freight terms lookup code entity object.

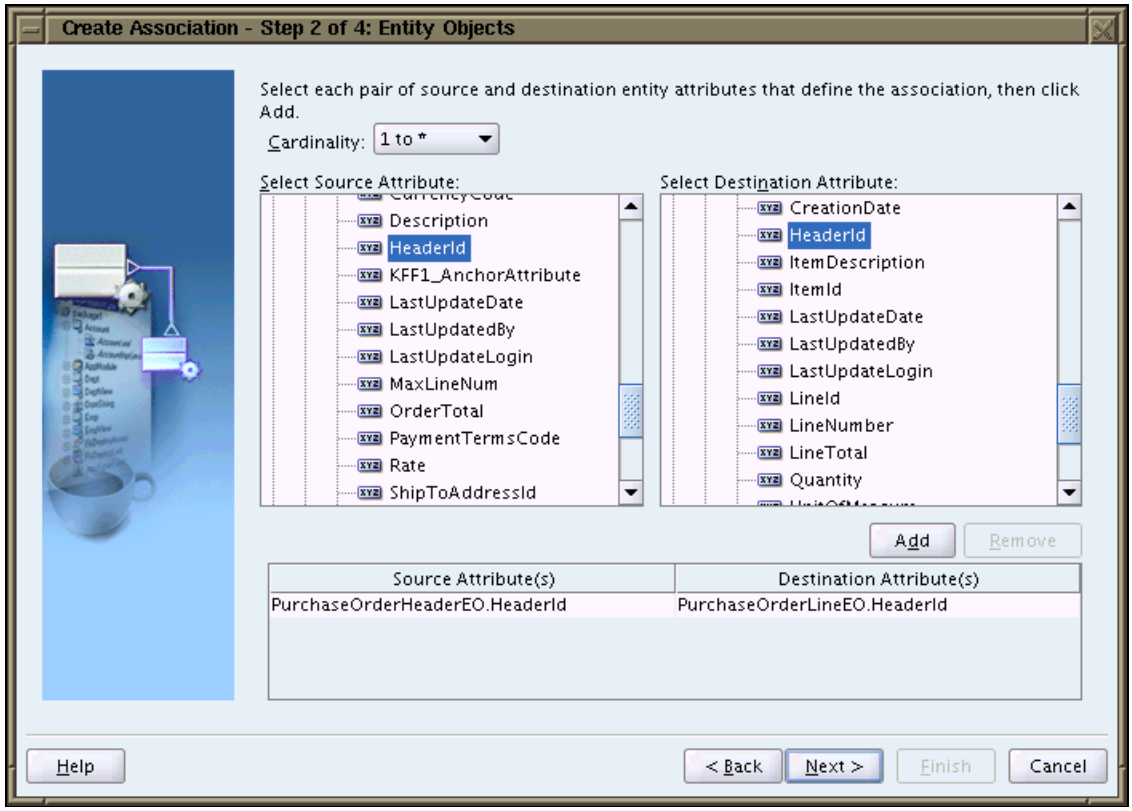
Declarative Implementation

For additional information about Association Wizard properties not specifically described here, see the JDeveloper documentation.

Note: You can access context-specific Help while in any of the BC4J wizards by selecting the **F1** key on your keyboard.

1. In the JDeveloper Navigator, select the BC4J package where you want to create your association object.
2. From the main menu, choose **File > New** to open the **New Gallery**.
3. In the **Categories** tree, expand the **Business Tier** node, and select **Business Components (BC4J)**.
4. In the **Items** list, select **Association** to open the **Association Wizard**. You can also right-click on the BC4J package and select **New Association Object** to navigate directly to the **Association Wizard**.
5. In the **Name** page (Step 1 of 4):
    - Specify the association's **Name** in accordance with the OA Framework File / Directory / Package Structure standards.
    - Verify that you have selected the right BC4J **Package**.
    - Do NOT enter a value in the **Extends Association** field unless you are deliberately subclassing one of your own associations.
    - Select **Next** to proceed.
6. In the **Entity Objects** page (Step 2 of 4), specify the association's cardinality (for example, is it a one-to-many relationship?) and select the source and destination join attributes as shown in Figure 6. Select the **Add** button to create the join (repeat as necessary for a multi-key relationship). Select **Next** twice to proceed.

Figure 6: Selecting source and destination entity objects and attributes in the Entity Object (Step 2 of 4) page



7. In the **Association Properties** page (Step 4 of 4):
    - Check the **Expose Accessor** options as appropriate for the source and destination objects (an accessor lets the object get a reference to the other end of the association).
    - Select the **Composition Association** checkbox if the destination object cannot exist outside the context of the source

object.

Note: For compositions, always check the **Expose Accessor** option on the destination object. Optionally, enable this for the source object as required in your code.

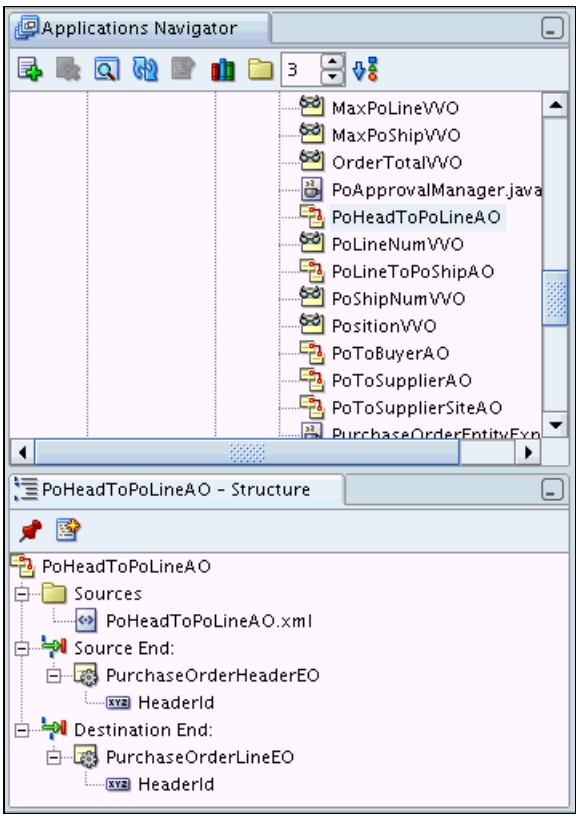
Note: Use the **Lock Top-Level Container** option to control whether adding, removing, or modifying a composed detail entity row should lock the composing entity before changes are saved.

- Do not select any of the other page options.

8. Select **Finish** to save your association. BC4J creates an XML definition file as shown in Figure 7.

Note: You can quickly view the underlying relationship by simply selecting the association in the Applications Navigator.

Figure 7: JDeveloper Applications Navigator and Structure pane showing a selected association object



Programmatic Control

Association objects have no implementation, so you will not write any code for them. In Chapter 5, we discuss how to access an entity object using an association.

View Objects and View Rows

This introduces the basics of view object and view row creation/usage. See View Objects in Detail in Chapter 5 for additional information about working with these objects.

Design Considerations

As described in Anatomy of an OA Framework Page, view objects encapsulate database queries and provide access to associated entity objects. One of the most important view object design decisions is whether it should be based on plain SQL, or on entity objects.

- All trivial UI view objects for things like Lists of Values (LOV) and poplists are based on plain SQL.
- Any *validation* view objects, created to implement simple business rules for an entity object, are based on plain SQL. See the Entity Objects, Entity Experts, "Validation" Application Modules and "Validation" View Objects topic below for additional information.
- All other view objects created for the UI, regardless of whether they are updateable, are based on entity objects.

For performance reasons, view objects need to be optimized for a given use case. Creating several small, task-specific view objects

is preferable to sharing a single, large view object in a range of UIs. View objects should be considered UI-specific.

Avoid using dynamic `WHERE` clauses wherever possible (view objects support the ability to modify their declarative definitions programmatically). If possible, statically define 3 different view objects for the same `SELECT` -- each with a declarative `WHERE` clause to which you can simply bind at runtime. However, It is appropriate to modify the `WHERE` clause in view objects used with complex query criteria because it is impractical to create individual definitions for every possible query criteria combination.

View objects, like any BC4J objects, can be created declaratively and programmatically. For performance reasons it is preferable, if you can, to declaratively define the view object.

Declarative Implementation

For additional information about View Object Wizard properties not specifically described here, see the JDeveloper documentation.

Note: You can access context-specific Help while in any of the BC4J wizards by selecting the F1 key on your keyboard.

Important: Whenever you create a view object, *always* generate a `view row implementation` class. You should generate a `view object implementation` class only if you intend to write any code for the view object.

You can create a shared view object, which subclasses `oracle.apps.fnd.framework.server.OAViewObjectImpl`, that you can then subclass to create more specific behaviors.

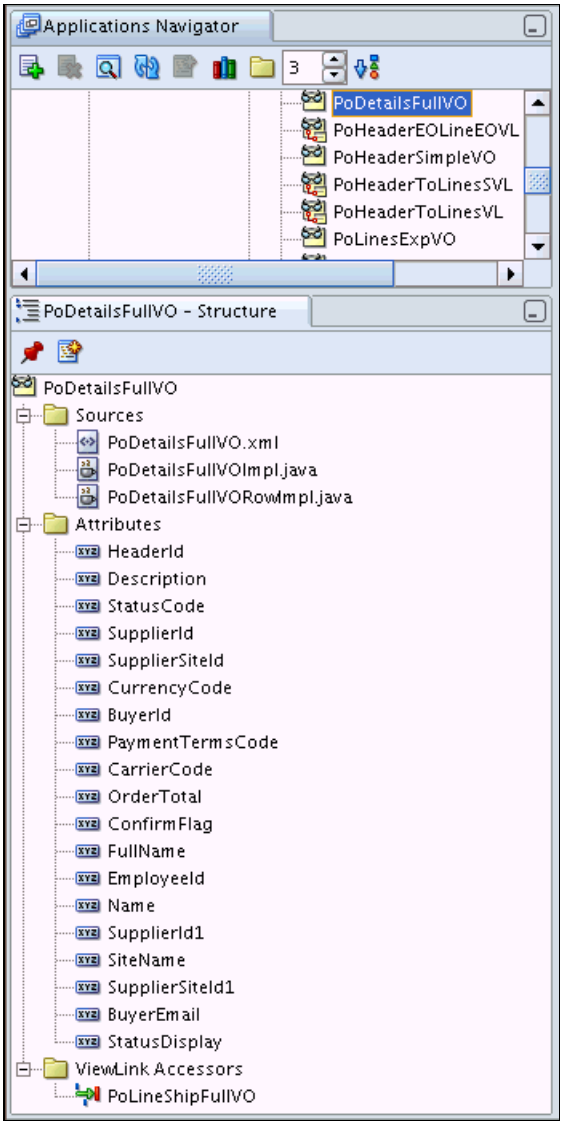
SQL View Objects

To create a new view object in a Business Components (BC4J) package that is based entirely on a SQL query:

1. In the JDeveloper Navigator, select the BC4J package where you want to create your view object.
2. From the main menu, choose **File > New** to open the **New Gallery**.
3. In the **Categories** tree, expand the **Business Tier** node, and select **Business Components (BC4J)**.
4. In the **Items** list, select **View Object** to open the **View Object Wizard**. Note that you can also right-click on the BC4J package and select **New View Object** to navigate directly to the **View Object Wizard**.
5. In the **Name** page (Step 1 of 8):
   - Specify the view object's **Name** in accordance with the OA Framework File / Directory / Package Structure standards.
   - Verify that you have selected the right BC4J package.
   - Do NOT enter a value in the **Extends View Object** field unless you are deliberately subclassing one of your own view objects.
   - Select **Next** until you get to Step 5.
6. In the **SQL Statement** page (Step 5 of 8):
   - Enter your query in the **Query Statement** field (do not include a semicolon). Note that you must always use Oracle-style bindings (`select emp_name from emp where emp_id = :1`) if you expect to bind variables at runtime.
   - Select the **Test...** button to verify that your query is correct.
   - Select **Next** to proceed.
7. In the **Attribute Mappings** page (Step 6 of 8):
   - Verify that **Query Columns** you defined in your `SELECT` match the **View Attributes**. If they don't match, click the View Attribute value that is in error to activate a poplist. Select the correct attribute.
   - Select **Next** until you get to Step 8.
8. In the **Java** page (Step 8 of 8):
   - Always check the option to generate a **View Row Class** (including accessors).
   - Check the option to generate a **View Object Class** only if you anticipate writing any code for your view object (you can always generate this class later if you need to, or delete it if you generate it now and find later that you don't have a need for it).
   - Select the **Class Extends...** button to ensure that you are subclassing the OA Framework classes oracle.apps.fnd.framework.server.OAViewObjectImpl and oracle.apps.fnd.framework.server.OAViewRowImpl as appropriate. If you need to correct the default values, select **Browse...** to open the **Find Superclass** window.
9. Select **Finish** to save your new view object. BC4J will create an XML definition file and Java implementations as shown in Figure 8.

   Note: You can quickly view the underlying attributes and view links by selecting the view object in the Applications Navigator.

Figure 8: JDeveloper Applications Navigator and Structure pane showing a selected view object



At this point, you are not quite finished with the creation process. To proceed, you need to edit the view object as follows:

1. In the JDeveloper Navigator, select the view object that you just created, right-click and select **Edit <*viewobject_name*>...**.
2. In the **View Object Wizard**, select **Tuning**.
3. In the **Tuning** page, deselect the **Enable Passivation** checkbox. Select **OK** to save your changes.

Entity Object View Objects

To create a new view object in a Business Components (BC4J) package that is based entirely on entity objects:

1. In the JDeveloper Navigator, select the BC4J package where you want to create your view object.
2. From the main menu, choose **File > New** to open the **New Gallery**.
3. In the **Categories** tree, expand the **Business Tier** node, and select **Business Components (BC4J)**.
4. In the **Items** list, select **View Object** to open the **View Object Wizard**. Note that you can also right-click on the BC4J package and select **New View Object** to navigate directly to the **View Object Wizard**.
5. In the **Name** page (Step 1 of 8):
    - Specify the view object's **Name** in accordance with the OA Framework File / Directory / Package Structure standards.
    - Verify that you have selected the right BC4J package.
    - Do NOT enter a value in the **Extends View Object** field unless you are deliberately subclassing one of your own view objects.
    - Select **Next** to proceed.
6. In the **Entity Objects** page (Step 2 of 8):
    - In the **Available** list, select the entity objects that you want to include in the view object and shuttle them to the **Selected** list.
    - Indicate whether the entity objects are **Read Only**, and if they should be treated as a **Reference** (see the JDeveloper documentation for additional information about this page).

    - Select **Next** to proceed.
7. In the **Attributes** page (Step 3 of 8) select the attributes that you want to include from the **Available** list and shuttle them to the **Selected** list. Select **Next** to proceed.
8. In the **Attribute Settings** page (Step 4 of 8), verify that the default information is correct. Select **Next** to proceed.
9. In the **SQL Statement** page (Step 5 of 8):
    - Verify that the query BC4J generated for you is correct. If not, select the **Expert Mode** checkbox to make the query editable.

      Note: For expert mode view objects, do NOT try to set a value in your `SELECT` statement for an EO attribute. For example, do not assume the *flag* column is based on an EO attribute as this results in a locking failure because BC4J tries to compare this value with the original database value and complains that they are different. See Java Entity Objects for valid approaches to setting attribute default values.

      ```
      SELECT x pk1, y pk2, z status, 'Y' flag, ....
      ```

    - Select the **Test...** button to verify that your query is correct.
    - Select **Next** until you get to Step 8.
10. In the **Java** page (Step 8 of 8):
    - Check the option to generate a **View Row Class** (including accessors).
    - Check the option to generate a **View Object Class** only if you anticipate writing any code for your view object (you can always generate this class later if you need to, or delete it if you generate it now and find later that you don't have a need for it).
    - Select the **Extends...** button to ensure that you are subclassing the OA Framework classes oracle.apps.fnd.framework.server.OAViewObjectImpl and oracle.apps.fnd.framework.server.OAViewRowImpl as appropriate. If you need to correct the default values, select **Browse...** to open the **Find Superclass** window.
11. Select **Finish** to save your new view object.

Hybrid View Objects (Expert-Mode View Objects)

You can also create view objects that are based on entity objects, and include SQL attributes. In this case, create the view object as described in the entity object case above, with a few small modifications:

- In the **Attributes** page, select the **New** button to create attributes for the non-entity object values that you want to query directly.
- In the **Query** page, select **Expert Mode** to edit the generated SQL as needed to retrieve these "calculated" values.
- In the **Attribute Mappings** page (displayed only if you have SQL-based attributes), ensure that the **Query Columns** and **View Attributes** match.

Primary Keys

Per the OA Framework Model Coding Standard M39, almost all view objects require primary keys. You can specify primary keys declaratively when defining attributes, or you can set them programmatically by calling `setKeyAttributeDefs()` on `OAViewObjectImpl`.

Programmatic Control

Query Handling

Each view object implements its own search, and if necessary, should be capable of translating incoming parameters to bind variables and `WHERE` clause phrases. As a general coding practice, all methods that perform this work should be named `initQuery()` or some descriptive variant like `initNewEmployeesQuery()` if you need multiple "init" methods.

Note: You must also use "Oracle-style" binding (`FOO >= :1`) instead of ANSI-style binding (`FOO >= ?`).

The following example illustrates how to modify a `WHERE` clause and bind search criteria based on the values passed to the `initQuery` method.

```
// Initialize and execute the query
public void initQuery(String name, String onHold, String number)
{

   StringBuffer whereClause = new StringBuffer(100);
   Vector parameters = new Vector(3);
   int clauseCount = 0;
   int bindCount = 0;

   setWhereClauseParams(null); // Always reset

   if ((name != null) && (!("".equals(name.trim()))))
   {
     whereClause.append(" NAME like :");
```

```
      whereClause.append(++bindCount);
      parameters.addElement(name + "%");
      clauseCount++;
    }

    if ((number != null) && (!(""Equals(number.trim()))))
    {

      Number supplierId = null;

      // SUPPLIER_ID is a NUMBER; datatypes should always
      // match, and the parameter passed to this method is a
      // String.
      try
      {
        supplierId = new Number(number);
      }
      catch(Exception e) {}

      if (clauseCount > 0)
      {
        whereClause.append(" AND ");
      }

      whereClause.append(" SUPPLIER_ID = :");
      whereClause.append(++bindCount);
      parameters.addElement(supplierId);
      clauseCount++;
    }

    if ((onHold != null) && (!(""Equals(onHold.trim()))))
    {
      if (clauseCount > 0)
      {
        whereClause.append(" AND ");
      }

      whereClause.append(" ON_HOLD_FLAG = :");
      whereClause.append(++bindCount);
      parameters.addElement("Y");
      clauseCount++;
    }

    setWhereClause(whereClause.toString());

    if (bindCount > 0)
    {
      Object[] params = new Object[bindCount];

      // the copyInto() is 1.1.8 compliant which, as of 4/02/03, is required by ARU

      parameters.copyInto(params);
      setWhereClauseParams(params);
   }

   executeQuery();

 } // end initQuery( )
```

Business Logic

View objects are not an appropriate home for business logic; you should not be writing validation rules in your view objects or view rows.

View Rows

Although you should always create a view row as mentioned above, for the most part, you won't need to write view row code. View row code is useful in cases where you want to calculate a transient attribute value, for example, but you can't or don't want to include the logic in your query (perhaps the performance cost is too high). You can also use view row code to perform simple validations of transient attributes used in the UI, or call custom entity object methods (see the "Approve" example in the Application Module section below for additional information).

Custom view row methods may not be accessed directly on the client. The client must first invoke a method on the application

module, which delegates to the view object. The view object provides access to the view row. Furthermore, to realize the performance benefit of having the view row class, always call the generated setters/getters (for example, `setSupplier()`) on the row if you need to programmatically access or set values. This is because it is much faster than calling the generic `setAttribute("<AttributeName>")` and `getAttribute("<AttributeName>")`. For example, the Entity Object Delete Example in the Application Module section below shows how to properly retrieve a view row attribute value.

View Links

As described above, an association defines a relationship between two entity objects. Similarly, a view link defines a relationship between two view objects that BC4J uses to automatically query and coordinate the destination view object based on the current source view object.

View links can be based on an association or a declarative join relationship between two view objects. For example, suppose two tables have a master-detail relationship based on a foreign key. The corresponding entity objects are related via an association, and view objects based on those entity objects can be related by a view link based on the association.

Although view links can be very convenient, use them sparingly in the web applications pages because they cache both the master and detail records as the user navigates from one master row to the next -- and this can be expensive. Use view links only in the following cases:

- When specific beans (like the HGrid) require them.
- When you have updateable master/detail view objects (on the same or different pages) whose underlying entity objects are related using composition, *you must define a view link between them* (we discuss this further in Chapter 5).
- When you have a read-only master and detail view object on the same page, and navigating to a master row should cause the children to query automatically.

Declarative Implementation

For additional information about View Link Wizard properties not specifically described here, see the JDeveloper documentation.

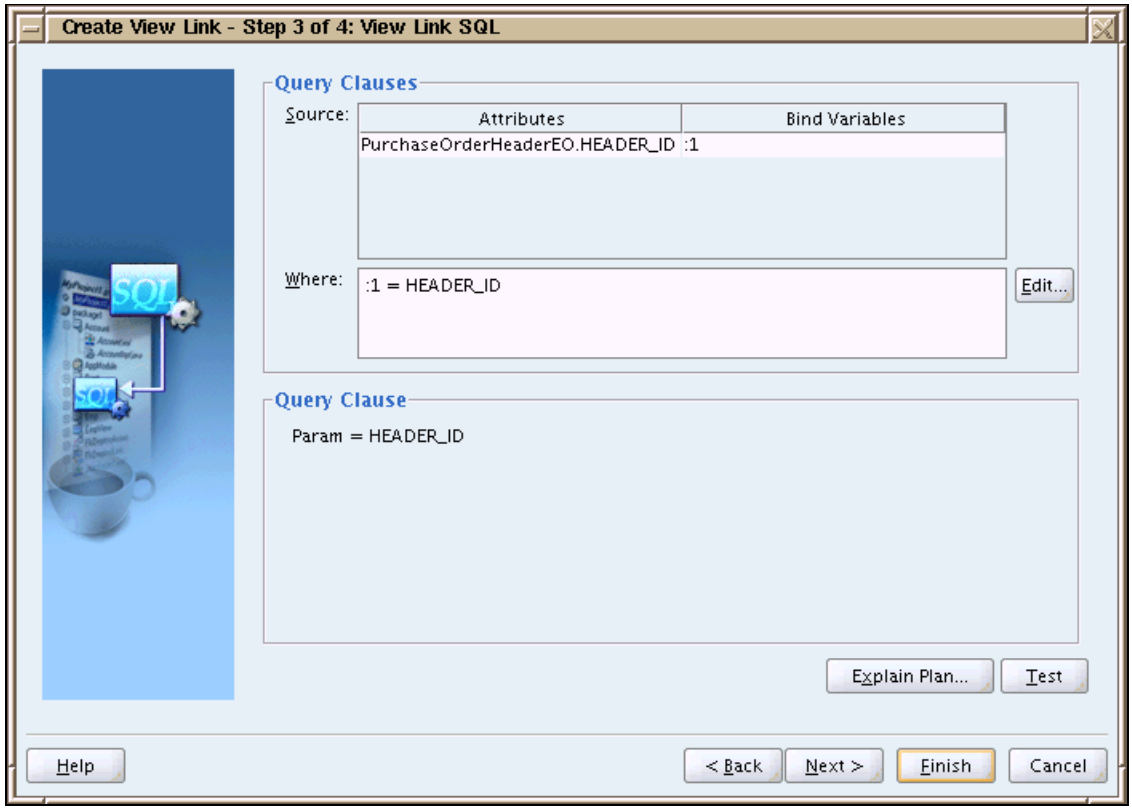
Note: You can access context-specific Help while in any of the BC4J wizards by selecting the **F1** key on your keyboard.

To create a new view link in a Business Components (BC4J) package:

1. In the JDeveloper Navigator, select the BC4J package where you want to create your view link.
2. From the main menu, choose **File > New** to open the **New Gallery**.
3. In the **Categories** tree, expand the **Business Tier** node, and select **Business Components (BC4J)**.
4. In the **Items** list, select **View Link** to open the **View Link Wizard**. Note that you can also right-click on the BC4J package and select **New View Link** to navigate directly to the **View Link Wizard**.
5. In the **Name** page (Step 1 of 4):
    - Specify the view link's **Name** in accordance with the OA Framework File / Directory / Package Structure standards.
    - Verify that you have selected the right BC4J package.
    - Do NOT enter a value in the **Extends View Link** field unless you are deliberately subclassing one of your own view objects.
    - Select **Next** to proceed.
6. In the **View Objects** page (Step 2 of 4), specify the view link's cardinality (for example, is it a one-to-many relationship?) and select the source and destination view object attributes that define the view link. Select the **Add** button to create the view link (repeat as necessary for a multi-key relationship). Select **Next** to proceed.
7. In the **View Link SQL** page (Step 3 of 4), review the `WHERE` clause that BC4J is going to create for you to ensure that it is correct.

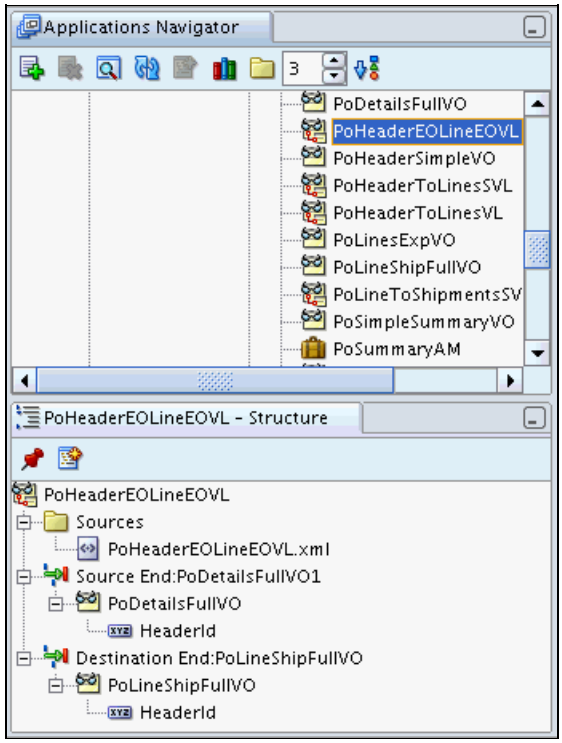
Tip: If there are no primary keys specified in the source and destination view objects, BC4J cannot properly create a `WHERE` clause. If these fields are disabled, check your view object definitions.

Figure 9: View Link Wizard showing a generated WHERE clause



10. In the **View Link Properties** page (Step 4 of 4,) indicate whether you want to generate accessors from the source to the destination and vice versa.
11. Select **Finish** to create your view link. BC4J will create an XML definition file as shown in Figure 10. Note that you can quickly view the underlying relationship by selecting the view link in the Applications Navigator.

Figure 10: JDeveloper Applications Navigator and Structure pane view of a selected view link



Programmatic Control

Since view links have no implementation, you do not write any code for them. In Chapter 5, we discuss how to access a view object using a view link.

You can however, create view links dynamically by using the `oracle.jbo.ApplicationModule createViewLinkBetweenViewObjects` API. Refer to the corresponding JavaDoc for an example of how to use this method.

Note: Both the Master and Detail view objects participating in a programmatically created view link should belong to the same application module instance.

Application Modules

This introduces the basics of application module creation/usage. See Application Modules in Detail in Chapter 5 for additional information about working with these objects.

Application Module Uses

The following is a list the distinct roles that application modules can play in your application:

- **UI Root Application Module** - Establishes the transaction context for one or several related UI pages. Every page has a root application module which includes any view objects and nested application modules used by the page.
- **UI Shared Region Application Module** - Any UI region with *created for use* in multiple pages should have its own containing application module. When this region is used in a page, OA Framework automatically nests its application module beneath the page's root application module. See Implementing the View for additional information on creating shared regions.
- **UI List of Values Application Module** - This is a special case of the previous role. When you create List of Values (LOV) view objects, you add these components to an application module dedicated to the job of aggregating LOVs for a given package.
- **Validation Application Module** - A *validation* application module aggregates related view objects created for the purpose of performing lightweight SQL validations. Entity objects and experts make use of validation application modules, which have nothing to do with the user interface. See the Entity Objects, Entity Experts, 'Validation' Application Modules and 'Validation' View Objects topic below for additional information.

Declarative Implementation

For additional information about Application Module Wizard properties not specifically described here, see the JDeveloper documentation.

Note: You can access context-specific Help while in any of the BC4J wizards by selecting the **F1** key on your keyboard.

Creating New Application Modules

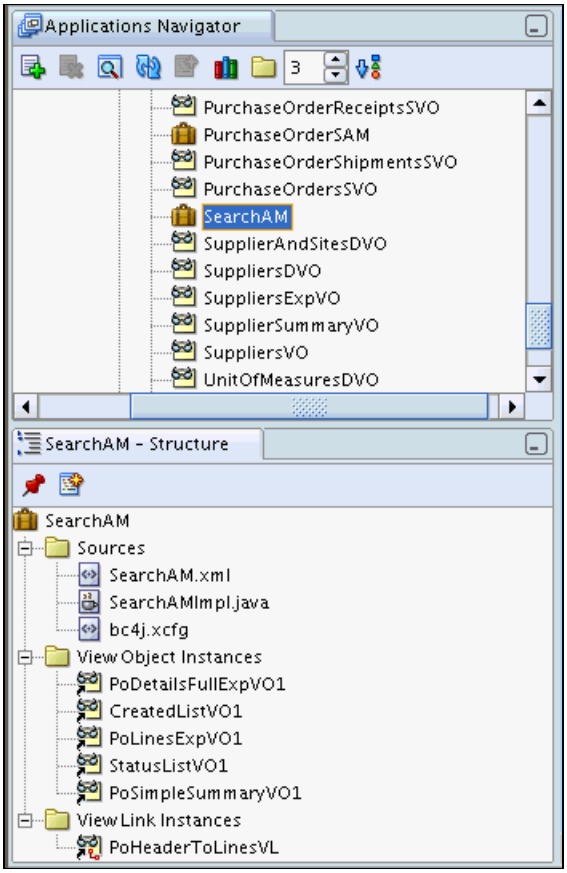
Note: Create a shared application module that subclasses `oracle.apps.fnd.framework.server.OAApplicationModuleImpl`, which you then subclass to create more specific behaviors.

To create a new application module in a Business Components (BC4J) package:

1. In the JDeveloper Navigator, select the BC4J package where you want to create your application module.
2. From the main menu, choose **File > New** to open the **New Gallery**.
3. In the **Categories** tree, expand the **Business Tier** node, and select **Business Components (BC4J)**.
4. In the **Items** list, select **Application Module** to open the **Application Module Wizard**. Note that you can also right-click on the BC4J package and select **New Application Module** to navigate directly to the **Application Module Wizard**.
5. In the **Name** page (Step 1 of 4):
   - Specify the application module's **Name** in accordance with the OA Framework File / Directory / Package Structure standards.
   - Verify that you have selected the right BC4J package.
   - Do NOT enter a value in the **Extends Application Module** field unless you are deliberately subclassing one of your own application modules.
   - Select **Next** until you get to Step 4.
6. In the **Java** page (Step 4 of 4), deselect the **Generate Java File(s)** checkbox ONLY if you are certain that you won't be writing any code for your application module (you can always delete the class later if you find that you don't need it, so it's probably best to simply generate it at this point unless you are creating a simple container for LOV view objects). If you do want to generate an implementation class for your application module, select the **Class Extends...** button to verify that you will be subclassing oracle.apps.fnd.framework.server.OAApplicationModuleImpl.
7. Select **Finish** to create your application module. BC4J will create an XML definition and implementation file as shown in Figure 11.

   Note: You can quickly view the underlying contents by selecting the application module in the Applications Navigator.

Figure 11: JDeveloper Applications Navigator and Structure pane view of a selected application module



At this point, you are not quite finished with the creation process. To proceed, edit the application module as follows:

1. In the JDeveloper Navigator, select the application module that you just created, right-click and select **Edit *<appmodule_name>*...**.
2. Navigate to the **Tuning** page. Verify that the **Customize Runtime Instantiation Behavior** checkbox is checked, and the **Lazy Loading** radio button is selected (note that you should also review Application Modules in Detail for a detailed description of the Lazy Loading feature and several use case considerations).
3. Select **OK** to save your changes.

Generating Application Module Interfaces

To generate an application module interface so you can invoke typed methods directly (with compile-time checking) instead of calling `invokeMethod()`, you must first create the methods that you want to expose to the client. Then:

1. In the JDeveloper Navigator, select the application module that you just created, right-click and select **Edit *<appmodule_name>*...**.
2. In the **Application Module Editor**, select **Client Interface.**
3. Select the methods you want to be able to invoke remotely in the **Available** list and shuttle them to the **Selected** list.
4. Select **OK** to create your interface.

JDeveloper automatically creates an interface in the correct package and with the correct name per the OA Framework File Standards.

Adding View Objects

Tip: When you create a view object for a particular purpose, immediately add it to the appropriate application module so you don't forget to do it later.

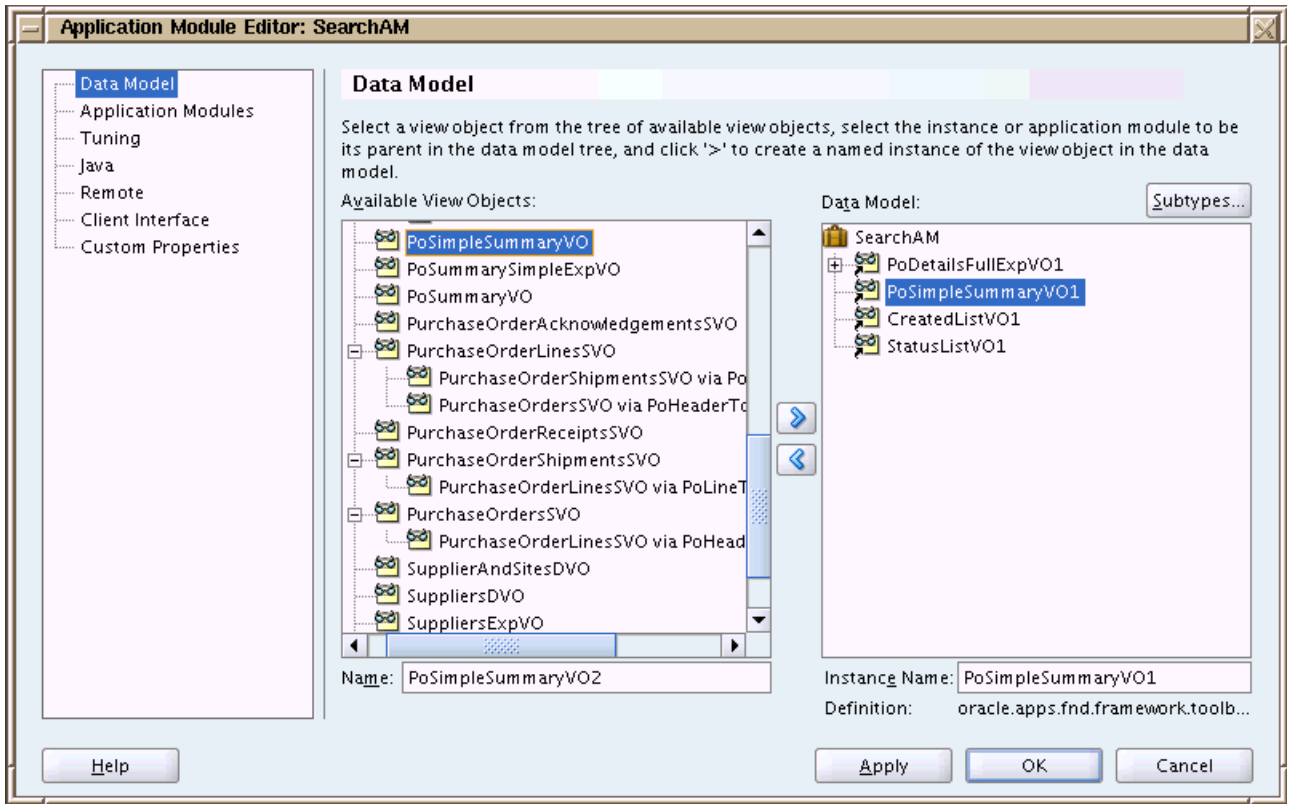
All view objects are used in the context of a given application module.

View objects are instantiated on an "as needed" basis (in previous releases, BC4J instantiated all the view objects associated with a given application module when the application module was created). For example, if you write code to find a specific view object so you can query it, or a UI page renders with items bound to view object attributes, BC4J automatically instantiates the necessary view objects. If a view object that is associated with an application module is not required at runtime, it is not instantiated.

To create this relationship declaratively for a given view object and application module:

1. In the JDeveloper Navigator, select the application module that you just created, right-click and select **Edit *<appmodule_name>*...**.
2. In the **Application Module Editor**, select **Data Model**.
3. In the **Data Model** page, select the view objects that you want to include from the **Available View Objects** list and shuttle them to the **Data Model** list.
4. Optionally change the default view **Instance Name**. A single view object can be added to the same application module multiple times (for example, you could perform the same query in multiple regions within the same UI task/module). Each view object instance has a unique identifier; this unique identifier is the **Instance Name**. When you add a view object to an application module, BC4J creates the default instance name by appending an integer to the view object's name as shown in the Figure 12. To edit this value, simply select the view object in the **Data Model** list and make your changes in the updateable field below the list.

Figure 12: Application Module Editor Data Model page showing a default view Instance Name

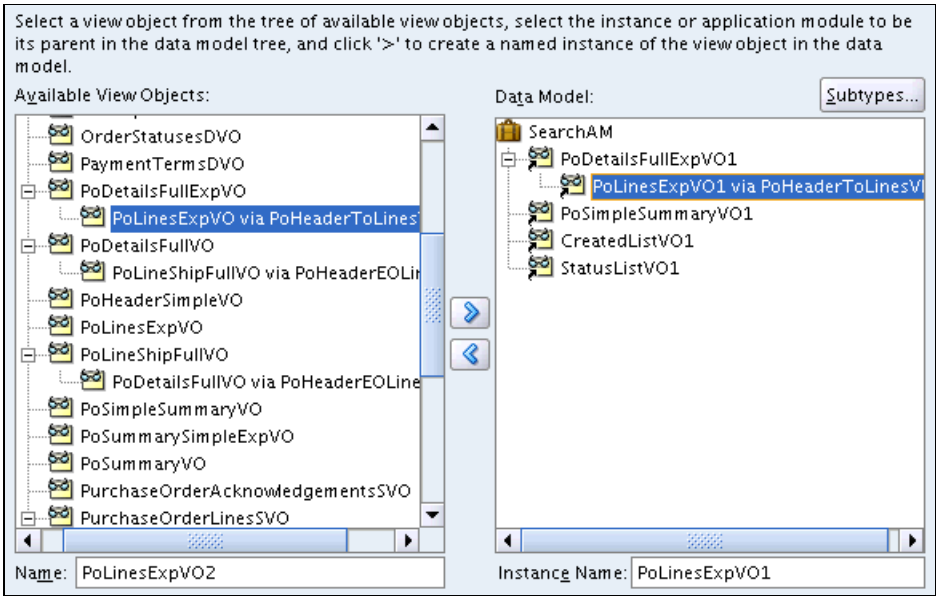


Note: To add a detail view object (accessed via a view link) to the application module, follow these steps in the **Edit Application Module** dialog. You must adhere to these instructions to properly access the detail view object; it's not sufficient to simply add the detail view object as a peer to the master view object.

1. Select the master view object in the **Data Model** view
2. Select the detail view object in the **Available View Objects** view and shuttle it to the **Data Model** view

If you added the detail view object correctly, it will appear as shown in Figure 13.

Figure 13: Application Module Editor Data Model page showing a detail view object added to its master via a view link



Adding Nested Application Modules

You can nest application modules to any arbitrary level.

Nested application modules are instantiated on an "as needed" basis (in previous releases, BC4J instantiated all the nested application modules when the containing application module was created). For example, if you do a findApplicationModule , BC4J will instantiate the object. If a nested application module is never accessed, it is not created.

To add a nested application module to your application module:

1. In the JDeveloper Navigator, select the application module that you just created, right-click and select **Edit *<appmodule_name>*...**.
2. In the **Application Module Editor**, select **Application Modules**.
3. In the **Application Modules** page, select the application module(s) that you want to include from the **Available** list and shuttle them to the **Selected** list.
4. Optionally change the default application module **Instance Name** as described for view objects above.

Programmatic Control

Do NOT code business logic, such as validations, in application modules; this should be coded in underlying entity objects instead. The application module is an appropriate place for logic that:

- Provides access to any associated BC4J objects. For example, in Implementing the Controller, you will see that controllers should never access view objects directly when they need to execute a query. Instead, they must invoke methods on the page's application module asking for a particular query to be executed.
- Performs multiple server-side actions, or spans multiple view objects as the result of a single event or method invocation. For example, code that copies all the rows from View Object A to View Object B belongs in this class.
- Returns server side values to the client that cannot be accessed from an OAPageContext. If, for example, your page needs a specific server value to determine if a region should be rendered or an item should be read-only, the application module should provide this information.
- Calls special PL/SQL routines.

  Tip: If the PL/SQL routines are used for validation and processing of individual rows (or a set of rows), then you should use PL/SQL-based entity objects instead. See Chapter 5 for additional information about using PL/SQL entity objects.

Method Naming

Any application module methods that directly supports the UI should be named for the corresponding UI "events." For example, if the user presses a **Create** button, the application module method should be named "create" and shown in the following examples.

Note: Corresponding controller invocations of all the following examples are included in Implementing the Controller.

Entity Object Create Example

The following example illustrates an application module method that creates and inserts a row into the `SuppliersVO` view object. This particular view object is based on the `SupplierEOImpl` entity object, so BC4J instantiates this behind the scenes when the

row is created.

```
public void createSupplier()
{
  OAViewObject vo = getSuppliersVO();
  Row row = vo.createRow();
  vo.insertRow();

  // As specified in OA Framework Model Coding Standards,
  // set the new row state to STATUS_INITIALIZED.

  row.setNewRowState(Row.STATUS_INITIALIZED);
}
```

View Object Query Examples

This shows an application module method that queries the `SuppliersVO` view object using search criteria passed from the client.

```
public void query(String supplierName, String onHoldFlag, String supplierNumber)
{

 SuppliersExpVOImpl vo = getSuppliersExpVO();

  if (vo == null)
  {
    MessageToken[] tokens = { new MessageToken("OBJECT_NAME", "SuppliersExpVO")};
    throw new OAException("ICX", "FWK_TBX_OBJECT_NOT_FOUND", tokens);
  }

  vo.initQuery(supplierName, onHoldFlag, supplierNumber);

} // end query()
```

This example illustrates a query that initializes a page when the user navigates to it. Note the browser Back button check to ensure that a query isn't executed needlessly. See Chapter 6 Supporting the Browser Back Button for additional information).

```
public void init(String status)
{
  PoSimpleSummaryVOImpl vo = getPoSimpleSummaryVO();
  if (vo == null)
  {
    MessageToken[] tokens = { new MessageToken("OBJECT_NAME", "PoSimpleSummaryVO")};
    throw new OAException("ICX", "FWK_TBX_OBJECT_NOT_FOUND",tokens);
  }
  // Follows Back Button standard of never performing a blind query without
  // checking to see if this is necessary.

  if (!vo.isPreparedForExecution())
  {
    vo.initQuery(status);
  }
} // end init()
```

Entity Object Delete Example

This illustrates how to search a view object row set for a single selected object so that the entity object can be deleted.

```
/**
 * Deletes a purchase order from the PoSimpleSummaryVO using the
 * poHeaderId parameter.
 */
public void delete(String poHeaderId)
{
  // First, we need to find the selected purchase order in our VO.
  // When we find it, we call remove( ) on the row which in turn
  // calls remove on the associated PurchaseOrderHeaderEOImpl object.

  int poToDelete = Integer.parseInt(poHeaderId);

  OAViewObject vo = getPoSimpleSummaryVO();
  PoSimpleSummaryVORowImpl row = null;

  // This tells us the number of rows that have been fetched in the
```

```
  // row set, and will not pull additional rows in like some of the
  // other "get count" methods.

  int fetchedRowCount = vo.getFetchedRowCount();

  // We use a separate iterator -- even though we could step through the
  // rows without it -- because we don't want to affect row currency.
  // Note that there are also convenience methods for finding matching rows
  // in a view object (see javadoc).

  RowSetIterator deleteIter = vo.createRowSetIterator("deleteIter");

  if (fetchedRowCount > 0)
  {
    deleteIter.setRangeStart(0);
    deleteIter.setRangeSize(fetchedRowCount);

        for (int i = 0; i < fetchedRowCount; i++)
    {
      row = (PoSimpleSummaryVORowImpl)deleteIter.getRowAtRangeIndex(i);

          // For performance reasons, we generate ViewRowImpls for all
      // View Objects. When we need to obtain an attribute value,
      // we use the named accessors instead of a generic String lookup.

      // Number primaryKey = (Number)row.getAttribute("HeaderId");
      Number primaryKey = row.getHeaderId();

          if (primaryKey.compareTo(poToDelete) == 0)
      {
        row.remove();
        getTransaction().commit();
        break; // only one possible selected row in this case
      }
    }
  }

  deleteIter.closeRowSetIterator();

} // end deletePurchaseOrder()
```

Custom Action Example ("Approve")

This illustrates how to search a view object row set for one or more selected objects to call a custom entity object event.

```
/**
 * Steps through the POSimpleSummaryVO to look for selected rows. For
 * each selected row, this calls the approve( ) method on the
 * PurchaseOrderHeaderEOImpl class.
 */
public void approvePurchaseOrders( )
{
  // To call a custom method on an Entity Object you should add a wrapper
  // in the VO's *RowImpl class (see
  // oracle.apps.fnd.framework.toolbox.schema.server.PoSimpleSumaryVORowImpl).

  OAViewObject vo = getPoSimpleSummaryVO();
  PoSimpleSummaryVORowImpl row = null;

  int matches = 0;

  // This tells us the number of rows that have been fetched in the
  // row set, and will not pull additional rows in like some of the
  // other "get count" methods.
  // Note that there are also convenience methods for finding matching rows
  // in a view object (see javadoc).

  int fetchedRowCount = vo.getFetchedRowCount();

  // We use a separate iterator -- even though we could step through the
  // rows without it -- because we don't want to affect row currency.

  RowSetIterator approveIter = vo.createRowSetIterator("approveIter");
```

```
  if (fetchedRowCount > 0)
  {
    approveIter.setRangeStart(0);
    approveIter.setRangeSize(fetchedRowCount);

        for (int i = 0; i < fetchedRowCount; i++)
    {
      // For every row with a selected checkbox, we want call
      // the approve( ) wrapper on the POSimpleSummaryVORowImpl which
      // in turn calls the approve ) method on the PurchaseOrderHeaderEOImpl.

      row = (PoSimpleSummaryVORowImpl)approveIter.getRowAtRangeIndex(i);

          // For performance reasons, we generate ViewRowImpls for all
      // View Objects. When we need to obtain an attribute value,
      // we use the named accessors instead of a generic String lookup.

      // String selectFlag = (String)row.getAttribute("SelectFlag");
      String selectFlag = row.getSelectFlag();

      if ("Y"Equals(selectFlag))
      {
        row.approve( );
        matches++;
      }
    }
  }

  approveIter.closeRowSetIterator();

  // If the user didn't actually select any rows, display an error message.

  if (matches > 0)
  {
    getTransaction().commit();
  }
  else
  {
    throw new OAException("ICX", "FWK_TBX_T_SELECT_FOR_APPROVE");
  }

} // end approve()
```

Commit Example

```
/**
 * Provides a "commit" wrapper so UI controller code doesn't need to
 * get a handle to the transaction itself which is a violation of the

 * client/sever tier separation rules.
 */
public void apply()
{
  getTransaction().commit();

} // end apply()
```

Testing Your Application Modules

Once you finish adding your view objects to your application modules, you can use the Business Component Browser (BC4J Tester) to run your view objects before you build an OA Framework UI for them, or write any code to support your BC4J objects. For example, you can query view objects (including the ability to navigate through master rows to query children linked with a view link).

See Testing OA Framework Applications for instructions on how to enable this utility.

Entity Objects, Entity Experts, 'Validation' Application Modules and 'Validation' View Objects

For detailed information about using entity objects, entity experts, validation application modules and validation view objects together, see Chapter 5. This section simply introduces the objects and the roles they play in an application.

Validation View Objects

When you implement business logic in your entity objects, you will frequently find that you need to execute some simple SQL statements, and not just for pure validation purposes. For example, a purchase order header has many lines. Each line is assigned a unique line number. This number is defined as *the current maximum line number for the entire purchase order + 1*. At runtime, we need to query the database to find out what the maximum line number is for a given purchase order header:

SELECT MAX(LINE_NUMBER) FROM FWK_TBX_PO_LINES WHERE HEADER_ID = :1;

Whenever you need to execute SQL like this, you can create a view object dynamically from a SQL statement, or you can predefine a declarative view object for it. That being said, OA Framework Model Coding Standards require that you use the declarative strategy in this case since it is more performant: a view object is cached in its respective application module, which allows entity object code to reuse it (and the underlying JDBC prepared statement) by simply rebinding and re-execute the query. This is an important performance benefit since validation routines are called repeatedly.

Implementation

From an implementation standpoint, *validation* view objects are no different from *regular* view objects; they are differentiated only by the use case.

Validation Application Modules (VAMs)

Predefined view objects must be assigned to an application module so that they can be accessed at runtime. In other words, view objects do not exist outside the context of an application module.

Since entity objects (and their associated validation view objects) can be shared by multiple UI clients (and the root application modules should be considered UI-specific), it is not appropriate to use the root application module for a particular page to hold your validation view objects. Instead, to group these utility view objects into meaningful, reusable units, create a *validation* application module per business object to hold them. A business object is defined the top-level entity object in a composition, or a single entity object if it stands alone. For example, the OA Framework ToolBox Tutorial purchase order is comprised of 3 entity objects, but the `PurchaseOrderHeaderEOImpl` class represents the purchase order business object.

For example, in the OA Framework ToolBox Tutorial, we created a business object-level validation application module called `PurchaseOrderVAM` and added all of the purchase order's validation view objects to it.

Implementation

From an implementation standpoint, *validation* application objects are no different from *regular* application objects; they are differentiated only by the use case. Create your validation application module declaratively and associate the appropriate validation view objects with it.

Entity Experts

The entity expert is a singleton defined to be a special affiliate of a business object (either the top entity object in a composition, or a standalone entity object). It includes common code called by the owning business object, or simple validation routines called by other entity objects that don't want the cost of instantiating the entity object itself. For example, a PurchaseOrderHeaderEOImpl class doesn't want to instantiate a whole SupplierEOImpl class just to find out if the supplierId foreign key it's about to set is valid. Instead, it calls an isSupplierIdValue(Number supplierId) method on the supplier's entity expert singleton -- a much lighter weight operation.

Implementation

To create an entity expert, first create a class that extends oracle.apps.fnd.framework.server.OAEntityExpert. The class should be named and packaged according to the standards published in the OA Framework File / Package / Directory Structure standards. Second, you need to associate this class with an entity object:

Note: For composition business objects, associate the expert with the top-level object. Otherwise, simply associate it with the standalone entity object.

1. In the JDeveloper System Navigator, select the entity object to which you want to attach the expert. Right-click and select **Edit *<EntityOjectName>*...**
2. Navigate to the **Properties** page to create a new property with the following characteristics:
   - Set the **Name** to ExpertClass.
   - Set the **Value** to the fully qualified name of your expert class. For example: oracle.apps.fnd.framework.toolbox.schema.server.PurchaseOrderEntityExpert.
3. Select **Add** to create your property.
4. Select **OK** to save your changes.

Reusing Business Objects

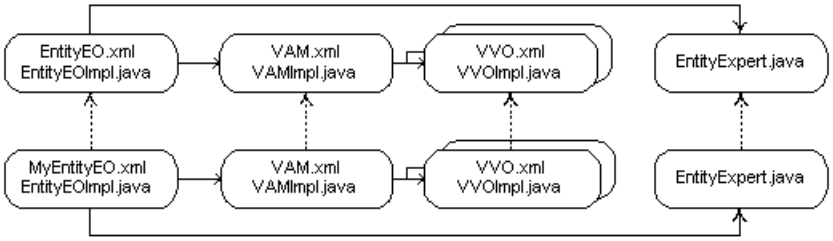
Entity Objects, Associations, Validation AMs, Validation VOs, Entity Experts

If you wish to create Entity Object, Association, Validation AM, Validation VO, or Entity Expert business objects for reuse, you should be aware of the following standards:

- The owner of the underlying data schema solely owns the corresponding Entity Objects, Associations, Validation Application Modules, Validation View Objects, and Entity Experts.
- The owner product team of these business objects must document and publish them for other product teams to reuse.
- When you create a composite association that involves your EO (YourEO) and another product team's EO (OtherBaseEO), you must first extend the OtherBaseEO into your product space (OtherExtendedEO) before creating the association between YourEO and OtherExtendedEO.
- Another product team who is consuming the business objects that you own may want to extend the validations that you provide. In such a case, the consumer product team should extend the Entity Object, Validation Application Module, Validation View Object and Entity Expert and include custom definitions and code in the extensions. When extending validation methods, make sure that you invoke `super()` at the top of the extended validation methods. Please see the Extending Business Objects subsection below for more details.

Extending Business Objects

The following diagram depicts objects that you deal with when creating business objects and the associated extensions that you want to make to extend validations.



The first row of the above diagram represents an exhaustive list of all possible objects you might create when defining an entity object. The first box illustrates that when creating an entity object, two files get generated: the metadata definition XML file and the actual implementation Java class file. Entity Objects handle attribute level and record level validations. These validations often need to use Validation View Objects (VVO). Validation Objects are grouped under Validation Application Module (VAM). Like Entity Objects, creating VVO's and VAM's, generates a metadata definition XML file and an implementation java class file for each object. Finally, Entity Objects sometimes rely on a helping class to offer among other services, a validation service optimized for usage by other Entity Objects. This helping class is the Entity Expert and linked to the Entity Object through an entity object property.

The above diagram illustrates a case where all business objects are extended. That is not necessarily always the case. In most cases, you may be satisfied with extending just some of these objects. Note that you should never edit the base definition of an object or make a copy of a base object. You should always extend the relevant object and use substitution/factory mechanisms to reference the extended objects.

For example you may be satisfied with extending the Entity Expert to override a validation method such as `isSupplierValid`. In this case, note that it is not wise to reference the extended Entity Expert (MyEntityExpert) directly from the base entity object (EntityEO.XML) as such an approach does not survive upgrades. A better approach requires extending the base entity object using the JDeveloper entity object creation wizard and updating the entity expert property on the extended entity object to point to the extended Entity Expert.

Another approach is to simply extend the entity object through the JDeveloper entity object creation wizard to add an extra validation to the `OtherExtendedEOImpl` class (make sure you invoke `super()` first) that doesn't require any additional validation view objects.

Note that business object extensions are placed under the consumer product teams' package hierarchy.

View Objects, View Links

If you wish to create View Object or View Link business objects for reuse, you should be aware of the following standards:

- View Objects and View Links are created for a particular user interface or business logic. Having said that, a product team that owns these objects may choose to publish certain View Objects and View Links for public consumption.
- The owning product team of a public view object must provide the necessary documentation and must guarantee the objects' contract (WHERE clause, attributes, and so on).
- Other product teams may extend a public base view object's capabilities by using the JDeveloper view object creation wizard.
- You can also extend a public view object definition to include extra attributes, if desired.

For more information on Extensibility, refer to Chapter 9: Extending and Deploying OA Framework Applications.

Implementing the View

Overview

This document describes how to implement an application's view components in generic terms. It does not describe how to implement specific UI features. For this information, see individual topics in Chapter 4: Implementing Specific UI Features.

Contents

- Designing the User Interface
- Pages
- Reusable Components
- Attribute Sets
- URL Parameters: Tokens, Encryption, Encoding
- Style Sheets
- Accessibility
- Internationalization
- Model Interaction
- Menus and Page Security

Prerequisite Reading

This document assumes that you have read the following in the OA Framework Developer's Guide:

- Building "Hello, World!"
- JSP Application Primer
- Anatomy of an OA Framework Page
- OA Framework State Management
- Implementing the Model

Designing the User Interface

All OA Framework applications must be designed in accordance with the appropriate Look-and-Feel (LAF) guidelines, such as Alta or Oracle Browser Look-and-Feel (BLAF) (note that you can find information about how to do the UI design, including instructions on using UIX as a prototyping tool, language standards for the UI as well as for language standards for page titles and breadbrumbs, at this site). Compliance with these guidelines yields several benefits:

- Users interact with a consistent, predictable, attractive interface as they navigate from one application to the next.
- If properly designed for your target audience, your applications are likely to be intuitive and usable since the UI guidelines themselves have been usability tested. Furthermore, the results of product team usability testing are considered and addressed on an ongoing basis.
- The underlying UIX beans that the OA Framework extends implement the UI guidelines. If your application complies with the UI guidelines, you can typically use the OA web beans "out of the box" without extensive programmatic effort.

For Oracle E-Business Suite applications, any deviations from these guidelines must be approved by the corporate UI Design and Usability team. Generally, valid deviations are driven by unique product requirements (which often introduce designs that are eventually rolled into the general standards to the benefit of all product teams). It is not acceptable to ignore individual standards simply because you disagree with them.

Pages

The basic steps for creating pages, regions and items are outlined in Chapter 2: Building "Hello, World!", and in the JDeveloper OA Extension Help. For information about implementing feature-specific regions and items, see Chapter 4.

Coding Standards Compliance

Before creating any OA components, you should read the following documents carefully:

- Chapter 8: OA Framework Naming / File / Package / Directory Structure Standards
- Chapter 8: OA Framework View Coding Standards (this includes standards for components that you create declaratively)

Key Page Layout Region Properties

Whenever you create a **pageLayout** region , pay special attention to the following properties:

- `AutoFooter` - always set this to **true** so the Applications-standard privacy and copyright links render in your page.
- `Help Target` - if you want to display help text for this page when the user selects the Help global button in your page, you must specify the help target (often the name of the help file) to display here. See Global Buttons for additional information about this.
- `AM Definition` - this is where you specify the root application module for your page. Note that you must specify the fully qualified name, such as: **oracle.apps.fnd.framework.toolbox.tutorial.server.SearchAM**
- `Function Name` - always specify the securing function for the page (see the Menus section below for additional information).
- `Window Title` - this text value is displayed in the browser's window title.

- `Title` - this text value renders as the page header. It's also used as the page's name if breadcrumbs are displayed (see Chapter 4: Breadcrumbs for detailed information about this feature).
- `Form` - always set this to **true** for a **pageLayout** region (this is the default setting), and never add additional child form beans to your page. The OA Framework supports only 1 form per page.
- `Browser Caching Disabled` - introduced in Release 12.2.5, set this to **true** or **false** to have better control over when the page should be cached when the Force Page Refresh profile option is set to **Yes**. This profile impacts all pages and can cause performance issues. The following table summarizes the effects of this property when set in conjunction with the profile option.

Browser Caching Disabled	Force Page Refresh	Cache-Control
True	True	Will be set
False	True	Will be set
True	False	Will be set
False	False	Will not be set

- `pageLayout Components` - as mentioned in Anatomy of an OA Framework Page, pages include special "named" components (also called named children), one of which is the branding image . To associate a branding image with your page, select the pageLayout region or the pageLayout Components node in the Structure pane and use the right mouse button to select **New > productBranding** from the context menu. JDeveloper automatically creates an image item whose `Image URI` property should be set to <imageName>.gif.

Key Item Properties

Since each item type has a distinct set of properties (often including properties that are unique to the item), it's impossible to introduce each and every relevant property. Instead, this section introduces some of the common properties that you should understand since you will set them frequently.

- `Extends` - For items (and regions as well), this indicates that the new item extends an existing item. This is discussed in more detail below.
- `Attribute Set` - A named set of properties applied to quickly configure an item. This is discussed in more detail below.
- `Destination URI` - For items that support navigation, this is the navigation target. For example: `OA.jsp?page=/oracle/apps/fnd/framework/toolbox/tutorial/webui/PoDetailsPG&retainAM=Y`.
- (Client Action) `Action Type` - Indicates whether the item is capable of submitting the form, or causing a partial page rendering (PPR) event. See Chapter 4's Dynamic User Interface and Declarative Submit Form for additional information about these features.
- `CSS Class` - Indicates which cascading style sheet class to apply to the item (for many items, UIX sets this value for you automatically to comply with the LAF UI Guidelines). This is discussed in more detail below.
- `Rendered` - Indicates whether the corresponding object is included in the web bean hierarchy, and the HTML that UIX sends to the browser for rendering. For most items, this indicates whether an item displays or not, but for some items that never actually display (like a hidden developer field), this indicates whether the object exists on the page.
- `View Instance` - For items that bind to an underlying view object for reading and writing (if needed) data, this identifies the view object instance (within the context of a containing application module) to which the item binds.
- `View Attribute` - This is the view instance's attribute to which the item binds.
- `Admin Personalization` - Indicates whether the property is system administrator personalizable. See the OA Framework Personalization Guide for additional information about personalization.
- `User Personalization` - Indicates whether the property is user personalizable. See the OA Framework Personalization Guide for additional information about personalization.
- `Initial Value` - Default value for the item (note that this can be personalized by customers). See the Defaulting topic below for additional information.

Expression Language (EL)

For selected properties, the OA Framework supports the use of EL expressions to quickly bind the property to an underlying data source that provides the property's value. For example, you could bind the `Rendered` property of a button to a view object attribute to ascertain whether it should be hidden or shown when the page renders. The EL syntax for this property looks like:

```
${oa.<ViewInstanceName>.<ViewAttributeName>}
```

Tip: EL is an industry-standard expression language included in the JSP Standard Tag Library (JSTL). If you're interested in learning more about this (although this isn't necessary for the limited use in the OA Framework), searching for "expression language (EL)" on the web returns numerous resources.

The use of EL expressions is fully described in Chapter 4's Dynamic User Interface.

Reusable Components

One of the key advantages of the declarative OA Component development environment is the ease with which you can reuse common page, region and item definitions.

See the Shared Regions section in this chapter to see how a common module with its own logic and application module is created and used.

Shared Regions

Comply with Reuse Standards

If you want to create a shared region, you must comply with the following standards.

Note: A shared region can include one or more subregions.

- The top-level (shared) region must be saved in its own XML file.
- You can design your shared region to accept values from a using region. Values may be passed on the request using any of the approaches described in OA Framework State Management, or as a cached value on the page's transaction (also described in the *State Management* document).
- The shared region must be implemented to fail gracefully. For example, if appropriate parameters are not passed from a using region, the shared region should set acceptable defaults or raise a meaningful (and documented) exception.
- If the region scope is set to **Public** (see *Create a Shared Region* below for additional information about this):
    - The top-level region must have its own application module. The application module should include only those view objects that are relevant for the shared region.
    - The top-level region must have its own controller. You may associate additional controllers with subregions as necessary.
- The shared region must be fully documented as described below.

Create a Shared Region

To create a shared region :

1. In the JDeveloper Navigator, select the OA Project where you want to create your region.
2. From the main menu, choose **File > New** to open the **New Object Gallery**.
3. In the **Categories** tree, expand the **Web Tier** node, and select **OA Components**.
4. In the **Items** list, select **Region** to open the **New Region** window.
5. Enter a **Name** and a **Package** in accordance with the OA Framework File Standards, and specify the `Style` of region that you want to create (select your style carefully since you cannot change it once you create the region). Select **OK** to save create your <Package>.<Name>.xml OA Component document.
6. Select the new region in the JDeveloper Structure pane and set the `Documentation Comment` property with the content described below.
7. Set the `Scope` property as appropriate for your planned use of the shared region. For example, for a private region to be used exclusively within the current package, select the **Current Package** option (note that you can limit access to just your product if you wish). Alternatively, set this to **Public** to let anyone use it in any page.
8. Set the other properties in your region.
9. When it is time to package and ship your shared region, you must generate the region's Javadoc-like HTML documentation using the Developer Documentation Generator Tool in JDeveloper (see **Getting Started with the OA Extension > Command Line Tools for the OA Extension > About the Developer Documentation Generator Tool** in the Oracle JDeveloper online Help for additional information).

Warning: Pay strict attention to the naming standards in the OA Framework File Standards document when naming your shared region and any of its items and subregions. Since all OA components in a page must have a unique name, adherence to the naming standards will help ensure that your reusable region truly can be reused.

Note: Due to naming restrictions, a single region cannot be used more than once in a page.

Documentation Comment Content

You must add the following content in the region's `Documentation Comment` property:

```
/**
 * Controller for: <shared page/region name including package>
 *
 * Scope: < private (for owning product team use only -- this is the default scope),
 * public (for use by anyone) or oracle (for Oracle E-Business Suite development use only)>
 *
 * Usage: < describe the component's purpose and use, including any error messages that
 * might be raised>
 *
 * @param <object parameter 1> <object parameter 1 description and acceptable values>
```

```
 * @param <object parameter 2> <object parameter 2 description and acceptable values>
 * @param <object parameter N> <object parameter N description and acceptable values>
 * @see <optionally include references to other objects such as other shared children controllers, if any>
 */
```

Note: When describing a parameter, clearly indicate whether the parameter should be passed to the region on the request, or on the application module transaction.

The following example illustrates appropriate content for a shared component controller:

```
Controller for: ford.oracle.apps.xyz.webui.FordDistributorAddressRN

Scope: Public

Usage: Implements a localized address region for distributors.

@param: distID disitributor ID which is passed on the request;
               required to initialize the region
@param: locale locale which is passed on the request; required to
               correctly localize the address
@param: submit passed on the request; if specified, this region will
commit its changes.
```

Extend a Reusable Region

As mentioned in Anatomy of an OA Framework page, to use a shared region in your page, you simply extend it:

1. In the JDeveloper Structure pane, select the region to which you want to add the shared region.
2. Use your right mouse button to select **New > Region** from the context menu.
3. Place your cursor in the new region's `Extends` field in the Property Inspector and select the **...** button to open the component browser. Search or browse as you prefer, select the region you want to extend from the results list and select **OK** to make your choice.
4. JDeveloper enters the fully qualified name of the shared region in the `Extends` field (for example, /oracle/apps/fnd/framework/toolbox/tutorial/webui/PoSummaryRN). Note that you can edit most of the properties of the base region that you created (you can't change its `Style`), but the extended region cannot be modified. In fact, its contents render with a gray font in the Structure pane and in the Property Inspector.
5. Save your work.

Tip: When you add a shared region with its own application module to your page, the OA Framework automatically nests the component application module beneath your root application module. You don't need to create any explicit design-time relationships between these application modules.

To clear an extension, place your cursor in the `Extends` field and select the Property Inspector's **Set to Default** toolbar button.

Special Case: List of Values (LOV)

Although the implementation steps are described elsewhere (see the List of Values topic in Chapter 4), it's worth noting that the LOV can be implemented as a special kind of shared region (you can also create a single-use LOV):

- You create a reusable List of Values using the same procedure as you would for any other other shared region, although it does not require an associated controller.
- When you want to use the shared LOV in a page, you do not extend it as described above. Instead, you set the base page field's `External LOV` property and configure the data exchange between the base page and the LOV.

Shared Pages

A page is really just a shared region whose top-level region happens to be designated as a **pageLayout** component. As such, a shared page should follow all the region creation standards and instructions described above.

- If you want to reuse a standalone page or page flow, simply create a new menu function and point it to the main page (menu functions are discussed below).
- If you want to insert a shared page into another page flow with a different root application module, you must create a new page, and then extend the shared page's contents below the **pageLayout** region. Remember to set the correct root application module on your new page.

Shared Items

You can also extend individual items from any region, although we recommend that you place items that you intend to share in a reusable region. Sharing the containing region will help ensure that someone doesn't change properties in any arbitrary item without realizing that the change could adversely impact pages using the item.

- In the JDeveloper Structure pane, select the item that will extend another item.
- Place your cursor in the item's `Extends` field in the Property Inspector and select the **...** button to open the component browser. Search or browse as you prefer, select the item you want to extend from the results list, and select **OK** to make your choice.

- JDeveloper enters the fully qualified name of the item in the `Extends` field (for example, /oracle/apps/fnd/framework/toolbox/tutorial/webui/PoSummaryRN.OrderTotal). Note that you can edit most of the extended item's properties, but you can't change its `Item Style`.
- Save your work.

Sharing Logistics

Oracle E-Business Suite product teams should produce internal ARUs to share reusable objects. For teams with a large number of sharable objects that are still changing rapidly, please contact the OA Framework team to investigate the possibility of leveraging the OA Framework daily freeze process.

Attribute Sets

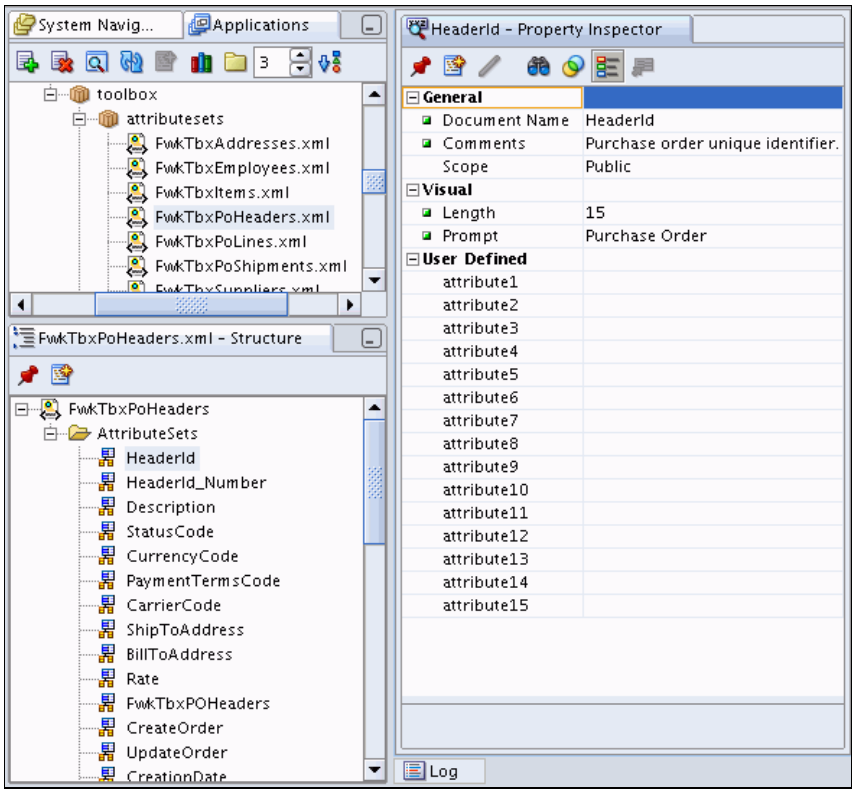
Attribute sets are named, reusable collections of properties (prompt, maximum display length, data type and so on as appropriate for the attribute set type) that can be used by any type of OA component, including regions, items, and other attribute sets. They are designed to facilitate the reuse of these components throughout Oracle E-Business Suite, which yields a significant cost savings to both Oracle and its customers:

- Oracle saves in translation and maintenance costs.
- Customers can make quick, global personalizations to their applications. Additionally, fewer UI elements translates to less middle-tier memory consumption, and ultimately, this means better performance and scalability.

In general terms, attribute sets are organized into OA component packages (individual XML package files), where you have one package file per database table in your application:

- The package name matches the underlying database table name without underscores. For example, in the OA Framework ToolBox, we have a table called FWK_TBX_PO_HEADERS. The corresponding attribute set package is named FwkTbxPoHeaders.
- Individual attribute sets are created for each displayable column in the table. TL translation column attribute sets are included with the base table, as are the displayable values for lookup codes (for example, if a table includes a lookup code for freight terms, the package would include an attribute set for the FreightTerms displayed value).
- Column-based attribute sets are named for the corresponding column. For example, in the FWK_TBX_PO_HEADERS table we have a HEADER_ID column. The corresponding attribute set is named HeaderId (this is always used in the context of the fully qualified package name as shown below, so you don't have to worry about multiple tables having HeaderId attribute sets). If there are multiple attribute sets for the same column (so the value is commonly used in several different contexts with different prompts, for example) they are differentiated with prompt suffixes as illustrated for the HeaderId case in the FwkTbxPoHeaders example below. The most common use case for the header id uses the prompt "Purchase Order." The HeaderId_Order attribute set's prompt is "Order" and the HeaderId_Num prompt is "Number."
- The table attribute set package may also include attribute sets for common region headers and buttons, which are named for their associated labels.

Figure 1: Attribute sets in the FwkTbxPoHeaders.xml package.



See Creating Attribute Sets for detailed instructions if you need to create or maintain your own attribute sets.

Using Attribute Sets

Oracle E-Business Suite developers should use attribute sets in all of the following cases:

- All items associated with `table.column` data values (including any transient items that relate to table values). For example, both a Supplier name search field and a Supplier name data entry field should use the same attribute set.
- All common buttons (Go, Apply, Cancel and so on). This also ensures that you correctly inherit the standard accelerator keys for these buttons.

  Tip: The OA Framework common buttons attribute set package is /oracle/apps/fnd/attributesets/Buttons/<AttributeSetName>.
- All common table action columns (like Delete, Update and so on) should use the corresponding OA Framework button attribute set.
- Any shared buttons in your product for which attribute sets have been created; you should not be creating or using attribute sets for single-use buttons
- Any shared header regions in your product for which attribute sets have been created; you should not be creating or using attribute sets for single-use headers

To use an attribute set for an item:

- In the JDeveloper Structure pane, select the item for which you want to specify an attribute set.
- Place your cursor in the item's `Attribute Set` field in the Property Inspector and select the **...** button to open the component browser. Search or browse as you prefer, select the attribute set you want to extend from the results list and select **OK** to make your choice.
- JDeveloper enters the fully qualified name of the attribute set in the `Attribute Set` field (for example, /oracle/apps/fnd/attributesets/Buttons/Apply).

Although you can override attribute set properties when you associate them with your items, you should avoid doing this for translated values. If you find, for example, that you need a variant of a preexisting attribute set to display a new prompt, you should create an additional attribute set as described in the Creating Attribute Sets document. Overriding something like a display width is fine.

To clear an attribute set, place your cursor in the `Attribute Set` field and select the Property Inspector's **Set to Default** toolbar button.

Programmatic Access to Attribute Sets

You can also access attribute sets in your controller . For example, the following code shows how to obtain a translated prompt from the common Create button attribute set:

```
import oracle.apps.fnd.framework.webui.AttributeSet;

...

public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
 super.processRequest(pageContext, webBean);

 AttributeSet attrSet =
    new AttributeSet(pageContext, "/oracle/apps/fnd/attributesets/Buttons/Create");
 String createPrompt = (String)attrSet.getAttributeValue(pageContext, PROMPT_ATTR);

}
```

URL Parameters: Tokens, Encryption, Encoding

Tokens

When you specify URL parameters in your declarative page definitions, you can specify both literal and token-substituted values that obtain their data from associated view object attributes at rendering time (in this case, the item must be bound to a view object). This is commonly used, for example, in a table column to pass a primary key value to a detail page for querying.

Token Substitution Example (using the view object attribute name "OrderNum"):
OA.jsp?OAFunc=FWK_TBX_T_PO_PAGE&order={@OrderNum}

Literal Example: OA.jsp?OAFunc=FWK_TBX_T_PO_PAGE&order=123

Token Types

Tokens use a special character prefix to tell the OA Framework how to resolve the value at runtime (note that the concepts of "encoding " and "encryption " are described below):

- **{!*Attr*}** - encrypts the attribute value while leaving the {!} in the URL (for example, OA.jsp?...&ssn={!SSN}&...). Using OAPageContext.getParameter("ssn") will return the decrypted value.

- **{@Attr}** - encodes the attribute value while leaving the {@} in the URL (for example, OA.jsp?...&addr={@EmpAdd}&...). Using OAPageContext.getParameter("addr") to get the parameter value will return the decoded value.
- **{$Attr}** - plain token substitution (no encoding or encryption) so it's your responsibility to ensure that a substituted value does not break the URL.
- **{@@RETURN_TO_MENU}** - Can be used exactly as shown to specify the `Destination URI` property of an application component if you want it to return the user to the E-Business Suite Personal Home Page. If you need to specify this when performing a JSP forward, the corresponding constant for this is `OAWebBeanValues.RETURN_TO_MENU_URL`.
- **{@@RETURN_TO_PORTAL}** -- Can be used exactly as shown to specify the `Destination URI` property of an application component if you want it to return the user to a launching portal page. If you need to specify this when performing a JSP forward, the corresponding constant for this is `OAWebBeanValues.RETURN_TO_PORTAL_URL`.

Encoding

Any value that you specify for a request parameter must conform to HTTP syntax rules. For example, you can't pass a URL parameter value with a blank space ; the following parameter value would cause a runtime error when the corresponding URL is accessed:buyerName=John Doe.

To fix this, we *encode* these values, meaning that the encoding routine replaces problematic characters with standard substitutions as shown in this example:buyerName=John%20Doe.

- When the OA Framework adds parameters to the request (form field values, for example), it automatically encodes them.
- When you put parameters on the request during a call to a setForward* method, the OA Framework automatically encodes these values.
- When you put parameters on a URL that you assemble yourself (if, for example, you set a bean's URL by calling its setDestination method), you must encode any part of the String that could include invalid characters. To do this, you pass the String to an encode method on the `oracle.apps.fnd.framework.webui.OAUrl` utility class.

  Tip: If you manually set a URL parameter value that can't include invalid characters (for example, "value=Y") then you don't need to bother with the encoding step.
- When you put values on the request using `OAPageContext.putParameter,` you must encode the String if necessary.

The OA Framework automatically decodes parameter values when you call the OAPageContext.getParameter* methods, except for the following cases:

- When you use the "#" character for Javascript function tokens, the OA Framework encodes the token values, but it does NOT automatically decode them when you call pageContext.getParameter("<tokenName>"). To do this yourself, you'll need to use the `OAUrl` decode method on the value that `getParameter` returns.
- When you call putParameter with an encoded value, the OA Framework does not decode it. You must also use the OAUrl `decode` method in this case on the value the `getParameter` returns.

Encryption

Encryption is the process of obfuscating data to make it illegible. Since URL request parameter values may be visible to the user (and hidden form field values if the user opts to view the HTML page source), you should always encrypt sensitive data if stored in a URL parameter or a hidden field.

In addition to the declarative, token-based encryption described above, the OA Framework also provides methods in `oracle.apps.fnd.framework.webui.OAPageContext` for manually encrypting and decrypting any parameter values that you put on the request programmatically.

Note: For additional information about URL and request parameters, refer to the Oracle Application Framework URL and Request Parameters appendix.

Style Sheets

One of the reasons why OA Framework applications have a pleasing, uniform user interface is the look and feel for each and every page is defined by the style sheet appropriate for the Look and Feel (LAF). See the BLAF UI Guideline: Text and CSS Standards for a quick reference to the BLAF styles.

Using Styles

All of the regions -- and most of the items -- that you add to the page have their styles set automatically; you don't need to do anything extra (nor should you). As described above, you should be setting region and item properties ONLY if you must override the default behavior.

That said, there are several cases where you must set the `CSS Class` property for your items:

- If you create a **staticStyledText** item to be used for instruction text, you must set its `CSS Class` to **OraInstructionText**.
- For any text entry fields, checkboxes, poplists and radio buttons you must set the `CSS Class` to **OraFieldText**. Do *not* use **OraPromptText** for your radio buttons and checkboxes.
- If you create a **messageStyledText** item for displaying read-only data, you must set the `CSS Class` to **OraDataText** for the data to render in bold (note that you don't need to set this value for table columns)

Tip: New OA Framework developers often make the mistake of trying to significantly change "native" component rendering by changing the CSS style. If you find yourself falling into this trap (and you're frustrated because your style settings don't appear to have any impact on the bean's runtime appearance):

- Make sure you're using the right bean (region or item style) for the job.
- If you're certain you're using the right bean, check to see if it publishes a method that lets you achieve the desired result. For example, an `oracle.apps.fnd.framework.webui.beans.layout.OAHeaderBean` inherits a setSize(int size) method that lets you control the size of the header text (which is useful when rendering headers in Home page "At a Glance" regions or in side navigation "Search" regions, for example). You cannot achieve this effect by trying to set the header's `CSS Class` to **OraHeaderSubSub**.

Creating Styles

Customers

The OA Framework automatically sets custom.xss to be its main style sheet. Any customizations that you have should be added to this style sheet.

For detailed information about style sheets (including customization instructions), see Style Sheets.

E-Business Suite Application Developers

The custom.xss style sheet mentioned above includes oa.xss, which in turn includes blaf.xss.

The oa.xss style sheet is intended to include any extensions that you have to the BLAF style sheet (contact the OA Framework team if you have additions that have been approved by the UI Design and Usability team). You should NOT try to create your own style sheet.

Accessibility

OA Framework applications are *accessible*, meaning they can be used by people with disabilities such as blindness, low-vision, color blindness and deafness. In simple terms, accessible products comply with the following guidelines:

- the product must be usable without a mouse (keyboard only)
- the product must be usable by a blind user (with a screen reader or Braille reader)
- there must be no reliance on sound
- there must be no reliance on color
- there must be no reliance on animation or timing

Note: Keyboard-only users should set their Accessibility mode to "Standard" to ensure that the keyboard-only accessible version of user interface enhancements introduced in Oracle E-Business Suite Release 12.1.3 are enabled. This can be done by setting the profile option Self Service Accessibility Features/ICX_ACCESSIBILITY_FEATURES to **Standard Accessibility**.

To create accessible pages:

- Customers may follow the Section 508 Standards and the Web Content Accessibility Guidelines (WCAG).

Internationalization

OA Framework applications are designed be fully localized. For the most part, this is transparent to you as long as you comply with all the internationalization standards outlined in Chapter 8. Also see the Internationalization document in this chapter for detailed information about language, time zone, date and number support in the OA Framework.

Model Interaction

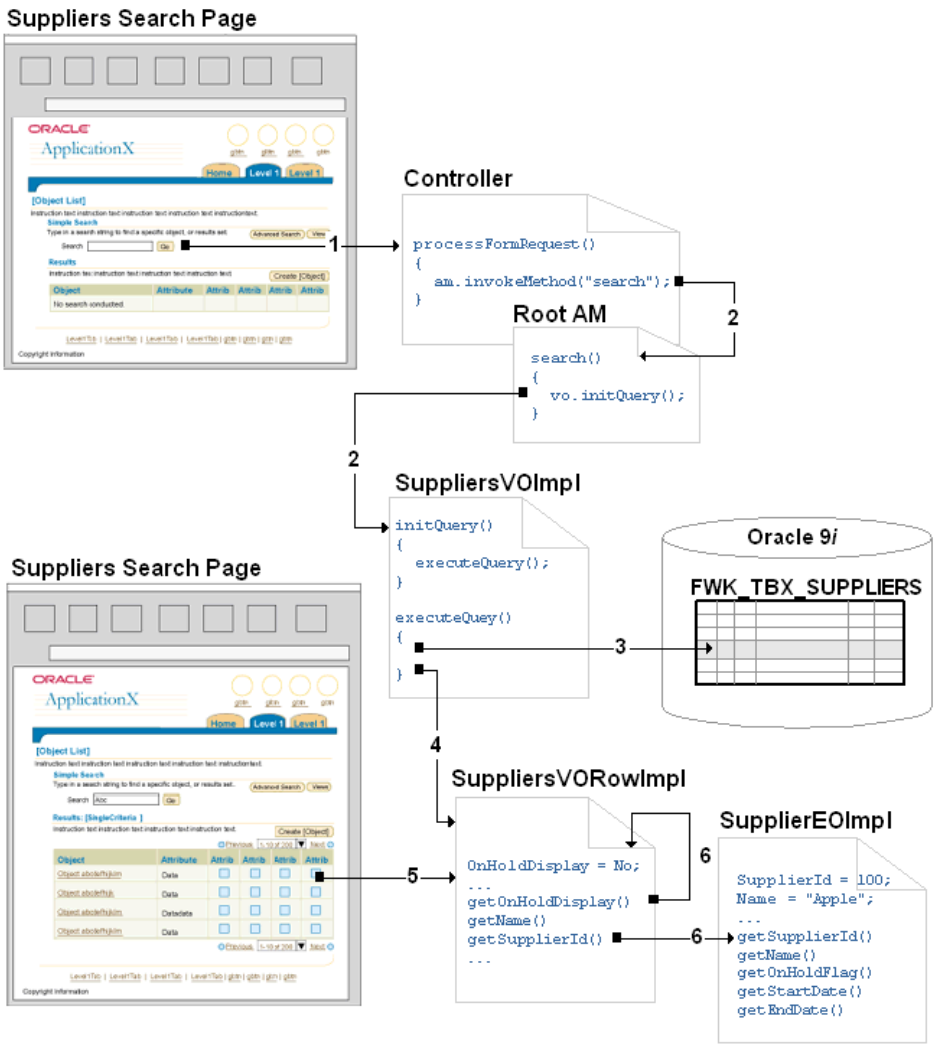
Assuming you have specified the appropriate data source bindings, the OA Framework automatically reads data from the model for display in the view, and writes user-entered data in the view back to the model. You don't need to write a single line of code (except for any validation you want to perform in the underlying entity objects, of course).

Reading Model Data

In simple terms, every time the OA Framework renders a page, it calls the current view object row's get<*AttributeName*> method for each web bean's associated view object attribute.

Consider an example page with a "Suppliers" table, which binds to a SuppliersVO view object. The SuppliersVO maps to an underlying SupplierEOImpl, although it also includes one "calculated" transient attribute ("OnHoldDisplay") that does not have a corresponding entity object attribute.

Figure 2: Illustration of how the OA Framework reads model data after a query is executed



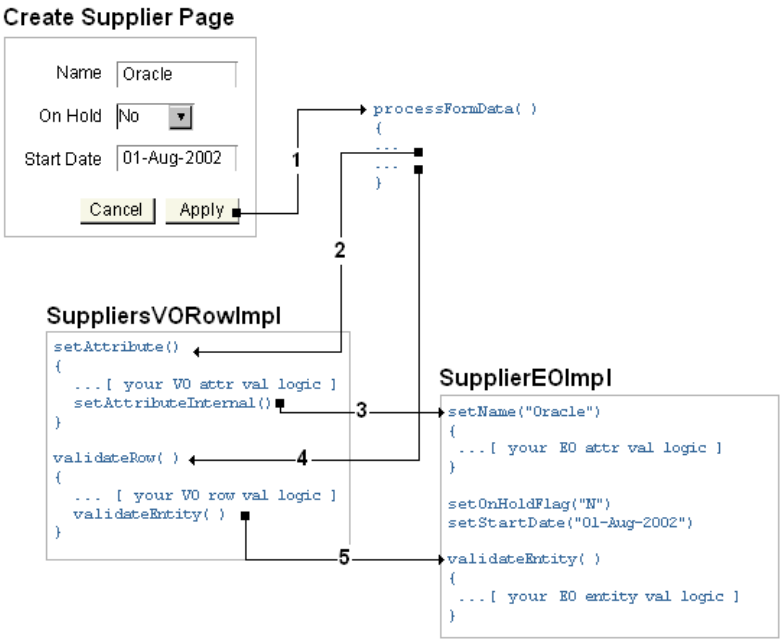
1. The user selects the "Search" region's "Go" button to populate search results in the "Suppliers" table.
2. The "Search" region's controller handles the button press by invoking a search method in the root application module, which in turn delegates to the SuppliersVOImpl class so it can query itself.
3. Within the executeQuery method, the SuppliersVOImpl view object performs its SQL SELECT in the database.
4. For each row returned in our example result set, the view object instantiates a SupplierEOImpl entity object and sets its attribute values with the query results.

   Note: Entity object-based attribute values aren't actually stored anywhere in the view object. They "live" in the entity object, and are retrieved as needed by the view object. "Calculated" (meaning the values are simply selected from a SQL statement and have no relationship to an entity object) or "Transient" view object attribute values are stored on the SuppliersVORowImpl object. See Chapter 5: Implementing Java Entity Objects for additional information about the entity object cache.
5. During page rendering (after all the query processing), the OA Framework uses the view object data bindings defined for each web bean to call the corresponding SuppliersVORowImpl object's `getAttribute("<attributeName">)` which in turns calls its get<*AttributeName*> method.
6. The SuppliersVORowImpl get<A*ttributeName*> method in turn calls the corresponding SupplierEOImpl get<*AttributeName*> method to retrieve the value. For the "calculated" OnHoldDisplay attribute, the view object row retrieves the value from its own cache.

Writing Model Data

Whenever the browser issues a POST request , the OA Framework automatically writes any user-entered form values back to the underlying view objects, which in turn update any corresponding entity objects as shown below.

Figure 3: HTTP POST Data Cycle



Note: The following steps assume that the entity object for the row has already been successfully instantiated and initialized (such as in the create method on the underlying view object for the page when the user originally comes into a Create page. The view object create method calls the corresponding create method on the entity object behind the scenes).

1. UIX performs `onSubmit` Javascript validation (required fields, data types, formats) and issues the POST request only if this validation succeeds.
2. The browser sends a POST request and the OA Framework calls the processFormData methods on all the web beans in the hierarchy as described in Anatomy of an OA Framework Page.
3. Within processFormData, the OA Framework automatically calls setAttribute(String name, Object value) on the current row of the underlying view object for each bean. This executes any *attribute-level* validation that you've written in the view object row.
4. Within this setAttribute method, the view object row automatically calls the corresponding set<*AttributeName*> method in the underlying entity object. This executes any associated *attribute-level* validation in the entity object.
5. Once all the attribute values have been set, the OA Framework calls the view object validate for each row it modified to execute any associated *row-level* validation.
6. Finally, within the validate method, the view object row calls validateEntity for the underlying entity object which executes any associated *entity-level* validation.

Note: The OA Framework automatically displays error messages for any exceptions thrown by the model layer during processFormData and does not proceed to the next phase of calling processFormRequest. See Error Handling for additional information about how the OA Framework displays error messages.

Bypassing Validation

As described above, the OA Framework writes model data for every form submit -- which means that all your attribute and entity level validation is executed. There are times when you need to "short-circuit" this process so errors aren't reported to the user at an inappropriate time.

See Implementing the Controller: Model Interaction for specific instructions on preventing this.

Defaulting

When you create a new row to be used in a "Create" page as shown in Figure 3 above, you can specify default values in three places:

- [Model] If your view object is based on one or more entity objects, you can override their `create()` method to programmatically set attribute-level defaults. See the Java Entity Objects Create topic for additional information.
- [Model] You can also declaratively associate defaults with entity object attributes using the BC4J entity object wizard.
- [View] Alternatively, you can set default values for individual items by setting their `Initial Value` property in the Oracle JDeveloper 10*g* OA Extension. The advantage of this approach -- for static values that can be determined at design time -- is customers can easily personalize these defaults. This feature can be used with items that do, and do not, map to an underlying view object. For example, you could use this for a search criteria field even though it does not have an associated view object instance.

Defaulting is turned off. You must set the value of the profile option FND:OA:Enable Defaults/FND_OA_ENABLE_DEFAULTS to **Y** to turn on defaulting.

If the profile is enabled and a default value is specified in the view (whether in OA Extension or as a personalization) on a form element that **is not a messageChoice or messageRadioGroup**, then OA Framework sets the value of the item according to the following rules:

- If the item has no associated view object data source, the profile option will have no effect and OA Framework automatically sets the default value directly on the item when the page is rendered.

  Note: If you are trying to set the "Initial Value" of a form bean with Data Type as "Date", then use the canonical format yyyy/MM/dd. Otherwise the default value will not apply.

- If the item has an associated view object, OA Framework sets the default value when you call createRow() on your view object.

If the profile is enabled and a default value is specified in the view (whether in the OA Extension or as personalization) on a form element that **is a messageChoice or messageRadioGroup**, then OA Framework sets the value according to the following rules:

- If the value from the current row for the view attribute **is not null**, the default value specified is set as the default value of the selection.
- If the value from the current row for the view attribute **is null**, the default value shown in the selection will be determined by the default value specified and **will be** applied to the view object.
- **Note:** If the profile option is not enabled, and the value from the current row for the view attribute **is null**, the default value shown in the selection will also be determined by the default value specified and **will not be** applied to the view object.

The following example shows typical code for creating a new row:

```
public void createSupplier()
{
  OAViewObject vo = getSuppliersVO();

  // The OA Framework applies UI defaults during the scope of this
  // method call.
  Row row = vo.createRow();
  vo.insertRow();

  // As specified in OA Framework Model Coding Standards,
  // always set the new row state to STATUS_INITIALIZED
  // after you create it.
  row.setNewRowState(Row.STATUS_INITIALIZED);
}
```

Specifically, the `createRow()` method calls `create()` on each `ViewRowImpl`. In the scope of the `create()` call, the OA Framework calls the attribute setter for each UI-specified default it needs to apply. This ensures that any view row attribute validation -- and associated entity object attribute validation -- is executed as described in Figure 3 above. The OA Framework then resets the state of your view row to `STATUS_INITIALIZED` so it appears untouched to BC4J. This ensures that users can navigate to and leave pages having only default values without being warned about the potential loss of work. Any validation exceptions detected during the defaulting process are displayed as normal.

Tip: Defaults are applied *once and only once* when the row is created. If you have a multipage flow with regions that bind to the same underlying view object -- and each region specifies a different `Initial Value` for one of the view attributes -- only the default associated with the first region that renders when the row is created is applied. The others are ignored. Similarly, if you create a new row on Page A, and then navigate to Page B where you have an `Initial Value` set for one of its attributes, the default isn't applied because the row was created before Page B renders (note that creating a row before navigating to the page where the user actually enters values for the new row is not consistent with the recommended Back button implementation for a Create flow).

Assuming defaults are using each of these three approaches, the OA Framework observes the following precedence priority:

1. (Highest) Declarative item property defaults as specified in the OA Extension or in the Personalizations module
2. Programmatic entity object attribute defaults (these are applied during the scope of the `vo.createRow()` method call which in turn delegates to your entity object's `create()` method)
3. (Lowest) Declarative entity object attribute defaults

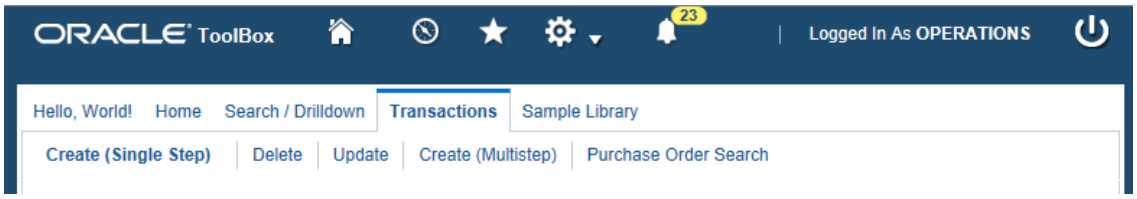
If you want to ensure that a default value is always set regardless of what values might be specified declaratively, you can override the `insertRow()` method in your view object row as shown below:

```
public void EmployeesVOImpl extends OAViewObjectImpl
{

  ...

  public void insertRow(Row row)
  {
    // Always call super.insertRow() first.
    super.insertRow();


    // Now call whatever attribute setters you need to call to ensure
    // that your defaults always take precedence.
    row.setAttribute("<attributeName>", <attributeValue>);
    ...

  }
}
```

Menus and Page Security

In an OA Framework application, the menu of available pages is presented to the user in a tab-based model as illustrated in the following example from the OA Framework ToolBox Tutorial Application:

Figure 4: OA Framework ToolBox Tutorial menus.



Within this basic model, you are free to choose from a range of valid design options based on your application's complexity and expected usage patterns (when you're ready to start designing menus for your application, consult the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Tabs/Navigation first for a detailed introduction to the various menu configurations).

This menu structure serves two distinct purposes in an OA Framework application:

- It organizes content into meaningful units.
- It lets users navigate easily between these meaningful units.

Menu Implementation

Behind the scenes, an OA Framework menu is actually comprised of Oracle E-Business Suite *functions* and *menus*.

Navigation Functions

Navigation functions represent the individual pages within your application; each page that a user accesses at runtime is associated with a predefined function, which is used solely for the purpose of navigating through your application. Perhaps most importantly, this function includes the **Web HTML Call** for the page. For example, in the ToolBox Tutorial Application, when the user selects the **Lesson 3** menu entry, the Purchase Order Search page is displayed. We created a function for this page, and set its **Web HTML Call** to point to the XML page we want to display : OA.jsp?page=/oracle/apps/fnd/framework/toolbox/webui/PoSearchPG
When the user selects the **Lesson 3** menu entry, the request is routed to the OA.jsp which initializes an `oracle.apps.fnd.framework.webui.OAPageBean` object to process the corresponding page XML file as described in Anatomy of an OA Framework page (in case you're wondering, OA.jsp is the only JSP that is invoked when accessing any OA Framework application page).

Note: A single page can be called by many functions (each potentially passing different parameters through the URL), which means it can be used in many different menus.

Navigation Menus

Navigation menus are reusable groupings of functions and submenus that ultimately create the tab structure that we described above. Each OA Framework menu that you create has an associated **Type** that determines how it should be rendered. For example, the Lesson 2 tab in Figure 1 above is of type "HTML Tab."

Navigation menus include ***all*** functions that can display in your application. You can also selectively grant access to individual functions within this navigation menu. This is described more fully below in the Application Security section.

For detailed instructions on designing and creating OA Framework navigation menus, see Chapter 4: Tabs/Navigation.

Application Security

The features of Application security are broad; in this introductory chapter we'll touch on some key concepts so you have a general idea of what is supported as it relates to menu definitions. When you're ready to start designing your application, we recommend familiarizing yourself with these features by reviewing Page Security in Chapter 4.

Users and Responsibilities

An Oracle E-Business Suite reponsibility is a collection of tasks that is granted to one or more users as a set. For example, you might create a **Benefits Manager** and a generic **Employee** responsibility, each with the appropriate HR-related applications. You would then assign these responsibilities to individual users to quickly grant access to these modules.

All responsibilities are associated with the single top-level navigation menu for your application. As described above, the navigation menu ultimately includes all the tabs supported by your application.

Previously, a responsibility was the primary mechanism for grouping users into role-based sets. You would then assign menus to responsibilities, and create security rules by excluding individual menu functions from your responsibility. At runtime, the current responsibility, organization and security group together comprised the security context.

With Release 12, the concept of responsibility has been expanded to a more generic **role**. Users can belong to one or more roles. All users assigned to a particular responsibility are also assigned to a correponding **role**. Security rules are based on **permission grants** instead of function exclusion rules. At runtime, these grants are evaluated for the current security context, which now includes roles (also known as a "grantee") in addition to responsibility, organization and security group.

The OA Framework recommends using permissions roles and grants for your security setup instead of responsibilities and exclusion rules.

Grants and Permissions

In addition to creating Navigation Functions, you must also create Authorization Functions (known as "permissions") for each of your pages. You then group these permissions into a "flat" menu structure (known as a "permission set") for the purpose of granting user access to the associated pages.

The simplest way to introduce the use of permission sets is by walking through a small use case. For example, assume you have a very simple Benefits application including the following four pages:

Page	Description	Benefits Manager Access?	Employee Access?
Administer Benefits	View, update, approve and discontinue benefits.	**Yes**	No
Create Benefit	Create a new benefit.	**Yes**	No
My Benefits	View current benefit selections and make new selections as appropriate.	**Yes**	**Yes**
Update Beneficiaries	Update designated beneficiaries.	**Yes**	**Yes**

As described above, you would create Navigation Functions for each of these pages and organize them into a comprehensive Navigation menu. To ensure that users have access to the right pages, you would then proceed as follows:

Step 1 : Create permissions.

Just like the Navigation functions, **permissions** are FND form functions, but in this context, they are used exclusively for application security.

In our example, we can use the Navigation Functions that we created for each page as permissions. There is no need to create additional permission functions.

Step 2 : Create roles or grantees.

A grantee can either be a **user** (FND_USER)**,** or a **user group**(also known as **role**), or **"global**". **User** identities are created in FND_USERS, and should map one-to-one with individual humans or systems. Users can belong to **groups** or **roles** that are formed by grouping organizational or position relationships modeled in products such as Human Resources. Roles are defined in WF_ROLES. Although its membership is not explicitly populated, there is a **Global** group which includes "everyone".

You need two user roles for the example above: one that groups all managers into a manager role, and another that groups all employees. Since all employees includes everyone, you can use a Global role for this purpose.

Alternately, you can create a responsibility that is assigned to all managers, and use that for your grants setup.

We will discuss both the above alternatives when we proceed to Step 4 to create the grants.

Step 3: Create permission sets.

Permission Sets are implemented as menus, but they are exist solely to group a flat list of permissions into sets for easy access granting. Ideally, you should group permissions that are required by a role into one or more permission sets.

You need two permission sets for the example above:

- A Manager Permission Set for all the tasks to which only managers should have access. This includes the navigation functions "Administer Benefits", and "Create Benefit".
- A Global permission set with permissions that are accessible by everyone. This includes the navigation functions "My Benefits" and "Update Beneficiaries".

Step 4: Create Grants

A **Grant** defines security rules that allows only certain users of your system access to specific functions or pages within your application. A **grant** gives a **grantee** access to the permission sets described above. In simple terms, grants link your grantees to your permission sets.

You need two grants for the example above:

- A Manager Grant to associate the manager permission set with the manager role.
- An Employee Grant that is associated with your Global permission set with a global grantee.

  Since this grant is associated with a global grantee (in other words, everyone) and has no additional security restrictions (in other words it is not restricted to any responsibility, organization or security group), it can also be called a **global grant**.

In addition to specifying a grantee, you could also restict your grant further with additional security context. This includes the current user's responsibility, organization and security group. So, for example, to restrict the manager grant to a specific organization, you can associate an organization context with the grant.

Instead of granting the manager permission set to the manager role, you can grant it to a global grantee. You can then restrict it to managers alone by associating a security context with the responsibility to which only managers have access. However note that the OA Framework recommends the use of role based grants instead of responsibilities.

At runtime, a user is granted access to a page if the permission associated with the page is granted access to the current user's security context. The user's security context as described above includes the user's role, responsibility, organization and security group.

Page Security

If you look at the example above, we mention that you can link the permissions with your pages to restrict access. This is one of the cases where you need to secure the rendering of your page with a permission.

Other cases where you may want to secure the rendering of your page with a permission include anonymous login pages, pages that require auto responsibility setting or switching, and shared/reusable pages.

For detailed instructions please look at Chapter 4: Page Security.

Implementing the Controller

Overview

This document describes how to implement an OA Controller in generic terms. It does not describe how to implement specific UI features. For this information, see individual topics in Chapter 4: Implementing Specific UI Features.

Contents

- Designing an OA Controller
- Creating an OA Controller
- Handling an HTTP GET
  - Modifying Bean Properties
  - Creating Beans Programmatically
- Handling an HTTP POST (Form Submit)
- Model Interaction
- Disabling Validation
- Error Handling
- Javascript

Prerequisite Reading

This document assumes that you have read the following in the OA Framework Developer Guide:

- Building "Hello, World!"
- JSP Application Primer
- Anatomy of an OA Framework Page
- OA Framework State Management
- Implementing the Model
- Implementing the View

Designing an OA Controller

As described in Anatomy of an OA Framework Page, the OA Controller is where you define how web beans behave. Specifically, you write controller code to:

- Manipulate/initialize the UI at runtime (including any programmatic layout that you are unable to do declaratively)
- Intercept and handle user events like a button press

Controllers should never include any kind of business logic; this belongs in your model classes.

Necessity

In general, before tackling the question of *how* to design your controller, it's important to consider whether you even need to create a controller.

As a rule, you should write controller code only if it is absolutely essential. If you can create your page declaratively, do not instantiate regions and items programmatically. Programmatically created web beans cannot be personalized, reused or extended. Furthermore, some hard-coded layouts may fall out of compliance with the Oracle Browser Look-and-Feel (BLAF) UI Guidelines as they evolve over time.

As described in Implementing the View, all top-level regions in a shared component must have an associated controller.

Granularity

OA controllers can be associated with any region (any web bean that implements `oracle.apps.fnd.framework.webui.beans.OAWebBeanContainer`); you cannot associate controllers with items.

Many new OA Framework developers wonder just "how big" a controller should be. Should you have one per page, one per meaningful region (like a "Search" region), one per complex web bean (like a table) -- or what? Unfortunately, the answer is *it depends*.

First and foremost, in a really simple page, you might not have any controllers (there is no requirement that you create controllers if they have no work to do). If you do need to write code, you should weigh the following carefully before deciding what controllers to create:

- the benefits of encapsulation -- ideally, a web bean implements its own behavior
- component reuse -- if a component is designed to be shared, it must be self-sufficient
- practical code usability -- although a page comprised of eight regions could easily have eight corresponding controllers (each with a few, trivial lines of code), this "pure" object oriented approach can make code maintenance more difficult, and it can cause unnecessary file bloat in your product

With that in mind, there are a few guidelines that might help the decision-making process:

- Never set properties on a parent/grandparent web bean from a child bean.
- Always define controllers so they control the regions with which they're associated, or set properties on children/grandchildren of this region. If you want a controller to manage multiple child/grandchild web beans, it should be associated with an appropriate parent/grandparent bean.
- For the really complex beans (like an OATableBean) you should associate a controller with the bean itself, or perhaps with a simple containing bean if it represents a logical unit of functionality.

In general, you should create the fewest number of controllers per page that satisfies the rules and considerations outlined above. For a very simple page, it is very common to associate a single controller with the **pageLayout** region. For a more complicated page, you might create a few different controllers for meaningful functional components (for example, a searchable summary page typically has a "Search" region controller and a "Results" region controller). Shared regions should obviously have their own controller(s) as appropriate.

Modularity/Reuse

Within any group of related pages, you will typically find opportunities to reuse code. The following are all valid approaches to creating more modular controller code:

- You can add your own private methods to your controllers.

- You can create a common controller that subclasses `oracle.apps.fnd.framework.webui.OAControllerImpl`, and then subclass this as needed for individual pages/regions.
- You can create helper utility classes to which your controller code can delegate as needed. These classes are not required to subclass/implement any OA Framework classes/interfaces, and should be included in the same package(s) as the controller(s) they help. Note that static methods are a natural fit in these (often procedural) classes, and while it is appropriate to include static methods, you should consider the following:
  - You (and more importantly, the customer) can't effectively subclass static methods.
  - There are packaging implications related to the use of constants and static methods (see the Oracle E-Business Suite Java Coding Standards).

Thread-Safety

The OA Framework is designed to support multithreaded web bean access (although this is not yet implemented). Most of this is transparent to you, however, there are a few rules that you must follow in your controller code:

- If you use static methods in your controllers or helper classes, *never* include state.
- Always pass the page's `OAPageContext` to any web bean accessors (if there is a signature that takes an `OAPageContext`). For example, choose setText(OAPageContext pageContext, String text) instead of setText(String text).

State Management

Never add non-transient member variables to your controllers, or to helper classes if you instantiate them. You may add `static final` member variables.

Coding Standards Compliance

Before writing any controller code, you should read the following documents carefully. While this topic mentions several key controller standards, it is not intended to be a comprehensive checklist. For any OA Framework code that you write, the documents in Chapter 8 should be considered the "single source of truth" for coding standards.

- Chapter 8: Oracle E-Business Suite Java Coding Standards
- Chapter 8: OA Framework File Standards (Naming, Package Structure and Standard Content)
- Chapter 8: OA Framework Controller Coding Standards

Creating an OA Controller

To create a controller for a region:

1. Select the region in the JDeveloper Structure pane
2. Use the right mouse button to select **Set New Controller...**
3. In the **New Controller** dialog, enter the package and class names in accordance with the OA Framework File Standards. Select **OK** to create the controller and associate it with the selected region. Note that the **Controller Class** value in the property inspector is a fully qualified class name: oracle.apps.fnd.framework.toolbox.tutorial.webui.HomeSearchCO.

JDeveloper creates a template controller for you with the following content.

```
/*===========================================================================+
 | Copyright (c) 2001, 2003 Oracle Corporation, Redwood Shores, CA, USA      |
 | All rights reserved.                                                      |
 +===========================================================================+
 | HISTORY                                                                   |
 +===========================================================================*/
package oracle.apps.fnd.framework.toolbox.tutorial.webui;

import oracle.apps.fnd.common.VersionInfo;
import oracle.apps.fnd.framework.webui.OAControllerImpl;
import oracle.apps.fnd.framework.webui.OAPageContext;
import oracle.apps.fnd.framework.webui.beans.OAWebBean;

/**
  * Controller for ...
  */
public class OrderSummaryCO extends OAControllerImpl
{
   public static final String RCS_ID="$Header$";
   public static final boolean RCS_ID_RECORDED =
   VersionInfo.recordClassVersion(RCS_ID, "%packagename%");

 /**
   * Layout and page setup logic for a region.
   * @param pageContext the current OA page context
   * @param webBean the web bean corresponding to the region
   */
   public void processRequest(OAPageContext pageContext, OAWebBean webBean)
   {
     super.processRequest(pageContext, webBean);
   }

 /**
   * Procedure to handle form submissions for form elements in
   * a region.
   * @param pageContext the current OA page context
   * @param webBean the web bean corresponding to the region
   */
   public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
   {
     super.processFormRequest(pageContext, webBean);
   }

}
```

Note: The default template does not include the processFormData(OAPageContext pageContext, OAWebBean webBean) method that is called during the first

phase of POST processing. If you find that you need to use this method (a fairly uncommon scenario), you can simply add it to your controller.

To copy a controller:

1. In JDeveloper, open the controller that you want to copy.
2. Select **File > Save As...** from the main menu.
3. In the **Save As** dialog, be sure to specify the right **Location** (Java package) and enter the **File name** (class name).
4. In the new controller file, remember to edit the package declaration (if necessary) and change the class name.

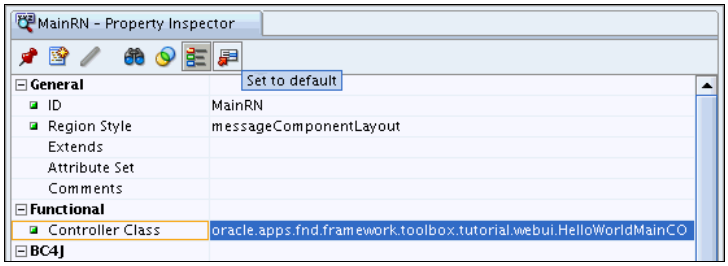
To associate a preexisting controller with a region:

1. Select the region in the JDeveloper Structure pane.
2. Place your cursor in the Property Inspector's **Controller Class** field and select the **...** button.
3. In the **Controller Class** dialog, expand the package hierarchies until you find the controller that you want to select. Select **OK** to make your choice.

To disassociate a controller from a region:

1. Select the region in the JDeveloper Structure pane.
2. Place your cursor in the Property Inspector's **Controller Class** field.
3. Select the **Set to Default** button in the Property Inspector's Toolbar (it is not sufficient to manually clear the value from the field) as shown in Figure 1 below.

Figure 1: Highlighted **Set to Default** button in the OA Extension Property Inspector toolbar



Note: You can also associate controllers with regions programmatically. See the setControllerClass(String javaClass) method in `OAWebBeanContainer`.

Handling an HTTP GET

During GET processing, each controller's processRequest(OAPageContext pageContext, OAWebBean webBean) method is called as the web bean hierarchy is instantiated. Processing begins with the **pageLayout** bean, and proceeds recursively throughout the entire hierarchy. Code that initializes your page -- or affects the web bean hierarchy in any way (by setting properties, creating web beans and so on) -- belongs in the processRequest() method.

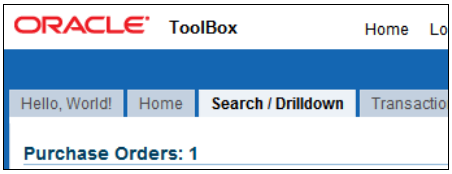
Note: The `oracle.apps.fnd.framework.webui.OAWebBean` parameter passed to the processRequest`()` method is the region with which the controller is associated.

The following example is typical of the processRequest() code that you will write. It illustrates the initialization of a view-only "detail" page based on a request parameter (for a selected purchase order) passed from a "search" page.

```
/**
 * Layout and page setup logic for region.
 * @param pageContext the current OA page context
 * @param webBean the web bean corresponding to the region
 */
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   // Always call this before adding your own code.
   super.processRequest(pageContext, webBean);

   // Get the purchase order number from the request.
   String orderNumber = pageContext.getParameter("headerId");

   // We need to set the page header text to include the PO order number for reference.
   MessageToken[] tokens = { new MessageToken("PO_NUMBER", orderNumber) };

   // Always use a translated value from Message Dictionary when setting strings in
   // your controllers.
   String pageHeaderText = pageContext.getMessage("ICX", "FWK_TBX_T_PO_HEADER_TEXT", tokens);

   // Set the po-specific page title (which also appears in the breadcrumbs. Since this
   // controller is associated with the page layout region, simply cast the webBean
   // parameter to the right type and set the title.

   ((OAPageLayoutBean)webBean).setTitle(pageHeaderText);

   // Now we want to initialize the query for our single purchase order with all of its
   // details.
   OAApplicationModule am = pageContext.getApplicationModule(webBean);
   Serializable[] parameters = { orderNumber };
   am.invokeMethod("initDetails", parameters);

} // end processRequest()
```

After calling super.processRequest(pageContext, webBean), the example code gets the value for a request parameter named "headerId" (the purchase order number the search page passed on the request). This value is then displayed in the page title and breadcrumbs as context for the user, and passed to the model so the purchase order can be queried.

Figure 2: Example of a dynamically defined page title



Since all values displayed in the page must be translatable, we created a message named **FWK_TBX_T_PO_HEADER_TEXT** in the Oracle E-Business Suite Message Dictionary with the text **"Purchase Order: &PO_NUMBER"**. The code defines the purchase order number as the replacement value for the token **PO_NUMBER**, and then obtains a translated version of this message from the `oracle.apps.fnd.framework.webui.OAPageContext` (which delegates to AOL/J). It then sets the translated String as the page's title.

Warning: Never display a hard-coded text value in the user interface. All text values that you display programmatically must be sourced from Message Dictionary as shown. You can also use a value from a web bean that was set declaratively (all displayable bean properties are translated), or you can display a value queried from a multilanguage product table.

Finally, this read-only "details" page automatically queries the given purchase order whenever it is rendered. It does this by passing the purchase order number to a method called initDetails() in the page's root application module. The application module then passes this parameter to the appropriate view object, which binds the WHERE clause parameter and executes its query. The Model Interaction section below describes this in greater detail.

Modifying Bean Properties

Note: As a rule, it is preferable to modify web bean properties using partial page rendering (PPR) and SPEL bindings as described in Dynamic User Interface. The instructions in this section assume you cannot leverage PPR and SPEL for your requirement, and therefore must make changes to the web bean hierarchy in `processRequest()` (this section is included in the `GET` handling because you are allowed to modify the web bean hierarchy ONLY in your processRequest() method).

If you need to programmatically modify the hierarchy in response to a form submit event (for example, the user presses a button), you must forward back to the same page so your processRequest() code can be executed (see the example in the `POST` event handling section below). Reasons for this restriction (which you should not try to "work around") include:

- Ensures that the web bean hierarchy can be correctly recreated if necessary.
- Beans are initialized properly. This is primarily an issue with the `Rendered` property, or complex components affected by the `prepareForRendering()` method.
- Bean hierarchy maintanence is encapsulated in a single method.

To modify a web bean's properties, you simply need to find the correct bean in the hierarchy using its name (the `ID` you assigned in JDeveloper), and call the appropriate method as shown in the following example.

Warning: When you get a handle to a web bean, always check whether the value is null before calling any of its methods. Even if you expect the bean to be included in the web bean hierarchy, it's possible that a customer may hide it with a personalization.

```
processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this before adding your own code.
  super.processRequest(pageContext, webBean);

  OATableBean table = (OATableBean)webBean.findIndexedChildRecursive("OrdersTable");

  if (table == null)
  {
     MessageToken[] tokens = { new MessageToken("OBJECT_NAME", "OrdersTable")};
     throw new OAException("ICX", "FWK_TBX_OBJECT_NOT_FOUND", tokens);
  }

  // Set the purchase-order specific "control bar" select text:
  // "Select Purchase Order(s) and..."

  String selectPOText = pageContext.getMessage("ICX", "FWK_TBX_T_SELECT_PO", null);
  table.setTableSelectionText(selectPOText);

}
```

Starting with the controller region's children, the findIndexedChildRecursive(String name)method searches the entire web bean hierarchy looking for the first indexed child with a matching name. If the web bean that you want to modify is a UIX named child (or, if you're not sure whether it is "named" or "indexed"), use the findChildRecursive(String name) method instead.

If you need to modify properties on the controller region, simply cast the processRequest() `OAWebBean` parameter to the right type and call whatever method you need (see the GET code example above for an illustration of this).

Creating Beans Programmatically

Note: This section is included in the GET handling because you are allowed to modify the web bean hierarchy ONLY in your processRequest() method.

If you need to add beans to the web bean hierarchy in response to a form submit event (for example, the user presses a submit button), you must forward back to the same page so your processRequest() code can be executed. See the example in the POST event handling section below.

As a rule, you should NOT create web beans programmatically if you can create them declaratively for all the reasons described above. Furthermore, if your page leverages partial page rendering, your web bean hierarchy cannot be changed at runtime.

For those rare cases when you must instantiate a web bean yourself, use the `createWebBean()` factory methods in the `OAControllerImpl` class. Do not use any of the web bean constructors directly, and do not worry about creating an `oracle.apps.fnd.framework.webui.OAWebBeanFactory` directly since the controller `createWebBean()` methods delegate to the `OAWebBeanFactory` behind the scenes.

Note: For beans created "from scratch" (meaning there is no declarative definition for the bean), use the factory method that allows you to specify the bean's "name" (the `ID` property in JDeveloper). Avoid the deprecated methods that allow you to create a web bean without specifying its name. The web bean's name uniquely

identifies it on a page (each name must be unique within a page), and it is essential for a variety of reasons described in the OA Framework Controller Coding Standards. Furthermore, a bean's name may be used by the OA Framework in a BC4J object instance name (such as an application module instance), and therefore should not include invalid characters for Java names.

For example, the following code illustrates how to create two web beans from scratch and add them to a parent region.

```
OATableLayoutBean tableLayout = (OATableLayoutBean)findIndexedChildRecursive("tableLayout");

// Create a row layout and give it the unique ID "topRow"
OARowLayoutBean row = (OARowLayoutBean)createWebBean(pageContext,
                                                     OAWebBeanConstants.ROW_LAYOUT_BEAN,
                                                     null, // no need to specify a data type
                                                     "topRow");

// Create a row layout and give it the unique ID "bottomRow"
OARowLayoutBean anotherRow = (OARowLayoutBean)createWebBean(pageContext,
                                                            OAWebBeanConstants.ROW_LAYOUT_BEAN,
                                                            null, // no need to specify a data type
                                                            "bottomRow");

// Always check to see if a web bean exists.
if (tableLayout != null)
{

   // Add the two row layout beans to the table so the "topRow" renders above
   // the "bottomRow"
   tableLayout.addIndexedChild(row);
   tableLayout.addIndexedChild(anotherRow);
}
```

You can also instantiate web beans that have been defined declaratively, but require a programmatic association with the parent. For example, in the following code, a **stackLayout** region named "HomeSearchRN" was defined in JDeveloper, but it must be added to the programmatically created side navigation component.

```
OASideNavBean sideNav = (OASideNavBean)createWebBean(pageContext,
                                                     OAWebBeanConstants.SIDE_NAV_BEAN,
                                                     null, // no need to specify a data type
                                                     "sideNav" // always specify name);

OAStackLayoutBean search =
   (OAStackLayoutBean)createWebBean(pageContext,
                                    "/oracle/apps/fnd/framework/toolbox/tutorial/webui/HomeSearchRN",
                                    "HomeSearchRN", // always specify name
                                     true); // region created in Oracle JDeveloper OA Extension

sideNav.addIndexedChild(search);
```

Restrictions

The OA Framework does not readily support the programmatic addition, removal or replacement of children to any of the "default" regions (for example, an OA Extension **defaultSingleColumn** region which is instantiated as an `oracle.apps.fnd.framework.webui.beans.layout.OADefaultSingleColumnBean`). These regions should be defined declaratively. If you absolutely must replace or remove items in a "default" region (you cannot add items), follow these steps:

1. Find the child web bean that you want to remove or replace by calling `webBean.findIndexedChildRecursive()`.
2. Get the child's parent web bean by calling `childWebBean.getAttribute(OAWebBeanConstants.PARENT)`.

   Note: The `OAWebBeanConstants.PARENT` attribute is intended exclusively for OA Framework internal development use (if you look at the `OAWebBeanConstants` Javadoc, you'll see a warning to this effect). You may leverage this approach only for default regions due to their unique implementation; as a rule, the OA Framework Controller Coding Standards discourage modifying parent web beans from child controllers. Furthermore, the default regions have been deprecated so you should not be using them for new development.

3. Then, perform the replace or remove on the parent bean itself.

Handling an HTTP POST (Form Submit)

During HTTP `POST` processing, the OA Framework first checks to see if the page's web bean hierarchy is available in its cache. If not, because resources are constrained or the user navigates with the browser Back button, the OA Framework must recreate the web bean hierarchy before proceeding. This means that all of your processRequest() code is re-executed as if the browser had issued an HTTP `GET` request.

Note: The potential recreation of the web bean hierarchy yields a number of coding considerations which are fully described in Chapter 6: Supporting the Browser Back Button and the corresponding OA Framework View and Controller coding standards.

The primary `POST` processing occurs in two separate passes over the web bean hierarchy:

- First, OA Framework writes form data to the model by calling processFormData() on each web bean recursively until the entire hierarchy is traversed. Any code that needs to be executed during this processing phase should be added in your controller's processFormData(OAPageContext pageContext, OAWebBean webBean) method.
- Assuming no exceptions were thrown during the first processing phase, OA Framework proceeds to the second phase which involves calling processFormRequest(OAPageContext pageContext, OAWebBean webBean)on each web bean.

processFormData( )

In most -- if not all -- of the pages that you build, you will have no cause to overwrite this method. In fact, the only use case we could think of is extremely unlikely in an OA Framework application: if the data source for a region is not a view object, so the view instance and attribute properties are not defined for the individual web beans, then you could code the region's processFormData() to write the child web bean data to the appropriate data source.

Note: The OA Framework implements processFormData() at the item level, but you can overwrite it only at the region level, so you must process all of the region's items if you ever implement anything like this. If you do choose to implement something like this, remember to call `super.processFormData(OAPageContext`

pageContext, OAWebBean webBean) first.

processFormRequest( )

Any code that handles user form submit actions belongs in the processFormRequest() method.

The following example is typical of the processFormRequest() code that you will write. It illustrates how to determine that a particular form submit component was selected (in this case, a "Go" button), how to initiate a query in the model code, and how to perform a JSP Forward back to the same page so web bean properties can be changed in the processRequest() method.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
   // Always call this before adding your code
   super.processFormRequest(pageContext, webBean);

   // Pressing the Go button causes the search to be executed.
   If (pageContext.getParameter("Go") != null)
   {
     String orderNumber = pageContext.getParameter("SearchOrder");
     String created = pageContext.getParameter("Created");
     String showMyOrders = pageContext.getParameter("MyOrders");

     OAApplicationModule am = pageContext.getApplicationModule(webBean);

     // All parameters passed using invokeMethod() must be serializable.

     Serializable[] parameters = { orderNumber, created, showMyOrders };
     am.invokeMethod("search", parameters);

     // Now forward back to this page so we can implement UI changes as a
     // consequence of the query in processRequest(). NEVER make UI changes in
     // processFormRequest().

     pageContext.setForwardURLToCurrentPage(null, // no parameters to pass
                                            true, // retain the AM
                                            OAWebBeanConstants.ADD_BREAD_CRUMB_NO,
                                            OAWebBeanConstants.IGNORE_MESSAGES);
  }
} // end processFormRequest();
```

This example shows how to pass request parameters using the setForwardUrl() method, including how to replace a pre-existing parameter value (in this case, with "X" which would be used as an "ignore" value in the target page).

```
import com.sun.java.util.collections.HashMap;
import oracle.bali.share.util.IntegerUtils;

...

processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
   // Always call this before adding your code
   super.processFormRequest(pageContext, webBean);

   String poEvent = pageContext.getParameter("poEvent");

   HashMap params = new HashMap(2);

   // Replace the current poEvent request parameter value with "X"
   params.put("poEvent", "X");

   // IntegerUtils is a handy utility
   params.put("poStep", IntegerUtils.getInteger(5));

   pageContext.setForwardURL("OA.jsp?page=/oracle/apps/dem/employee/webui/EmpDetailsPG", // target page
                             null, // not necessary with KEEP_MENU_CONTEXT
                             OAWebBeanConstants.KEEP_MENU_CONTEXT, // no change to menu context
                             null, // No need to specify since we're keeping menu context
                             params, // request parameters
                             true, // retain the root application module
                             OAWebBeanConstants.ADD_BREAD_CRUMB_YES, // display breadcrumbs
                             OAException.ERROR); // do not forward w/ errors
}
```

Note: If a view object is used as a data source for a web bean that is to be displayed, do not remove the view object, its rows, or the containing nested application module. If you want to remove any of these objects before redirecting to a page that will no longer show the view object data (for performance optimization reasons), then after making the remove calls, be sure to redirect to the new page using oracle.apps.fnd.framework.webui.OAPageContext.forwardImmediatelyOAPageContext.setforwardURL). This ensures that the forward action takes place immediately and no further web bean processing is done on the current page after the forward call; otherwise, the removed view object or row instances may cause side effects in subsequent OA Framework web bean processing routines.

Posting to an OA Framework Page from a Different Technology

If you post to an OA Framework page from a different technology (a JTT page, for example) the OA Framework executes only the processRequest phase in the target page. It does not execute the processFormData and processFormRequest phase. It is important that you do not work around this behavior; please design your application to accommodate this expected behavior.

Model Interaction

In simple terms, *the only model object that you should access directly from your OA controller is the application module*. In other words, the only valid model import

in your controller code is:

import oracle.apps.fnd.framework.OAApplicationModule;

You should **not** access view objects directly to execute queries, iterate the rowset, or interact with the underlying entity object(s). For example, the following code (although technically feasible) is **incorrect** according to the OA Framework Controller Coding Standards.

```
import oracle.apps.fnd.framework.OAViewObject;
   ...

   // Get the root application module
   OAApplicationModule am = pageContext.getRootApplicationModule();

   // Find the view object you want to query
   OAViewObject vo = (OAViewObject)am.findViewObject("<instanceName>");

   ...
```

Instead, if you need to execute a view object query, you should proceed as shown in the following example which illustrates handling a "Go" button press in a "Search" region.

First, add a method to the containing application module (in this example, it's the page's root application module) which accepts search criteria and then delegates to the target view object for query execution (see Implementing the Model for information about query execution).

```
public void search(String orderNumber, String created, String showMyOrders)
{
  PoSummarySimpleExpVOImpl vo = getPoSummarySimpleExpVO();

  // Always check for the null condition if the VO cannot be found/created
  if (vo == null)
  {
    MessageToken[] tokens = { new MessageToken("OBJECT_NAME", "PoSummarySimpleExpVO")};
    throw new OAException("ICX", "FWK_TBX_OBJECT_NOT_FOUND", tokens);
  }

  vo.initQuery(orderNumber, created, showMyOrders);

} // end search()
```

Then, add button handler code like the following to your controller which invokes the correct method in the application module.

Note that you should always check for an event source in your `processFormRequest()` code; never assume that the browser issued a `POST` request because your item was selected (even in a simple page with just one button, for example). Behind the scenes, the OA Framework often submits for the page form when you might not be expecting it to do this.

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{

  // Check to see if the "Go" button was pressed...
  if (pageContext.getParameter("gButton") != null)
  {
    // Get the search criteria
    String orderNumber = pageContext.getParameter("SearchOrder");
    String created = pageContext.getParameter("Created");
    String showMyOrders = pageContext.getParameter("MyOrders");

    OAApplicationModule am = pageContext.getApplicationModule(webBean);

    // All parameters passed using invokeMethod() must be serializable.
    Serializable[] parameters = { orderNumber, created, showMyOrders };
    am.invokeMethod("search", parameters);
  }
}
```

Tip: Whenever you call `invokeMethod()` on a server-side BC4J component, any parameters that you pass must be `Serializable`. The example above illustrates the `invokeMethod()` signature that expects all the parameters to be Strings. If you need to pass other object types, use the version of `invokeMethod()` that takes an array of parameter types. For example:

```
Class[] parameterTypes = { String.class, Hashtable.class, Number.class ...};
am.invokeMethod("search", parameters, parameterTypes);
```

Similarly, since the view object is the conduit to the entity object -- and you should not interact directly with view objects in your controllers -- it stands to reason that you should also route all entity object actions through an application module also.

Note: As described in Implementing the Model, the methods that you add in your application module should be named for the corresponding UI "events." For example, if the user presses a "Create" button, the application module method should be named "create" and so on.

Create Example

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  OAApplicationModule am = pageContext.getApplicationModule(webBean);
  am.invokeMethod("create", null);
}
```

Delete Example

This example illustrates invoking a delete method on a nested application module associated with a shared region as opposed to the page's root application module.

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
```

```
{

 if (pageContext.getParameter("DeleteYesButton") != null)
 {
   // User has confirmed that she wants to delete this purchase order.
   // Invoke a method on the AM to set the current row in the VO and
   // call remove() on this row.

   String poHeaderId = pageContext.getParameter("poHeaderId");
   Serializable[] parameters = { poHeaderId };

   OAApplicationModule am = pageContext.getApplicationModule(webBean);
   am.invokeMethod("delete", parameters);
  }

 ...
```

Custom Action Example ("Approve")

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  if (pageContext.getParameter("Approve") != null)
  {
    OAApplicationModule am = pageContext.getApplicationModule(webBean);
    am.invokeMethod("approve");
  }
}
```

Commit Example

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
   // Simply telling the transaction to commit will cause all the Entity Object validation
   // to fire.
   //
   // Note: there's no reason for a developer to perform a rollback. This is handled by
   // the OA Framework if errors are encountered during the processFormData phase.

   OAApplicationModule am = pageContext.getApplicationModule(webBean);
   am.invokeMethod("apply");
}
```

Disabling Validation

There are times when you need to bypass the normal validation that occurs during the OA Framework HTTP `POST` processing. For example, if you are implementing an "Add Another Row" button in a table, you may not want error messages to be displayed for incomplete rows when the user opts to add a new row. Similarly, you might want to defer validation in a multistep page flow until the final review and submit page, or while navigating through sub tabs presenting different views of the same underlying object.

Disabling Server-Side Validation

To prevent exceptions from being thrown by your model validation logic, call the setServerUnvalidated(true) method on any of the following beans as appropriate for your page (remember to add this web bean modification code in a processRequest() method!):

- OASubmitButtonBean
- OATableBean
- OAAdvancedTableBean
- OASubTabLayoutBean
- OANavigationBarBean
- OADefaultHideShowBean
- OAHideShowHeaderBean

Note: You can also set this declaratively by setting the component's `Disable Server Side Validation` property to **True**, and you can disable this validation for links or icon that have been configured to perform a form submit. See the Javascript URL section below for additional information.

When the user performs an action on one of these beans that causes the form to be submitted, the OA Framework proceeds through all the HTTP `POST` processing described above -- including executing all your attribute-level validation logic (entity-level validation is not performed). If `oracle.apps.fnd.framework.OARowValException` or `oracle.apps.fnd.framework.OAAttrValException` exceptions (or their deprecated suprclasses) are thrown during processFormData(), the OA Framework simply ignores them and continues processing as if it had encountered no exceptions.

Note: The OA Framework does *not* ignore serious exceptions (like a `NullPointerException`) thrown in processFormData(). These are displayed as usual, and processing does not proceed to processFormRequest(). Furthermore, any exceptions that you or BC4J throw in processFormRequest`()` are displayed as usual.

Disabling Client-Side Validation

Whenever a form with user-entered data submits, UIX performs some basic `onSubmit` Javascript validation (it verifies required fields, data types and formats), and submits the form only if the validation succeeds. To complete the process of bypassing validation, you also need to disable these client-side checks by calling setUnvalidated(true)for the same beans listed in the "Disabling Server-Side Validation" section above.

Note: You can also set this declaratively by setting the component's `Disable Client Side Validation` property to **True**, and you can disable this validation for links or icon that have been configured to perform a form submit. See the Javascript URL section below for additional information.

Tip: For tables and HGrid components, you must enable/disable client-side validation by setting a property for the table and HGrid regions themselves since you do not have direct access to the `OANavigationBarBean` child web beans used for data set navigation. Note that you currently cannot disable server-side validation for these components.

Error Handling

The OA Framework automatically displays any error messages thrown in the model layer; you don't need to do anything in your controller to facilitate this.

- See Error Handling for information about throwing exceptions in your controller code and displaying Error, Warning, Confirmation and Information messages at the top of a page.
- See Chapter 4: Dialog Pages for information about displaying a model Error, Warning, Confirmation, and Information dialog page.

Javascript

UIX and the OA Framework are rapidly adding new features to provide a more interactive user experience (partial page rendering, automatic table totaling, and so on). You are certainly encouraged to leverage these features as they are released, however, you should not attempt to implement them yourself before they're ready.

As a customer, you may add your own Javascript code to OA Framework applications through controller customizations using setter methods for Javascript events that are exposed on form web beans such as check boxes, text inputs, and so on. However, as with any customization, Oracle does not support or entertain bugs on such custom Javascript usage. As a customer, it is your responsibility to certify your custom Javascript code, ensuring that it does not break the functionality of the page and is compliant with usability, security and browser standards.

Javascript URL

Previously, if you wanted to configure a link or image to submit the page form (because you need an event to handle before navigating to a new page), you could set its destination property to the UIX `submitForm` Javascript function

Now, you should configure a **fireAction** event instead of using the Javascript URL. See the Declarative Submit Form documentation for additional information.

Error Handling

Overview

This document describes how to throw OA Framework exceptions in your model and controller code.

Contents

- Exception Types
- Exception Classes
- Bundled Exceptions
- Exception Examples
- Dialog Pages and Message Boxes

Prerequisite Reading

- Implementing the Model
- Implementing the View
- Implementing the Controller

Related Information

- Implementing Entity Objects
- Implementing PL/SQL Entity Objects

Exception Types

OA Framework handles three basic types of exceptions: general, validation and severe. These types are briefly described in this section; specific exception usage instructions are provided below.

General Exceptions

Errors in the BC4J framework are handled by throwing an implicit (runtime) exception of the type `oracle.jbo.JBOException`. OA Framework has its own specialization of this called `oracle.apps.fnd.framework.OAException` . This specialization provides a mechanism for bundling multiple exceptions together, while also translating the exception messages using Oracle E-Business Suite Message Dictionary, so that useful messages can be displayed. In any of your code, you can throw an `OAException` for general, page-level exceptions.

Validation Exceptions

Validation exceptions are thrown from entity objects and view objects for both row and attribute level validation failures.

- `oracle.apps.fnd.framework.OAAttrValException` - specialization of `OAException` used for attribute level validation failures
- `oracle.apps.fnd.framework.OARowValException` - specialization of `OAException` used for row (entity) level validation failures

OA Framework displays error messages to the user as follows:

- Attribute-level exceptions are visually indicated on the error item(s) and at the top of the page
- Row-level exceptions are visually indicated on the error row(s) and at the top of the page
- Page-level exceptions are visually indicated at the top of the page

Severe Exceptions

Severe (or "fatal") exceptions include unexpected system-level errors (like a `NullPointerException`) and selected JBOExceptions like NoDefException. You can also deliberately throw a severe exception in your code.

If a fatal exception occurs, the user is directed to the OAErrorPage (f the fatal exception occurs in the middle of page rendering, the page is partially rendered with a user-friendly error message that includes a link to the detail stack trace). The OAErrorPage also displays a user-friendly error message with a link to the detail stack trace.

Note: This is an untranslated message that customers can change on site.

Oracle Workflow Notification

OA Framework also ships a seeded business event (oracle.apps.fnd.framework.OAFatalError)which sends a notification to the SYSADMIN user whenever OA Framework page reports a severe or fatal exception. The notification includes the detailed error stack for the fatal exception and information about the user who encountered the exception. If you wish to change the notification's default recipient from SYSADMIN, you need to customize the definition of Item Type OAERROR.

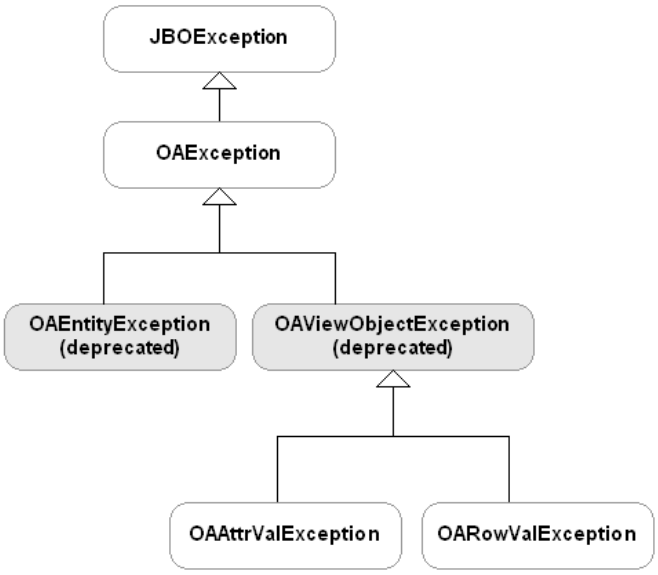
The subscription to this business event is disabled by default. To enable the subscription, refer to the Oracle Workflow documentation on how to enable a subscription for a business event.

- If you are using Oracle Workflow Release 11i/2.6, with the original Oracle Applications user interface, see "To Update or Delete an Event Subscription", *Oracle Workflow Developer's Guide*.
- If you are using OA Framework-based user interface for the Business Event System, see "To View and Maintain Event Subscriptions", *Oracle Workflow Developer's Guide*.

Exception Classes

OA Framework exception inheritance hierarchy is shown in Figure 1 below. The OAException superclass extends JBOException. OAAttrValException and OARowValException extend OAViewObjectException, a deprecated class that extends OAException.

Figure 1: OA Framework exception inheritance hierarchy



OAException

`OAException` is the exception that you would throw in generic error cases (for example, if you encountered an unexpected problem in some controller code as shown):

```
OACellFormatBean shipTermsCell =
   (OACellFormatBean)webBean.findIndexedChildRecursive("ShipTermsCell");

 if (shipTermsCell == null)
 {
   MessageToken[] tokens = { new MessageToken("OBJECT_NAME", "ShipTermsCell")};
   throw new OAException("AK", "FWK_TBX_OBJECT_NOT_FOUND", tokens);
 }
```

Note that we created a message in the Oracle E-Business Suite Message Dictionary (FWK_TBX_OBJECT_NOT_FOUND) in the AK application. The message was defined with a token (OBJECT_NAME) that we replace with the name of the UI component that we expected to find. OA Framework will use this information to automatically display a properly translated error message at the top of the page (if you instantiate an OAException, and don't specify the message type, it always renders as an error).

Note: Although it is not an explicit coding standard, it is good coding practice and beneficial to wrap an exception rather than create a new exception:

```
throw OAException.wrapperException(myException);
```

Wrapping an exception creates a new OAException that includes the original exception as a detail exception. By wrapping an exception, the core exception gets reported and OA Framework inspects the original exception to determine its severity. If you create a new exception, you do not get that information, and the exception stack trace stops with your code, resulting in a bug with less information.

Since the OAException is a flexible object that can also be used to display other types of messages to the user (Information, Confirmation and Warning), you can also instantiate it with a message type as shown for the following Confirmation example (see the Dialog Pages and Message Boxes section below for pointers to documents that describe how to use these features to display Information, Confirmation and Warning messages).

```
MessageToken[] tokens = { new MessageToken("SUPPLIER_NAME", name),
                          new MessageToken("SUPPLIER_NUMBER", supplierId) };
OAException confirmMessage = new OAException("AK",
                                             "FWK_TBX_T_SUPPLIER_CREATE_CONF",
                                             tokens,
                                             OAException.CONFIRMATION,
                                             null);
```

Message Type

The `OAException`, `OAAttrValException`, and `OARowValException` classes all include constructors that accept a message type (byte) parameter. The message type parameter tells OA Framework the type of message to display to a user. Valid options include:

- OAException.ERROR
- OAException.WARNING
- OAException.INFORMATION
- OAException.CONFIRMATION
- OAException.SEVERE

OAAttrValException

If any attribute-level validation fails in a view object row or an entity object, you can throw an `OAAttrValException` as shown below.

To instantiate this exception, you must pass the following information:

- Source object type (OAException.TYP_ENTITY_OBJECT or OAException.TYP_VIEW_OBJECT)
- Full entity definition name or view instance name as appropriate
- Primary key of the entity or row
- Attribute name being validated
- Attribute value that failed validation
- Error message application short name
- Error message name

Entity Object Example

```
public void setSalary(Number value)
{
   if (value != null)
   {
      // Verify value is > 0
      if (value.compareTo(0) <= 0)
      {
         throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT, // indicates EO source
                                      getEntityDef().getFullName(), // entity name
                                      getPrimaryKey(), // entity primary key
                                      "Salary", // attribute Name
                                      value, // bad attribute value
                                      "AK", // nessage application short name
                                      "FWK_TBX_T_EMP_SALARY_REQUIRED"); // message name

      }
     setAttributeInternal(SALARY, value);
   }
} // end setSalary()
```

View Row Example

Also see the Mapping section below for additional information about raising these exceptions from a view row.

```
setDescription(String value)
{
  if("XXX".equals(value)
  {
     throw new OAAttrValException (
                               OAException.TYP_VIEW_OBJECT, // indicates VO row source
                               getViewObject().getFullName(), //View Object full usage name
                               getKey(), // row primary key
                               "Description", //attribute name
                               value, // bad attribute value
                               "FND", //message application short name
                               "ATTR_EXCEPTION"); // message name
  }
  setAttributeInternal("Description", value);
} // end setDescription()
```

OARowValException

If any row-level validation fails in a view object row or an entity object, you can throw an `OARowValException` as shown below.

To instantiate this exception, you must pass the following information:

- Full entity definition name or view instance name as appropriate
- Primary key of the entity or row
- Error message application short name
- Error message name

Entity Object Example

```
protected void validateEntity()
{
 super.validateEntity();
 if(attr1!=attr2)
   throw new OARowValException (
        getEntityDef().getFullName(), // entity full definition name
        getPrimaryKey(),              // entity object primary key
        "FND",                        // message application short name
        "ATTR_EXCEPTION");            // message name
}
```

View Row Example

Also see the Mapping section below for additional information about raising these exceptions from a view row.

```
protected void validate()
{
  super.validate();
  if(attr1!=attr2)
    throw new OARowValException (
        getViewObject().getFullName(),// View Object full usage name
        getKey(),                     // row primary key
        "FND",                        // message application short name
        "ATTR_EXCEPTION");            // message name
}
```

Overriding the Row-Level Error Prefix

When OA Framework renders row and attribute error and warning messages in tables, the message is comprised of two components: row prefix + error message. For example:

- Row 2 Error: <*Some error message relating to the entire row 2*>
- Row 2 <*Attribute Prompt*>: <*Some error message relating to the given attribute in Row 2*>

You can optionally override this prefix if the default row references are not appropriate for your user interface. For example:

- Line 2-3 Error: <*Some error message relating to this row in the table*>
- Line 2-3 <*Attribute Prompt*>: <*Some error message relating to the given attribute in this designated row*>

To implement this:

Step 1: Create a transient view object attribute to include the translated String that you want to display as the row prefix.

Step 2: Create a custom property for the view object:

1. Set its **Name** to **ROW_DISPLAY_PREFIX_ATTR_NAME**.
2. Set its **Value** to the name of the attribute that you created in Step 1.

When processing exceptions related to this view object, OA Framework will check to see if this custom property is set, and if it is, will use the designated attribute value as the row prefix.

Note: For consistency, OA Framework applies this prefix to any error or warning message that you generate, plus any row-level messages that it generates internally.

Mapping Entity Attributes into VO Attributes

When you create custom view row methods that throw exceptions originating in entity object, you must call `doEntityToVOMapping` on the exception to create a mapping between the entity object attributes and the view object attributes as shown in the following example:

```
/**
 * Approves the purchase order associated with this row.
 */
public void approve()
{
  // Whenever you write custom methods on the VO Row that call custom methods
  // on the Entity Object you need to do a manual mapping as shown below
  // to correctly handle the entity exceptions.
  try
  {
    getPurchaseOrderHeaderEO().approve();
  }
  catch(OARowValException e)
  {
    OAViewObject[] vos = {(OAViewObject)getViewObject()};
    e.doEntityToVOMapping(getApplicationModule(), vos);
    throw e;
  }
} // end approve()
```

Behind the scenes, OA Framework calls this method for you for exceptions thrown in the following methods:

- `viewRow.setAttribute()`
- `viewRow.validate()` (catches all exceptions thrown from `eo.validate()`)
- `create(AttributeList)`
- `viewRow.remove()`

Note: If you override these methods, this mapping is performed when you call `super`. If your overriding code explicitly throws entity object exceptions, then you need to call `doEntityToVOMapping`.

Bundled Exceptions

Bundled exceptions let you accumulate "peer" exceptions while proceeding with validation, and then display them as a set when you are done. These peer exceptions are grouped in a container exception called a *bundled exception*.

Bundled exceptions can include any kind of server-side exceptions (including system-level exceptions, data formatting errors, attribute validation errors, row validation errors, and entity creation errors).

Peer Exceptions List

To creat a bundled exception, you first must create a list to which you add exceptions as you encounter them:

```
ArryList peerExceptions = new ArrayList();
peerExceptions.add(new OAException(....));
peerExceptions.add(new OAException(....));
...
```

Bundled Exceptions

When you're ready to throw your bundled exception, call `OAException.getBundledOAException` to create the bundled OAException from the list of peer exceptions that you pass to it, or call `OAException.raiseBundledOAException` to create and throw the bundled OAException immediately.

Be aware of the following supplementary points:

- Bundling similar APIs are also available on OAAttrValException and OARowValException.
- See the various accessors published by the OA*Exception classes for interacting with bundled exceptions (remember that the bundled exception itself is simply a container including the peer exceptions array).

During entity and row validation, if you don't want to do your own bundling, you can also register exceptions. These exceptions will be thrown when validation completes, or when an exception is explicitly thrown as illustrated in the examples below (see the Javadoc for `oracle.apps.fnd.framework.server.OAEntityImpl` and `oracle.apps.fnd.framework.server.OAViewRowImpl`).

BC4J Bundled Exception Mode

When this mode is disabled, all exceptions thrown by the entity attribute setters are thrown right away to the calling view row, which then throws the exception to the caller. When you enable bundled exception mode, BC4J stacks exceptions thrown from the entity attribute setters, and throws them end of valdiateEntity, or when validateEntity throws an exception. All these exceptions are bundled into a single exception that is returned to the caller.

You can enable this mode by calling:

```
OADBTransaction.setBundledExceptionMode(true);
```

By default, this mode is disabled. We recommend that you do not use this feature, as OA Framework collects all exceptions on your behalf without this being needed.

Exception Examples

Example 1

The following code example illustrates how to catch exceptions and throw them as a single bundled exception.

```
public void foo()
{
   ArrayList exceptions = new ArrayList();

   for(int ...; ...; ...)
   {
     if(.....)
     {
       exceptions.add(new OAException(.....));
     }
   }

  OAException.raiseBundledOAException(exceptions);

}
```

Example 2

The following code caches exceptions thrown during `validateEntity()`, and then throws the cached exceptions as one bundled exception.

```
protected void validateEntity()
{
  super.validateEntity();

  ArrayList exceptions = new ArrayList();

  //check for duplicate Filter Name
  if (getEntityState() == STATUS_NEW)
  {
    String value = getFilterName();
    OADBTransaction tx = getOADBTransaction();
        OAApplicationModule vam = getMyValidationAM();
        FiltersVOImpl vo = vam.findViewObject("filtersViewUsage");
        if (vo == null)
    {
          vo = vam.createViewObject("filtersViewUsage","oracle.apps.qrm.filter.server.FiltersVO");
          vo.setMaxFetchSize(-1);
          vo.initQuery(value,"C");
          Row r = vo.first();
          if (r != null)
```

```
        {
               exceptions.add(
                 new OAAttrValException (
                       OAException.TYP_ENTITY_OBJECT, // Entity attribute exception.
                       getEntityDef().getFullName(), //Entity full def name
                       getPrimaryKey(), //Row primary key
                       "FilterName", //Attribute Name
                       value, //Bad Value
                       "QRM", //Message Application Short Code
                       "QRM_UNIQUE_FILTERS_ERR")); //Message Code
        }
      }

     //check for empty filters(no conditions)

       EntityDefImpl def = EntityDefImpl.findDefObject("oracle.apps.qrm.filter.server.QrmFilterConditionsEO");
       Iterator iterator = def.getAllEntityInstancesIterator(getDBTransaction());
       String flag = "no";
       while (iterator.hasNext())
       {
               QrmFilterConditionsEOImpl fcEO = (QrmFilterConditionsEOImpl)iterator.next();
               // only check rows in valid state
               if ( fcEO.getEntityState() != STATUS_DELETED && fcEO.getEntityState() != STATUS_DEAD )
       {
                 flag = "OK";
       }
       }
       if (flag.equals("no"))
       {
               exceptions.add(
                new OARowValException (
                       getEntityDef().getFullName(),
                       getPrimaryKey(), //Row primary key
                       "QRM", //Message Application Short Code
                       "QRM_NO_CONDITIONS_ERR")); //Message Code
       }

       OAException.raiseBundledOAException(exceptions);
 }
```

Example 3

The following code example caches exceptions thrown in a view object method, and then throws the cached exceptions as one bundled exception.

```
public void checkUsed()
{
   String ifSelected = null;
   String name;
   ArrayList exceptions = new ArrayList();
   FiltersVORowImpl row = (FiltersVORowImpl)first();
   while (row != null)
   {
     ifSelected = (String)row.getAttribute("SelectFlag");

     if ("Y".equals(ifSelected))
     {
        name = (String)row.getAttribute("FilterName");
        OAViewObjectImpl vo =
          (OAViewObjectImpl)getApplicationModule().findViewObject("IsFilterUsedVO");
        vo.setWhereClause(null);
        vo.setWhereClauseParams(null);
        vo.setWhereClauseParam(0,name);
        vo.executeQuery();
        Row r = vo.first();

        //if there are analyses, then use them
        if (r != null)
        {
          String msg= (String)r.getAttribute("AnalysisName");
          String flag ="f";

          while (r != null)
          {
            //change flag if it was the first row,if not append analysis name
            if (flag.equals("f"))
              flag = "N";
            else
              msg = msg +", "+ (String)r.getAttribute("AnalysisName");
              r = vo.next();
          }
          MessageToken[] tokens = {new MessageToken("FILTER_NAME",name),
                                   new MessageToken("ANALYSIS",msg)};
          exceptions.add(
             new OARowValException(
                  getViewObject().getFullName(),
                  row.getKey(),
```

```
                    "QRM",
                    "QRM_FILTER_REMOVE_ERR",
                    tokens));
          }
       }

      row =(FiltersVORowImpl)next();
    }
    OAException.raiseBundledOAException(exceptions);
}
```

Example 4

The following code example registers a validation exception in `set<Attribute>()` so BC4J can throw this exception later during the entity validation.

```
public void setAmount(oracle.jbo.Number amnt)
{
   // Clears any old exceptions for a fresh start.
   clearAttributeException("Amount");
   if(amnt < 0)
   {
      OAAttrValException attrEx = new OAAttrValException(
         OAAttrValException.TYP_ENTITY_OBJECT,
         getEntityDef().getFullName(),
         getPrimaryKey(),
         "Amount",
         amnt,
         "QRM",
         "QRM_AMOUNT_IS_NEGATIVE");

        registerAttributeException(getEntityDef().getAttributeDefImpl("Amount"),amnt, attrEx);
        }
 }
```

Example 5

This code example registers exceptions thrown during `validateEntity()` so BC4J can throw these exceptions when validation completes.

```
protected void validateEntity()
{
  super.validateEntity();

  // Clears all Row and Attribute exceptions registered in validateEntity() for a fresh start.
  clearAttributeException("FilterNAme");
  clearRowExceptions();

  //check for duplicate Filter Name
  if (getEntityState()==STATUS_NEW)
  {
     String value = getFilterName();
     OADBTransaction tx = getOADBTransaction();
     OAApplicationModule vam = getMyValidationAM();
     FiltersVOImpl vo = vam.findViewObject("filtersViewUsage");

     if(vo == null)
     {
        vo = vam.createViewObject("filtersViewUsage", "oracle.apps.qrm.filter.server.FiltersVO");
     }

     vo.setMaxFetchSize(-1);
     vo.initQuery(value,"C");
     Row r = vo.first();
     if (r != null)
      {
        OAAttrValException attrEx = new OAAttrValException (
            OAException.TYP_ENTITY_OBJECT, // Entity attribute exception.
            getEntityDef().getFullName(), //Entity full def name
            getPrimaryKey(), //Row primary key
            "FilterName", //Attribute Name
            value, //Bad Value
            "QRM", //Message Application Short Code
            "QRM_UNIQUE_FILTERS_ERR")); //Message Code

                registerAttributeException(getEntityDef().getAttributeDefImpl("FilterName"), value, attrEx);
      }
    }

        //check for empty filters(no conditions)
    EntityDefImpl def =
       EntityDefImpl.findDefObject("oracle.apps.qrm.filter.server.QrmFilterConditionsEO");
    Iterator iterator = def.getAllEntityInstancesIterator(getDBTransaction());
    String flag = "no";
    while (iterator.hasNext())
    {
      QrmFilterConditionsEOImpl fcEO = (QrmFilterConditionsEOImpl)iterator.next();
```

```
      // only check rows in valid state
      if ( fcEO.getEntityState() != STATUS_DELETED && fcEO.getEntityState() != STATUS_DEAD )
       flag = "OK";
    }
    if (flag.equals("no"))
    {
      registerRowException(
        new OARowValException (
             getEntityDef().getFullName(),
             getPrimaryKey(), //Row primary key
             "QRM", //Message Application Short Code
             "QRM_NO_CONDITIONS_ERR")); //Message Code
    }
  }
```

Dialog Pages and Message Boxes

For information about displaying modal Error, Information, Warning and Confirmation dialog pages, see Chapter 4: Implementing Dialog Pages.

For information about displaying Error, Information, Warning and Confirmation messages boxes at the top of a page (when not simply displayed automatically as a consequence of throwing an exception), see Chapter 4: Implementing Message Boxes.

Note: We will keep existing errors on the page in either of the following cases:

- If the page was submitted due to an LOV event.
- If the page was submitted with server-side validation disabled.

Creating Attribute Sets

Overview

This document describes how to create attribute sets in accordance with the OA Framework coding standards.

Contents

- Designing Attribute Sets
- Creating Attribute Set Packages Manually
- Creating Attribute Sets Manually
- Generating Attribute Sets Automatically
- Deploying Attribute Sets

Prerequisite Reading

This document assumes that you have read the following in the OA Framework Developer Guide:

- Anatomy of an OA Framework Page
- Implementing the View

Designing Attribute Sets

Note: This section is intended for internal Oracle E-Business Suite developers only.

First and foremost, all attribute sets that Oracle E-Business Suite developers create must comply with the following standards:

- The OA Framework View Coding Standards describe the circumstances under which you should (must!) create attribute sets. Note that attriubte sets are also required for service definitions. See Creating Services for additional information.
- The OA Framework File / Package / Directory Structure standards describe all the naming conventions that you should use for the attribute set packages and individual attribute sets. It also describes the directory structure that you should use for the XML package files.

Once you understand the standards, you're ready to proceed:

1. Product teams who own some of the key shared data entities (example: TCA, ITEMS and FND) should take the lead on building the attribute sets associated with their schema.
2. If you need to use a column for which no attribute set is currently published, you should coordinate with the owner team to have the attribute set created. If, for any reason, the owner team can't meet your schedule requirements, you can make an arrangement with the owner team to create the attribute set on their behalf and hand it over for source control and packaging.
3. Before creating an attribute set, make sure it is truly unique and warranted (do NOT duplicate existing attribute sets without explicit permission from the OA Framework team).
4. Changes to existing attribute sets are not permitted until an impact and mitigation study is conducted by the OA Framework team.
5. As a rule, you should generate your *table-based* attribute set packages automtically using the procedure described below, and then make any necessary manual edits.

Creating Attribute Set Packages Manually

As described in Implementing the View, all attribute sets are created into the context of an OA Component package file.

To create an attribute set package in JDeveloper:

1. In the JDeveloper Navigator, select the OA Project where you want to create your package.
2. From the main menu, choose **File > New** to open the **New Object Gallery**.
3. In the **Categories** tree, expand the **Web Tier** node, and select **OA Components**.
4. In the **Items** list, select **Package File** to open the **New Package File** dialog.
5. Enter a **Name** and a **Package** in accordance with the OA Framework File / Package / Directory Structure standards. Select **OK** to save create your <Package>.<Name>.xml OA Component document.

6. Save your work.

Creating Attribute Sets Manually

To create and edit individual attribute sets in JDeveloper:

1. Identify which Attribute Set Template (all of which are described in the next section) you should be using based on the kind of attribute set you need to create. For example, if you are creating a button attribute set, you need to use the Button Template.
2. In the JDeveloper Applications Navigator pane, select the attribute set package to display its contents in the Structure pane.
3. In the Structure pane, select the **Attribute Set** folder, right-click and select **New > Attribute Set** to open the **New Attribute Set** dialog.
4. In the **New Attribute Set** dialog, enter an **Attribute Set Name** in accordance with the OA Framework File / Package / Directory Structure standards. Per the attribute set template that you identify in Step 1, select all the appropriate properties in the **Available Properties** list and shuttle them to the **Included Properties** list. Select **OK** to create the attribute set.
5. Enter values for the attribute set's included properties in the Property Inspector.
6. Save your work.

Generating Attribute Sets Automatically (Only on Linux)

In order to facilitate the process of creating table-related attribute set packages and attribute sets, you should automatically generate these using a command-line utility.

Prerequisites

1. Before proceeding, make sure that any tables for which you want to generate attribute sets are fully described in FND_TABLES and FND_COLUMNS (to do this, you need to load properly defined XDF table definitions). The table name, column name and column descriptions defined in these entities become the basis for the generated attribute sets.
2. You should check out any preexisting attribute set packages from source control that you want to update with the generator. Note the physical location of the XML files in your working area.

Running the Attribute Set Generator

To automatically generate new and updated existing attribute sets, run the **Attribute Set Generator** from the Linux command line as shown below. This utility will create new attribute sets, and update preexisting attribute set `Comments` property values with the corresponding column descriptions in FND_COLUMNS. You can elect to generate attribute sets for an entire product, or for specific tables within a product. Once you generate your attribute sets, you can maintain all other properties in JDeveloper as described above (the generator never overrides any other property values).

Note: For persistent attributes, the `Comments` property value is *always* copied from FND_COLUMNS (the column description value in FND_COLUMNS should be treated as the single source of truth). If you need to make changes to the `Comments` property value, you need to change the underlying column description.

The resulting attribute set packages and attribute sets are named according to the OA Framework naming conventions. Correct naming and packaging is essential for service creation since the OA Framework automatically detects **table.column** attribute sets using these naming conventions, and defaults them when you create view object attributes that map to persistent entity attributes.

The following shows the online help for the `attributesets` command:

```
****** attributesets ******

Creates or Updates attribute set files

Parameter(s) passed to attributesets:
   1 = Username to database
   2 = Password to database
   3 = Database hostname
   4 = Database port number
   5 = Database SID
   6 = Xml classpath. Location to where your XML files are located
      (Example: /home/[name]/jdevhome/myprojects)
   7 = Application Short Name (Example: PO)
   ----- optional parameter(s) -----
   8 = Table Specification. Default is to generate attribute sets for all tables
       belonging to Application Short Name (parameter 7).
   Example 1) PO_VENDORS - generates the attribute set PoVendors.xml
```

```
Example 2) PO_VENDOR% - generates the attribute sets for all tables that match
                        the query PO_VENDOR%.
9 = Package (Example: /oracle/apps/po/attributesets)
```

Examples

Assuming the generated attribute set package will be `/oracle/apps/po/attributesets`, the following command generates all the attribute sets for the PO product, and puts the resulting files in the `/home/jfrost/jdevhome/jdev/myprojects/oracle/apps/po/attributesets` directory.

```
/jdevbin/<SelectedJDevBuild>/build attributesets <username> <password>
<server>.example.com 1521 devdb /home/jfrost/jdevhome/jdev/myprojects PO
```

Assuming the generated attribute set package will be `/oracle/apps/po/attributesets`, the following command generates all the attribute sets for PO tables starting with the name **PO_VENDOR**, and puts the resulting files in the `/home/jfrost/jdevhome/jdev/myprojects/oracle/apps/po/attriubtesets` directory.

```
/jdevbin/<SelectedJDevBuild>/build attributesets <username> <password>
<server>.example.com 1521 devdb /home/jfrost/jdevhome/jdev/myprojects PO PO_VENDOR%
```

Attribute Set Templates

JDeveloper supports a large variety of properties when creating an attribute sets. In consideration of reuse, customization, verticalization and translation realities, Oracle E-Business Suite product teams should limit themselves to the templates provided in this section.

- Properties designated as **Required** must be included in the attribute set definition with appropriate values.
- Optional properties must **NOT** be included in the attribute set definition unless there is a sensible property to specify. If you include a property in an attribute set without populating its value, you run the risk of unintended behavior when combining attribute set usage with extension.
- Do **NOT** include any properties not explicitly listed in these templates.

Table.Column Template

The following is a list of the properties that may be used to define table column attribute sets. You should also use this template if you create a `Transient.xml` attribute set package for commonly used attributes that have no associated base table.

Property	Description	Required	Specific Guidelines
Additional Text	Used for bubble help and tooltips.	Optional	
Comments	Desribes the associated attribute. **Note:** This value is used as for attribute-level comments as described in Creating Services. **Note:** For persistent attributes, this value should be copied from FND_TABLES as described above.	Required	Examples include: *Need-By Date* Date by which line items must be delivered. *Promise Date* Date by which supplier promises to deliver line items. *Receipt Date* Date on which line items were received.
Data Type	VARCHAR2, DATE, DATETIME, NUMBER, or BOOLEAN	Required for transient items	Specify only for transient items that has no mapping to a data entity. Otherwise must default to data type of associated data attribute.
Document Name		Required	See the OA Framework File / Package / Directory Structure standards.
Height	Display height	Optional	Use for multi-line input text only.

Length	Item display Length	Optional	Number of bytes to display, should be between 1 and maximumLength value
Maximum Length	Maximum number of bytes allowed in the item value.	Required for transient items	Specify only for transient items that has no mapping to a data entity. Otherwise must default to value length of associated data attribute.
Prompt	Item's prompt	Required	
Required	Is runtime form value required?	Required for transient items	Specify only for transient items that has no mapping to a data entity or when overriding a not-required value corresponding to a persistent data entity.
Tip Message Appl Short Name	Message Dictionary message application short name.	Optional, depends on tipType	Specify if applicable
Tip Message Name	Message Dictionary message name.	Optional, depends on tipType	Specify if applicable
Tip Type	dateFormat, longMessage, shortTip, and none. For longMessage and shortTip, you must specify the Tip Message Appl Short Name and the Tip Message Name.	Optional	Specify if applicable

Button Template

The following is a list of the properties that may be used to define button attribute sets.

Property	Description	Required	Specific Guidelines
Additional Text	Used for bubble help and tooltips.	Required	Required for buttons per Accessibility standards.
Comments	Describes attribute set usage	Required	Must include the context in which the attribute set is used.
Document Name		Required	See the OA Framework File / Package / Directory Structure standards.
Prompt	Button label	Required	N/A

Region Header Template

The following is a list of the properties that may be used to define header attribute sets.

Property	Description	Required	Specific Guidelines
Additional Text	Used for bubble help and tooltips.	Optional	N/A
Comments	Describes attribute set usage	Required	Must include the context in which the attribute set is used.
Document Name	N/A	Required	See the OA Framework File / Package / Directory Structure standards.
Icon URI	Header icon	Optional	Populate only if applicable to most uses.
Maximum Length	Maximum number of bytes allowed in the item value	Optional	Defaults to value length of associated data attribute.
Prompt	Header text	Required	N/A

Deploying Attribute Sets

Once you create your attribute sets, you need to deploy them so they can be used by other developers.

Note: the process for this is currently being designed for the Oracle E-Business Suite Division.

Internationalization

Overview

Contents

- User Preferences
- Language
- Timezone
- Dates
- Numbers/Currency
- Text and Component Alignment
- Localized Layouts
- Calendar Support
- Character Encoding of BC4J XML Files
- Text Length Validation

Prerequisite Reading

- Implementing the Model
- Implementing the View
- Implementing the Controller

Related Information

- OA Framework Naming / File / Package / Directory Structure Standards
- OA Framework View Coding Standards

User Preferences

Most OA Framework application pages include a standard global **Preferences** button, which is used to change the following locale-related session settings:

- Language
- Territory
- Timezone
- Client character encoding
- Date format
- Number format

When a user logs in, locale information is stored in ICX_SESSIONS and OA Framework automatically considers these settings when rendering pages and handling data. See OA Framework State Management for additional information about the Oracle E-Business Suite User Session.

However, you *must* use the OA Framework methods described below when doing any explicit manipulation of locale-sensitive data. This is so that it can be converted and formatted correctly according to user's preferences and cultural conventions and *not* the default locale setting and default encoding.

Note: Obtain the locale information like country, language, date/number format and so on by calling the `getUserLocale()` and `getUserLocaleContext()` methods on `oracle.apps.fnd.framework.OANLSServices`, which itself can be retrieved from an `oracle.apps.fnd.framework.webui.OAPageContext` in a controller or an `oracle.apps.fnd.framework.OADBTransaction` in model code).

Language

All text that is displayed to a user **must** be fully translatable. This section describes how you ensure this for an application.

Note: When a value is "eligible for translation," it displays in the correct language for the user assuming a translation is present. If not, the value is displayed in American English regardless of the user's language preference.

Menu Definitions and Responsibilities

All the displayable properties that you define for menus, functions, and responsibilities are eligible for translation. At runtime, the Oracle E-Business Suite displays these values in the user's language, assuming that a translation for that language has been completed.

OA Component Definitions

All the displayable properties that you define declaratively in the Oracle JDeveloper OA Extension are also eligible for translation. At runtime, OA Framework displays these values in the user's language, assuming that a translation for that language has been completed.

Some web beans let you declaratively specify a message name for long text. This message is defined in an Applications repository called *Message Dictionary*, and it is fully translatable. At runtime, OA Framework displays the appropriate text for the user's language.

Tip: See the *Oracle E-Business Suite Developer's Guide* for information about creating messages using Message Dictionary.

You can also define lists of attribute/value pairs (known as "Lookup Codes") for use in the UI to display translated values to the user while your code deals with the static attributes.

Tip: See the Oracle E-Business Suite Lookups online help for information about creating lookup codes.

To translate an OA component definition, use the OA Extension Translation toolset. Refer to Translating Personalizations Stored in MDS in the *OA Framework Personalization Guide* for detailed instructions. Although these instructions are specific to translating a personalization, you can follow the same steps, but instead of locating and specifying a personalization document to translate, you simply locate and specify a OA component definition to translate.

Code

Any other text that you display yourself, such as in exceptions or programmatically defined beans, *must* be retrieved from a translated data source (Message Dictionary, a translated product table, or from a declaratively defined web bean whose displayed values are translated). *Never* display hard-coded text strings in the UI!

Note: Oracle E-Business Suite developers using OA Framework do *not* use Java resource bundles.

To retrieve a product specific message from Message Dictionary, use `OAPageContext.getMessage()` as shown in the following example:

```
MessageToken[] tokens = { new MessageToken("PO_NUMBER", orderNumber)};
String pageHeaderText =
  pageContext.getMessage("AK", "FWK_TBX_T_PO_HEADER_TEXT", tokens);
```

To set a message for bundled exceptions or validation exceptions, use the `oracle.apps.fnd.framework.OAAttrValException` and `oracle.apps.fnd.framework.OARowValException` classes as illustrated in Error Handling.

To display the translated text for a static styled text item, for example, simply call its `getText()` method as shown. Remember to always use any web bean accessor signatures that take `OAPageContext` as a parameter:

```
OAMessageStyledTextBean styledTextBean =
  (OAMessageStyledTextBean)webBean.findIndexedChildRecursive("itemName");
String itemText = styledTextBean.getText(pageContext);
```

Tip: For information about creating messages using Message Dictionary, see the *Oracle E-Business Suite Developer's Guide*.

Timezone

With global e-business, customers from one a different timezone. In this case, the time difference (encompassing both the time zone and Daylight Saving Time) *must* be reconciled between the client and the server. See Profile Options for information about the Oracle E-Business Suite client and server timezone profile options.

OA Framework automatically reconciles this difference for you. For example, all queried date-time values are displayed in the user's time and written in the server's time. To manually convert values based on the client and server timezones, `OALNSServices` publishes a number of methods for this purpose.

Client Character Encoding

During file upload, file download or data export, OA Framework automatically sets the file encoding to the client character encoding defined by the profile option FND: NATIVE CLIENT ENCODING.

Note: The value of this profile option is an Oracle character set name (for example WE8ISO8859P1), which is not understood by Java. If you get the client character encoding value from the FND: NATIVE CLIENT ENCODING profile using the `OAPageContext` method `getProfile()`, the value returned may cause your Java code to throw a runtime exception because it will not recognize the Oracle character set name. Instead, always get the client character encoding value by calling the `OANLSServices` method `getJAVAClientEncoding()`. This method maps the Oracle character set name to its equivalent Java encoding name and returns a Java value that you can then supply to any Java API that requires encoding.

Date and Time

Date and time information can be represented as a String in Java Date/Time Format, a String in Oracle Date/Time Format, a Java Date/Time Object, or as an Oracle Date Object. You need to know what kind of format you are dealing with, and call the right methods to convert from one type to another. Some of the most common conversion methods are listed below; see the `OANLSServices` and `oracle.apps.fnd.framework.OAFwkUtils` Javadoc for additional information.

To convert a String in user specified date/time format (Oracle date/time format) to a Java Object, use the `OANLSServices` methods:

```
public Date stringToDate(String text);
public Timestamp stringToDateTime(String text);
```

To convert a Java Date/Time object to a String in the user specified date/time format (Oracle date/time format), use the following `OANLSServices` methods:

```
public String dateTimeToString(Date dateTime);
public String dateToString(Date date);
```

To convert from Oracle Date/Time Format to Java Date/Time Format, use the following `OAFwkUtils` methods. Retrieve the user specified date/time format mask by calling `OANLSServices.getDateFormatMask()`:

```
public static String oracleToJavaDateFormat(String userDateFormatMask);
public static String oracleToJavaDateFormat(String userDateFormatMask, boolean isNumeric);
public static String oracleRRRRToJavaDateFormat(String userDateFormatMask);
public static String oracleRRRRToJavaDateFormat(String userDateFormatMask, Boolean isNumeric);
```

Note: These methods support only a subset of the Oracle formats. For unsupported Oracle formats, the MM/dd/yyyy or mm/dd/yy Java format will be returned. Also, the conversion between Oracle Date/Time Format and Java Date/Time Format is currently hardcoded, so adding a new format for conversion requires an OA Framework code change.

Warning: There is a mismatch between the Java Date Oracle Date formats when the month format mask is MON. In this case, the to_date function might return an error. To avoid this, *always* set any `WHERE` clause bind variables with Date data.

The following example illustrates converting a String date to an appropriate format for use as a `WHERE` clause bind variable:

```
initQuery(String submittedDateString)
{
  Vector parameters = new Vector(1);
  StringBuffer whereClause = new StringBuffer(100); // where clause for ViewObjects.

  if ((submitedDateString != null) && (submitedDateString.length() != 0))
  {
    java.sql.Date javaSqlDate =
      transaction.getOANLSServices().stringToDate(submitedDateString);
    whereClause.append("DATE_COLUMN_NAME = :1");
    parameters.addElement(javaSqlDate);
  }

setWhereClause(whereClause.toString());
Object[] params = new Object[1];
parameters.copyInto(params);
setWhereClauseParams(params);
executeQuery();
```

Formatting DateTime Fields

The following example illustrates changing the formatter/validator of number or date beans to format DateTime fields *without* time:

```
OAWebBean bean = ...
OANLSServices nls = pageContext.getOANLSServices();
oracle.cabo.ui.validate.Formatter formatter =
  new OADateValidater(nls.getUserJavaDateFormat(), nls.getUserRRRRJavaDateFormat());
bean.setAttributeValue(ON_SUBMIT_VALIDATER_ATTR, formatter);
```

The following example illustrates changing the formatter/validator to format dates *with* time:

```
OAWebBean bean = ...
OANLSServices nls = pageContext.getOANLSServices();
oracle.cabo.ui.validate.Formatter formatter =
  new OADateValidater(nls.getUserJavaDateFormat()+"HH:mm:ss",
  nls.getUserRRRRJavaDateFormat()+"HH:mm:ss");
bean.setAttributeValue(ON_SUBMIT_VALIDATER_ATTR, formatter);
```

To provide a specific date format mask for date fields, use:

```
oracle.cabu.ui.validate.Formatter formatter =
  new OADateValidater("dd-MMM-yyyy", "dd-MMM-YY");
```

Note: Although the DateTime field gets formatted to *not* display the time, it still presents the value in the user's timezone.

Note: Arabic and Hijrah calendars are supported as of OA Framework Release 12.1. To provide a specific date format mask for date fields that include a calendar, specify the desired calendar type when using `OADateValidater`:

```
oracle.cabu.ui.validate.Formatter formatter =
  new OADateValidater("dd-MMM-yyyy", "dd-MMM-YY", calendarType);
```

The session calendarType can be retrieved by calling `OANLSServices` as follows:

```
OANLSServices nls = pageContext.getOANLSServices();
int calendarType = svc.getCalendarType(svc.getUserCalendar());
```

Numbers/Currency

Numbers

When a Number is rendered as a String in the browser, OA Framework automatically formats it based on the user's number format preference (numbers can be displayed with different grouping size, grouping separators as well as different decimal separators (for example,. "," or ".")). If you need to perform your own conversions between Java Numbers, Oracle SQL Numbers and Strings, you can use `OANLSServices` without having to worry about the user's locale because OA Framework handles this for you.

For example, to convert a String number representation to a Java Number or an Oracle SQL Number object, you should use the following APIs.

Warning: Do *not* use Java functions like `Integer.parseInt()` or `Double.parseDouble()` because they do not take the locale related number formatting into consideration when they do String parsing.

```
public Number stringToNumber(String text);
public Number stringToNumber(String text, String formatMask);
```

To convert a Java Number object to a String number representation use:

```
public String NumberToString(Number num);
public String NumberToString(Number num, String formatMask);
```

The following code illustrates correctly converting a Double to a String:

```
java.lang.Double num = new java.lang.Double(123456.7890);
String strNumber =
  pageContext.getOANLSServcies().NumberToString(num,"###,###,##0.00000;-###,###,##0.00000");
```

The following API's have been introduced in OANLSServices for BigDecimal data.

For converting String to java.math.BigDecimal object:

```
public BigDecimal stringToBigDecimal(String text);
public BigDecimal stringToBigDecimal(String text, String formatMask);
```

For converting java.math.BigDecimal to a String representation

```
public String bigDecimalToString(BigDecimal num);
public String bigDecimalToString(BigDecimal bigDecimal, String formatMask);
```

Validating Number Fields

OA Framework uses the following rules to validate number fields:

- The decimal and thousand separator characters *must* follow the preference setting 10,000.00 or 10.000,00.
- When validating, the thousand separator character is ignored.
- The decimal separator character is validated.

Examples for a 10,000.00 preference:

- 10,345,2 will be taken as 103452.
- 10.32 will be taken as 10.32.
- 10.2.3 will not pass validation.

To provide a specific number format mask for number fields, use:

```
oracle.cabu.ui.validate.Formatter formatter =
  new OADecimalValidater("###0.###;-###0.###","#.##0.###;-#,##0.###");
```

Currency

In addition to the basic number formatting, with a currency value you must also display the correct symbol for the currency code and locate it properly in relation to the value.

To convert a Number (a Java Number or an Oracle SQL Number) to its String currency representation, call `OANLSServices.formatCurrency(Number num, String currencyCode)`. To parse a String representation of a currency value to produce a Java Number, call `OANLSServices.parseCurrency(String text, String currencyCode)`.

The following example shows how to properly format a currency number value:

```
java.lang.Double num = new java.lang.Double(123456.7890);
String currencyFormattedNumber =
  pageContext.getOANLSServcies().formatCurrency(num,"USD");
```

To convert a java.math.BigDecimal to its String currency representation.

```
public String formatCurrency(BigDecimal num, String currencyCode);
```

To parse a String representation of a currency value to produce a java.math.BigDecimal

```
public BigDecimal parseCurrencyToBigDecimal(String text, String currencyCode);
```

Validating Currency Fields

OA Framework uses the following rules to validate currency fields:

- The decimal and thousand separator characters must follow the preference setting 10,000.00 or 10.000,00.
- Validation is much stricter than for a normal number field.
- The number format *must* be in exactly the current format regarding where the thousand separator is placed as well as in the number of decimal points.

Examples for a 10,000.00" preference and USD currency:

- 1,034,522 will pass the validation.
- 1,034,52 will not pass the validation.
- 10.3 will be taken as 10.30.
- 10.312 will be taken as 10.31.
- 10.2.3 will not pass validation.

Text and Component Alignment

OA Framework automatically aligns components correctly in a bi-direction session. If you explicitly align components, always set "start" and "end" alignment; *never* use "right" and "left." If you need to set this programmatically, use the OAWebBeanConstants `H_ALIGN_START` and `H_ALIGN_END` as shown:

```
OACellFormatBean cell =
  (OACellFormatBean)createWebBean(pageContext,
                                  OAWebBeanConstants.CELL_FORMAT_BEAN,
                                  null,"exampleCell");
formatedCell.setHAlign(OAWebBeanConstants.H_ALIGN_END);
```

Localized Layouts

Unfortunately, OA Framework APIs do not currently provide standard web beans for common localized layouts like name and address. In the interim, you *must* implement appropriate layouts yourself.

Calendar Support

OA Framework supports multiple calendars in date and datetime formatting and parsing. The supported calendars are below:

- Gregorian Calendar (default)
- English Hijrah Calendar
- Arabic Hijrah Calendar
- Thai Buddihst Calendar

The date formatting and parsing are handled by OA Framework automatically based on the user's preferred calendar.

If there is a requirement to support an additional calendar, as an exceptional case, developers must change the existing code to take advantage of multiple calendar support.

The exception is to format or parse a date/datetime manually with an arbitrary date format.

As an example, if the following **OANLSServices** APIs are called in the code, developers can consider calling new APIs.

```
dateToString(java.util.Date date, String formatMask)

stringToDate(String text, String formatMask)
```

OANLSServices has a variety of APIs to format or parse a date/datetime, and most of the APIs are sensitive to the calendar. However, the APIs listed above always work on a **Gregorian** based Calendar and this is intentional behavior.

The APIs that take a date format as an argument are **calendar insensitive**. This is based on the assumption that the caller of those APIs must know the date format of the date to be parsed or formatted, and the Gregorian calendar date/datetime would be expected in such cases.

If non-Gregorian calendars are required, where the above APIs are called, developers must change the existing code to call one of the following new OANLSServices APIs:

```
dateToString(java.util.Date date, String formatMask, String calendarName)

dateToString(java.util.Date date, String formatMask, int calendarType)

stringToDate(String text, String formatMask, String calendarName)

stringToDate(String text, String formatMask, int calendarType)
```

calendarName is the name of the calendar stored in the profile option.

calendarType is the integer calendar type code that is defined in `oracle.apps.fnd.i18n.text.AppsDateFormatter`. Developers have to specify the target calendar to format/parse a date/datetime with arbitrary date format.

The majority of the cases in manual date formatting/parsing can be based on the user's preferred calendar. Developers can get the user's preferred calendar using the `getUserCalendar()` API in OANLSServices.

Refer to the following code snippets to rewrite the existing API call with calendar support.

```
//Format/parse a date with user's preferred calendar
OANLSServices nls = pageContext.getOANLSServices();
//String formattedDate = nls.dateToString(date, formatMask);
String formattedDate = nls.dateToString(date, formatMask, nls.getUserCalendar());
//Date parsedDate = nls.stringToDate(text, formatMask);
Date parsedDate = nls. stringToDate(text, formatMask, nls.getUserCalendar());
```

Here are some sample cases where API calls might need to be replaced

Hardcoded Date String

If you are using hardcoded date strings in your logic, such as a particular date, "**12-31-4712"** to indicate "does not expire", the hardcoded strings are usually based on the Gregorian calendar, requiring attention to your OANLSServices API choice. To make a Date object from a hardcoded Gregorian date string, you must call a calendar insensitive API such as `OANLSService.stringToDate(String dateString, String format)`.

Calendar Insenstive Conversion

As an example, your need to convert between **OANLSServices.dateTimeToString()** and **OANLSServices.StringToDateTime()** and the conversion is to be done based on a Gregorian Calendar, the API calls must be replaced as follows:

```
OANLSServices nls = pageContext.getOANLSServices();
// String FormattedDate = nls.dateTimeToString(dateTime)
String FormattedDate = nls.dateToString(dateTime, nls.getUserJavaDateFormat() + " HH:mm:ss");
// Date parsedDate = nls.stringToDateTime(dateTimeString)
Date parsedDate = nls.stringToDate(dateTimeString, nls.getUserJavaDateFormat() + " HH:mm:ss");
```

Since `dateTimeToString()` and `StringToDateTime()` do not have a signature to specify a format or a calendar, use `dateToString()` or `stringToDate()` with the session date time format. The session date time format can be assembled from the session date format and the constant string " HH:mm:ss". Remember to add one space character before the time format. The session date format can be retrieved using `getUserJavaDateFormat()` API.

Refer to the OANLSServices javadoc for more detail of the new APIs and their calendar sensitivity.

Character Encoding of BC4J XML Files

Character encoding of BC4J region or page definition XML files is automatically set based on JDeveloper's environment encoding. JDeveloper's environment encoding is set based on your operating system's locale when you launch JDeveloper for the first time (specifically when you create your system folder).

OA Framework and the Oracle E-Business Suite Internationalization team strongly recommend using "UTF-8" for BC4J XML file encoding. Some encodings, such as Asian native encodings, may cause unexpected Java errors depending on the environment configuration.

Note: US-ASCII encoding and some European encodings are acceptable for use, based on the "Basic Encoding Set" of Java-supported encoding. If you need to use encodings other than the Java "Basic Encoding Set", please consult with the OA Framework team first.

Text Length Validation

When a text input field is bound to a database column through BC4J, OA Framework must validate the length of the text entered by the user to ensure the bound database column can accommodate the entire text without truncation. OA Framework has the following two length validators for this requirement.

1. Client-side length validator

   The client-side length validator is implemented using Javascript and works in the client browser. When a user types in text data and selects the **Submit** button, this validator immediately checks the text length in the field. If the text length exceeds the maximum length of the field, a Javascript alert window with a length validation error message appears and prevents the page from being submitted. No server roundtrip takes place.

2. Server-side length validator

   The server-side length validator is implemented using Java and works on the server side. When a user types in text data to send to the server, this validator checks the length of the submitted text data. If the text length exceeds the maximum length of the field, it throws an OAException with a validation error message. Since this validator resides in the server side, a server roundtrip takes place.

Note: The OA Framework text length validators work in byte semantic by counting the text length as bytes, not as characters.

Both text length validators are turned on ONLY WHEN the `Maximum Length` property is set for a text input field. If length validation is required, you must set the `Maximum Length` property. The value of the `Maximum Length` property must equal the precision of the corresponding VO attribute.

OA Framework offers a runtime checker that warns of cases where the `Maximum Length` property is required but not set. To turn on the checker, turn on Developer Mode by setting the **FND:Developer Mode** (FND_DEVELOPER_MODE) profile option to **Yes**. We strongly recommend you use Developer Mode during design time to identify cases of missing length validation.

If required length validation is not set, OA Framework tries to insert or update the user-entered string data to the database as is, which may cause an unexpected exception from the technology stack layer. In the worst case, the string data truncates with no error message.

Files in a Typical OA Framework Application

Overview

This document lists the files that comprise a typical OA Framework application and is organized into the following categories:

- Workspaces and Projects
- Java
- XML
- Seed Data
- Images
- JSPs
- Style Sheets

For each file type the list indicates whether you should expect to source control and package/ship the file in a product ARU.

Note: The instructions for source controlling and packaging/shipping apply to Oracle's internal E-Business Suite developers.

Related Information

- Chapter 8: OA Framework Files / Package / Directory Standards

Workspaces and Projects

Type	Description	Source Control	Package/Ship
<Name>.jws	A JDeveloper OA Workspace XML definition	Optional *	Never
<Name>.jpr	A JDeveloper OA Project XML definition	Optional *	Never
<Name>.jpx	A JDeveloper OA Project XML definition (additional file created for BC4J projects)	Optional *	Never
bc4j.xcfg	JDeveloper configuration file for each BC4J project	Optional *	Never

*** Tip:** Source controlling your workspaces and projects so people don't need to track down all the files that comprise a given module makes collaborative development easier. If you do want to source control your workspaces and projects, you should include all of the files listed here.

Java

Type	Description	Source Control	Package/Ship
UI Controller	Java controllers associated with UI regions.	Required	Required
BC4J Implementations: • Application Module • Entity Object • View Object • View Row	Java implementation for each BC4J component that you create.	Required	Required
Entity Expert	A special class associated with entity objects.	Required	Required
Utility Classes	Any supplemental classes that you create for use by other model or controller code.	Required	Required

XML

Type	Description	Source Control	Package/Ship

UI Definitions • Page • Reusable Region • Attribute Sets Package • UI Components Package	XML definition for each UI component file that you create. A file can include a complete page definition or a shared region. You can also create "packages" for attribute set definitions and related UI components (for example, all the pages/shared regions in a multistep transaction flow).	Required	Required
BC4J Definitions • Application Module • Association Object • Entity Object • View Object • View Link	XML definition for each BC4J component that you create.	Required	Required
BC4J server.xml	Automatically maintained by JDeveloper for each BC4J package that you create.	Required	Required

Seed Data

Although an OA Framework application can include any standard Oracle E-Business Suite seed data (concurrent program definitions, attachments, flexfields, profile options, lookup types/codes and so on) menus, messages and page flow Workflow definitions are listed separately since they are particularly common in any OA Framework application (the menu definition is required, and it is almost certain that you will create translatable messages).

Type	Description	Source Control	Package/Ship
Menus (.ldt)	All menu definitions associated with an OA Framework application.	Required	Required
Messages (.ldt)	Translatable text strings defined in Message Dictionary.	Required	Required
Page Flow Workflow Definitions (.wft)	Item type for an OA Framework UI page flow.	Required	Required
Others • Security (Users, Responsibilities) • Sequential Numbers • Attachments • Concurrent Program Libraries, Definitions and so on • Profile Options and Values • Multilanguage/Currencies/Territories • Lookup Types and Lookup Codes • Flexfields • Functional Workflow Definitions • Online Help Navigation Trees	Standard Oracle E-Business Suite seed data as required for any application.	Required	Required

Images

OA Framework pages can include a rich visual vocabulary. However, all images are created by the Oracle User Interface Design and Usability team and deployed centrally in special OA Framework Image ARUs, so individual applications do not include image files.

JSPs

Alll OA Framework page URLs are routed through a single, central JSP (`OA.jsp`) which is provided by the OA Framework. There is no need for an OA Framework application to create additional JSPs, although external JSPs can be embedded in an OA Framework page (see JTT/OA Framework Interoperability for additional information).

Style Sheets

All OA Framework applications use a single style sheet which is provided by the OA Framework. Oracle's internal E-Business Suite Development teams cannot ship additional, product-specific style sheets (you can, however, add styles to the central style sheet). See Working with Style Sheets additional information about this.

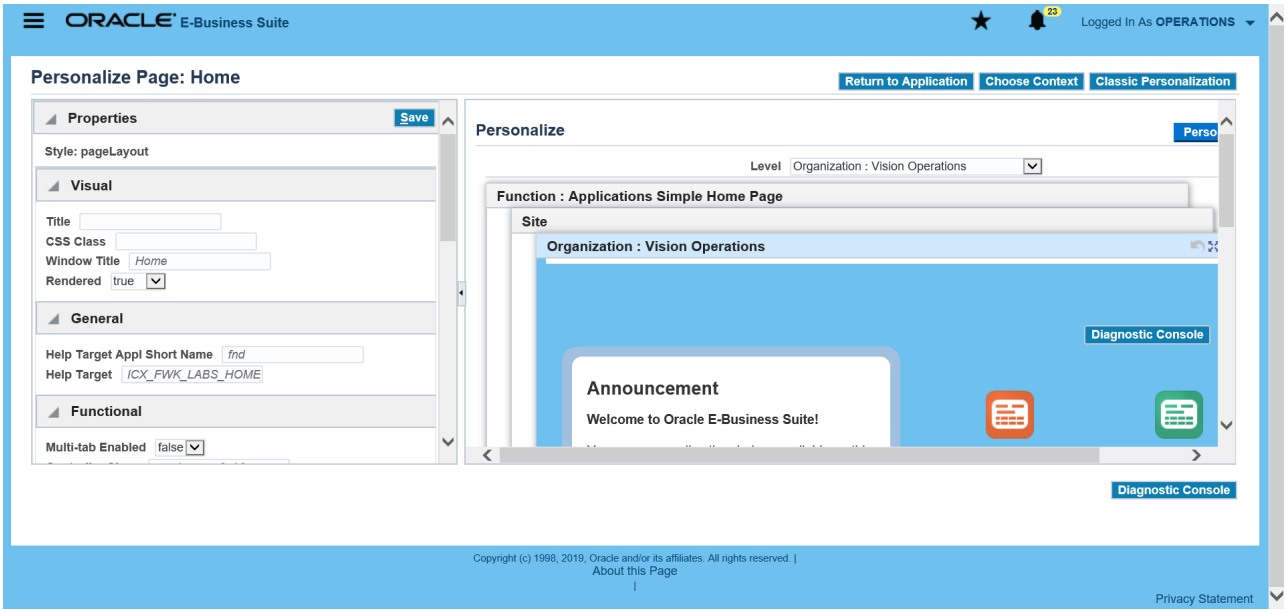
Chapter 4: Implementing Specific UI Features

Accordion

Overview

The Accordion component consists of a stack of header and body panels that can be used to provide better access to content in restricted screen space. All panels can be expanded or collapsed as needed.

Here is an example of an accordion used in the Personalization Workbench:



The Accordion component solves the problem of adjusting content in small spaces or screens, enabling all content to be collapsed and expanded as required. All sections of content can be added in different accordions, and all headers can be visible without scrolling. Selective viewing of content is available, and the space occupied by individual body panels can be controlled. Nesting of accordions is also supported.

The other component available for similar space management is hide/show, but there is no relation between the hide-show panels. Multiple header-content panels can be part of a single accordion panel.

OA Framework developers can add an Accordion to any page or region, both programmatically and via Design Time during page creation in Oracle JDeveloper. This is discussed further in the following sections.

Accordion Features

Programmatic Creation of Accordions

The user can add an accordion component on any page, region or layout. The accordion component is delivered as a layout so that other layouts and regions can be added to the body of indvidual accordions. Users can toggle between the expanding and collapsing the content by clicking on the header of individual accordions. The content expands and collapses with animation. Hovering over the header shows the selected header. Accordions can be disabled individually.

An accordion has a structure that can be demonstrated by the following example:

General	
ID	accordion1
Region Style	accordion
Extends	
Attribute Set	
Comments	
Functional	
Controller Class	
Disabled	False
BC4J	
AM Definition	
AM Instance	
Visual	
Prompt	Basic Page Structure
Rendered	True
Border Width	10
Height	
Disclosed	False
Personalization	
Admin Personalizati...	True
Personalization Label	

The above structure can be created programmatically in three parts as follows.

1. Creating an AccordionLayout bean and setting the appropriate properties:

```
OAAccordionLayoutBean accLayout = (OAAccordionLayoutBean)createWebBean(pageContext,
OAWebBeanConstants.ACCORDION_LAYOUT_BEAN ,null,"accLayout");
accLayout.setHeight("600px");
accLayout.setWidth("1000px");
webBean.addIndexedChild(accLayout);
accLayout.setDiscloseOne(false);
```

2. Creating accordion beans and adding them as indexed children to the layout:

```
OAAccordionBean accordion1 =
(OAAccordionBean)createWebBean(pageContext,OAWebBeanConstants.ACCORDION_BEAN
,null,"accordion1");
```

3. Setting required properties:

```
// Accordion text is mandatory
accordion1.setText("Basic Page Structure");
accordion1.setDisclosed(true);
accLayout.addIndexedChild(accordion1);
OABulletedListBean basicListRN =
(OABulletedListBean)webBean.findChildRecursive("BasicListRN");
accordion1.addIndexedChild(basicListRN);
OAAccordionBean disabledAcc = (OAAccordionBean)createWebBean(pageContext,
OAWebBeanConstants.ACCORDION_BEAN,null,"disabledAcc");
disabledAcc.setText("Disabled Accordion");
disabledAcc.setDisabled(true);
accLayout.addIndexedChild(disabledAcc);
```

Accordion Creation in Design Time

Accordion is available as a component during Design Time (DT) page creation in Oracle JDeveloper, with all the required features available to be set during DT.

Internationalization Support

The Accordion component deals with alignment and translation of content when using RTL sessions.

Accessibility Support

The Accordion component adheres to Oracle accessibility guidelines and is accessible to users using all browsers and devices. It also caters for voice-over software for visually challenged and keyboard-only users.

Support for Nested Accordion Layouts

The Accordion component supports nested accordions, and users can toggle between the contents of different accordions seamlessly.

Personalization Support

The Accordion component should be personalizable with important properties presented to be modified at runtime and implemented. Reordering among accordions should also be provided.

Functional and Technical Design

The Accordion component consists of two parts:

- **Accordion Layout**: This is the outer layout that contains all the accordions inside it as indexed children. Here you can specify the height and the width you want the accordion layout to occupy. It defaults to 100%.
  Accordion layout can be added to any layout or region, but can have only accordions as children.
- **Accordion**: This includes a header and body pair. Multiple accordions can be added to an Accordion layout. The header text needs to be added as prompt. The body content can be added as indexed children of the accordion. The height for each accordion can also be set individually. Any layout or region can be added as children to an accordion.

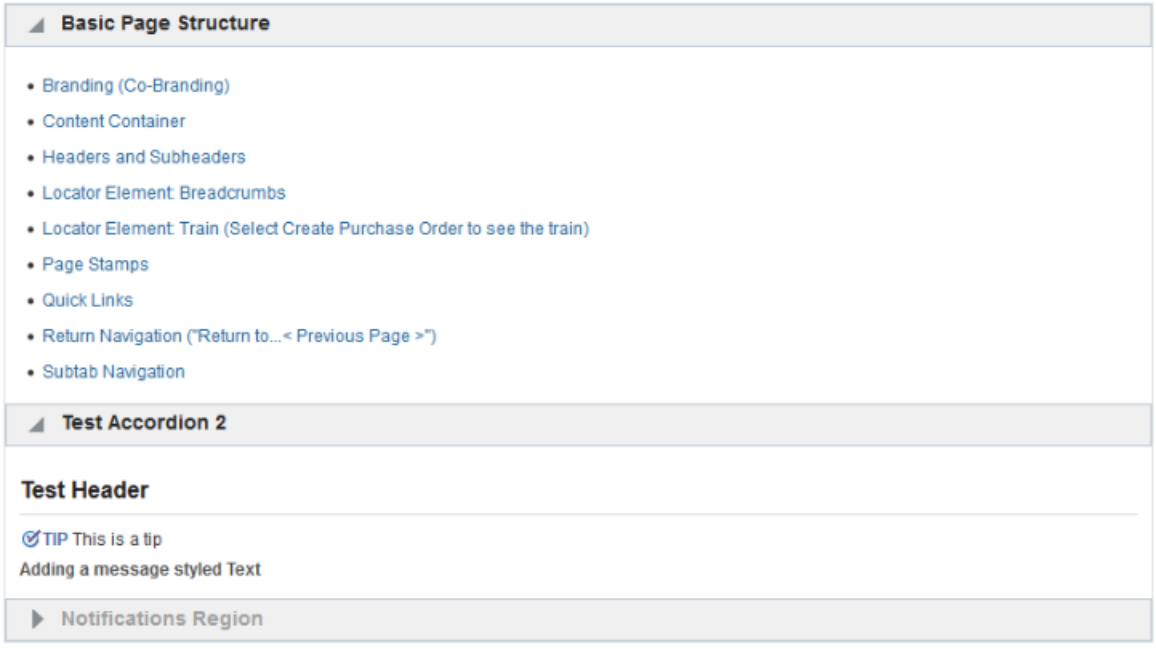
View Design

This section describes the design of the component's user interface and its metadata.

User Interface

This section describes the design of the Accordion component's user interface, and its metadata where applicable.

The screenshot below shows an accordion layout with three accordions added as indexed children with individual header and body parts. Here, the first two accordions are in an expanded state, while the third accordion is in a collapsed state as it is disabled.



To set the disclosed state of an accordion during page load, set the 'disclosed' attribute to True or False as required. A disabled accordion appears greyed out, and does not repond to onClick on touch events.

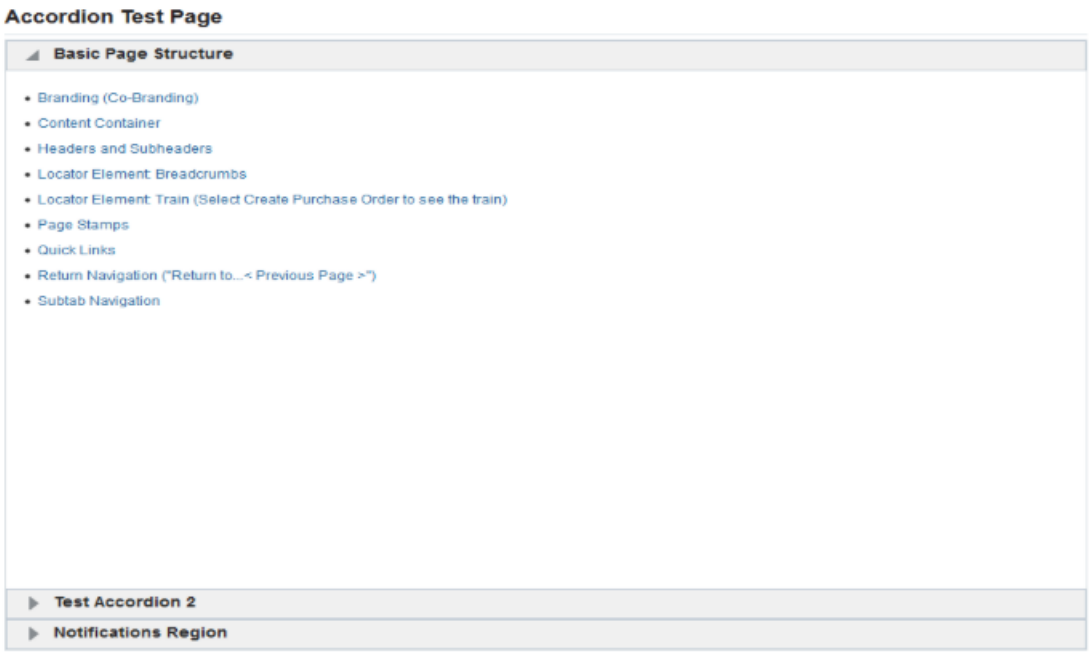
Modes

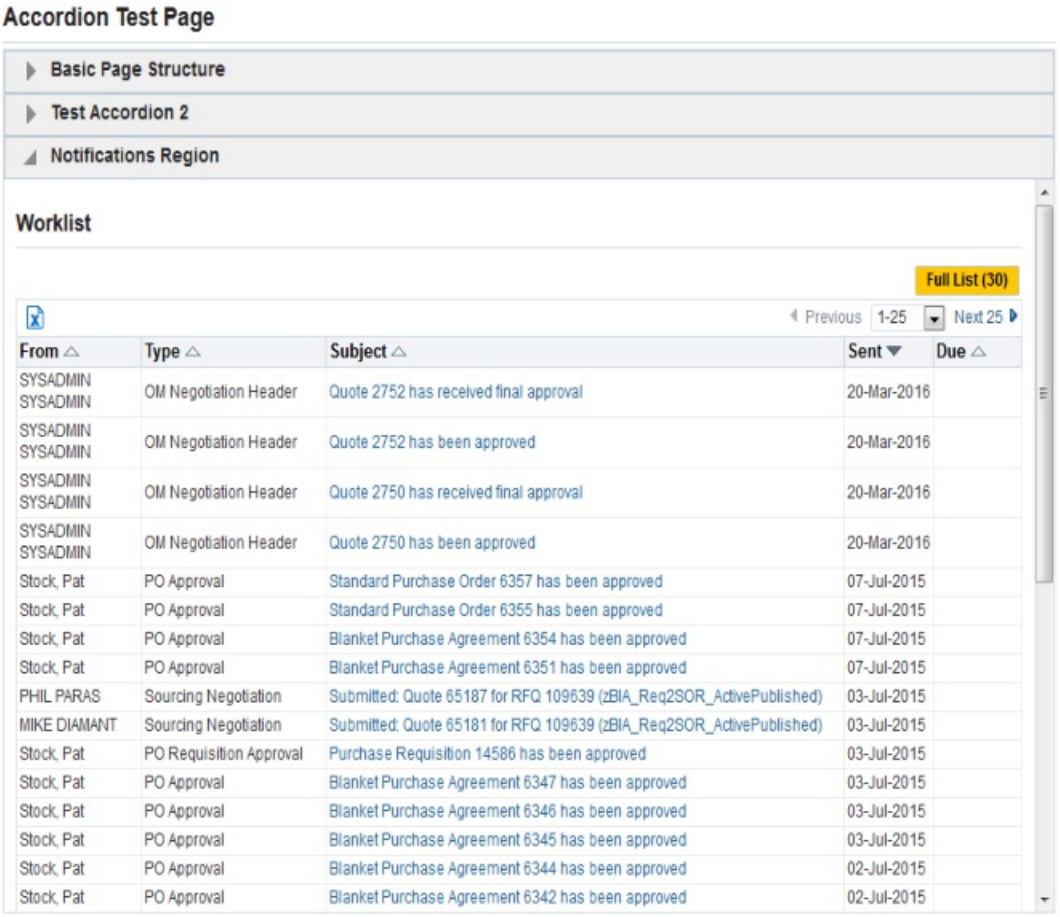
The accordion layout supports two modes: `discloseOne` and `discloseMany`. Which of these modes is used is determined by the `discloseOne` attribute on the accordion layout element.

Â

- **discloseMany**: Default state of the accordion (meaning that discloseOne is set to False). More than one accordion can be in disclosed state at all times. In this case, the accordion length flows with the content of the accordion if no specific height has been set.
- **discloseOne**: In this case only one accordion remains in disclosed state at all times, irrespective of the state set in the accordion component. On page load, the first accordion is disclosed and the others are collapsed. This layout tends to occupy all the available space in the parent layout if no height is set explicitly. Therefore, if the content does not fill the space available inside the body, you might see whitespace inside the body rather than outside. Also note that if the content overflows, you will see a scroll inside the accordion rather than on the page.

The next two screenshots both show accordion layouts that both have discloseOne set to True. The key difference is that there is less content in the first, while the second has overflowing content (hence the scroll bars).





Scrolling

There are two cases, depending on the setting of the discloseOne Boolean attribute in the Accordion bean.

- **discloseOne set to False**: This is the default mode, where more than one accordion canÂ remain opened or closed as per user action. The product team can control the height of each accordion and accordion layout separately.
  - If the accordion height is set and the content cannot all be displayed, a scroll bar will appear inside the accordion.
  - If the accordion layout has a height specified, and the content extends beyond that height, a scroll bar will appear at the accordion Layout level on opening one or more accordions. If no height is specified, it will expand and collapse with the accordions. You may see a page level/parent layout level scroll when the content goes beyond the parent layout height.

- **discloseOne set to True**: When this is set to True, only one accordion will be openÂ at all times. By design the first one will be open when the page loads. The disclosed accordion will be collapsed on opening other accordions. Setting the height of the accordion layout is mandatory in this case. The product team cannot manipulate the height of individual accordions as it is automatically set toÂ the full accordion layout height.
  - As the accordion height is automatically set in this case, on overflow you will notice a scroll inside the accordion. If the content is less, you will see whitespace inside the accordion. As only 1 accordion will be disclosed at one point of time, there will never be a scroll on the accordion layout or the outer parent.

User Interface Interactions

Users can toggle between the disclosed and undisclosed state of the accordions by clicking on the header.
When the discloseOne attribute is set to True, clicking on a disclosed accordion header will not collapse the content. Clicking on a collapsed accordion header will disclose that particular accordion and collapse the others.

Hovering over the header highlights the selected header, and the disclose/undisclose arrow icons change to blue to signify the selection. The body content of each accordion opens and closes with animation.

To add an accordion to a page:

1. Add an accordion layout as an indexed child to the page layout or any other layout as suited.
2. Add accordions as indexed children to the accordion layout and set the required header text. Then add content to each

accordion as indexed children. Accordion can also have a layout as a child.
3. Set the default disclosed state for the accordion by using the disclosed attribute.

Keyboard Interactions

The following keystrokes are supported.

Enter or Space:

- Focus on the accordion header for a collapsed panel expands the associated panel. If the implementation allows only one panel to be expanded and another panel is already expanded, that panel is collapsed.
- Focus on the accordion header for an expanded panel collapses the panel if the implementation supports collapsing.
  - Some implementations require one panel to be expanded at all times and allow only one panel to be expanded, so they do not support a collapse function.

Tab:

- Moves focus to the next focusable element.
- All focusable elements in the accordion are included in the page Tab sequence.

Shift + Tab:

- Moves focus to the previous focusable element.
- All focusable elements in the accordion are included in the page Tab sequence.

Down Arrow (Optional):

- If focus is on an accordion header, moves focus to the next accordion header.
- If focus is on the last accordion header, either does nothing or moves focus to the first accordion header.

Up Arrow (Optional):

- If focus is on an accordion header, moves focus to the previous accordion header.
- If focus is on the first accordion header, either does nothing or moves focus to the last accordion header.

Home (Optional):

- When focus is on an accordion header, moves focus to the first accordion header.

End (Optional):

- When focus is on an accordion header, moves focus to the last accordion header.

Touch Interactions

The accordion component is touch device complaint, and has been tested on major hand-held devices. The accordions can be expanded and collapsed by touch actions on the headers of the accordions. The content inside the body can be scrolled using touch.

Properties

The following table shows the properties of an accordion layout (not including general layout properties such as ID, extends controller class, and security mode).

Properties of Accordion Layout

Property Code	Property Display Name	Description	Type	Values	Default Value	Personalizable	Translatable
rendered	Rendered	Whether this component is displayed	Boolean	True/False	True	Yes	No
height	Height	Display height in pixels or percentage	Number	Integer	100%	Yes	No
width	Width	Display width in pixels or percentage	Number	Integer	100%	Yes	No

discloseOne	Disclose One	Indicates if only one accordion should be disclosed at all times	Boolean	True/False	False	Yes	No

The following table contains the properties exposed for individual accordion component (not including general element properties like ID, extends controller class, and security mode).

Properties of Accordion Component

Property Code	Property Display Name	Description	Type	Values	Default Value	Personalizable	Translatable
rendered	Rendered	Whether this component is displayed	Boolean	True/False	True	Yes	No
disabled	Disabled	Whether this accordion is disabled	Boolean	True/False	False	Yes	No
text	Prompt	Header Text	String	String	N/A	Yes	Yes
borderWidth	Border Width	Width of the border in pixels drawn around the accordion body content	Number	Integer	10	Yes	Â
height	Height	Height of the accordion body	Number	Integer	N/A	Yes	No
disclosed	Disclosed	Default disclosed state of the accordion	Boolean	True/False	False	Yes	No

Indexed and Named Children

Indexed children of Accordion layout:

Child Name	Description	Possible Occurrences	Allowed Web Bean
accordion	Accordion header and body	One or more times	accordion

Indexed children of Accordion component:

Child Name	Description	Possible Occurrences	Allowed Web Bean
Indexed child	Accordion body content	One or more times	Any web bean

Sample Page with Accordion

```
<oa:pageLayout id="pagelayoutRN" windowTitle="Accordion Test Page" title="Accordion Test Page"
amDefName="oracle.apps.fnd.framework.server.OAApplicationModule">
<ui:corporateBranding>
<oa:image id="corporateBrandingImage" source="/OA_MEDIA/FNDSSCORP.gif"/>
</ui:corporateBranding>
<ui:contents>
<oa:accordionLayout id="accLayout" discloseOne="false">
Â <ui:contents>
<oa:accordion id="accordion1" prompt="Basic Page Structure">
<ui:contents>
<oa:bulletedList id="BasicListRN" rows="10">
</oa:bulletedList>
</ui:contents>
</oa:accordion>
<oa:accordion id="accordion2" prompt="Test Accordion 2">
<ui:contents>
<oa:header id="region1" text="Test Header"/>
<oa:tip id="item1" text="This is a tip"/>
<oa:messageStyledText id="item2" prompt="Adding a message styled Text" shortDesc="Adding a
message styled Text"/>
</ui:contents>
</oa:accordion>
<oa:accordion id="accordion3" prompt="Notifications Region" disabled="true">
```

```
<ui:contents>
<oa:rowLayout id="worklistRow" adminCustomizable="false">
<ui:contents>
<oa:cellFormat id="worklistCell" vAlign="top" hAlign="start" adminCustomizable="false">
</oa:cellFormat>
</ui:contents>
</oa:rowLayout>
</ui:contents>
</oa:accordion>
</ui:contents>
</oa:accordionLayout>
Â </ui:contents>
</oa:pageLayout>
```

JDeveloper Design Time Example

The following example images sum up the principles of the Accordion component in the context of Oracle JDeveloper.

Steps to add an accordion to a page:

1. Add a region to page/region and change its region style to accordionLayout as shown below.
2. Set properties such as height, width and discloseOne as needed.

As accordionLayout can have only accordions as children. Only the option of adding accordion is seen. Any webbean can be added as an indexed child to an accordion. Properties such as Height, Disabled, Prompt, Disclosed, and Borderwidth can be set as needed. Apart from Disabled, which is a functional property, all others are visual properties. Prompt (header text) is a mandatory property and cannot be left blank.

Advanced Tables

Previously, OA Framework used `oracle.apps.fnd.framework.webui.beans.table.OATableBean`, an Oracle Application implementation of UIX `oracle.cabo.ui.beans.table.TableBean`, to render tables (as described in Classic Tables).

Now, `oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean` extends `OATableBean` to provide declarative support for existing table features that previously required programmatic control. `OAAdvancedTableBean` also provides declarative support for features not available with `OATableBean`, such as column span in a table column header. We recommend new tables be implemented as "advanced tables" based on `oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean`.

Overview

A table can contain instruction text and tip, a navigation bar, selection column, control bar, add row(s) button, column tabulation, and a recalculate button. In addition, each column can be a simple web bean (such as a text input field, a link, a poplist, and so on), or a composite container that groups these simple beans. A common example of a composite container is a flow layout containing an image and a textual description.

Refer to the Oracle Browser Look-and-Feel (BLAF) Guidelines: Tables and Table Navigation/Action Methods and the `OAAdvancedTableBean` Javadoc for additional information about tables.

This document describes how to define a table using `OAAdvancedTableBean` and discusses how to implement the various features and components of a table:

- Defining an Advanced Table
  - Declarative Implementation
    - Advanced Table under a Query Region
  - Runtime Control
    - Event Handling
    - Avoiding Stale Data Checks for Read-Only Tables
  - Personalization Considerations
- Advanced Table Accessibility Support
- Partial Page Rendering (PPR) and Tables
- Table Features and Components
  - Row Headers
  - Column Headers
  - Column Span
  - Row Span
  - Navigation
  - Selection and Control Bar
  - Table Actions
  - Sorting
  - Middle Tier Sorting of an Advanced Table
  - Adding Rows
  - Totaling
  - Detail Disclosure
  - Advanced Table-in-Advanced Table
  - Formatting a Table
  - Table Row Dynamic Poplist
  - Table Performance Issues
  - Row-Level Error Prefix
- Export Table Data
- Rich Table Interactions
- Known Issues
- Related Information

Defining an Advanced Table

This section discusses how to define an advanced table.

Declarative Implementation

Create a table by specifying appropriate information in Oracle JDeveloper OA Extension. Currently, you can declaratively specify the following features/components/attributes on a table:

- Number of rows to display in a table
- Width of a table
- Header text for individual table columns
- Column span in column headers
- Table formatting
- Single selection and multiple selection on a table and adding other controls to a selection bar
- Sorting and initial sorting of a table on up to three columns
- Adding new rows
- Totaling a column
- Detail Disclosure
- Row Headers
- Wrap Settings

Below is a brief outline of how to declaratively implement a table in OA Extension. Refer to the Advanced Table example of the Sample Library to help understand the following steps. For more detail about how to modify the default behavior of a table or to implement other features of a table, refer to the specific table features and components.

Step 1: To define a table, create a new region, and set its `Region Style` property to advancedTable, OA Framework instantiates an `OAAdvancedTableBean` at runtime.

Note: Unless it is defined in or as a reusable region, the table region *must* be a recursive child of a page layout region. The page layout region *must* have its `Form` property set to **True** because a table is capable of form submissions and therefore should be enclosed in a form.

Step 2: Using the OA Extension Property Inspector, set the following properties on the new advanced table region (* indicates a required property):

Properties	Description
ID*	Specify an identifier that uniquely identifies the table in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
View Instance*	Enter the BC4J view object that provides data for the table. All columns in the table derives its data from this source. **Note:** Unless your tables are read-only, where no events are ever invoked on the tables (such as with navigation and sorting), you can *not* have two tables on the same page based on the same view object. Table event handling results in modifications to the state of the underlying view object (range start, range size) and to the values of the view attributes. If you have two tables based on the same view object on the same page, any event on one table alters the content and state of the other table.
Text	Specify text that appears as the title above the table.
Additional Text	To meet the standards of 508, specify summary text about the table. This text maps to the HTML summary tag on the table.
Records Displayed	Set to the number of rows you wish to display in the table. The default is 10 as per Oracle Browser Look and Feel (BLAF) Guidelines.
Width	Set the width of the table in pixels or as a percentage (by including the percent symbol '%' after the value.) If you set the Width to 100%, the **Next** and **Previous** links that allow you to navigate among the rows in a table, remain visible above the table without horizontal scrolling. This is especially useful when you have a wide table that requires horizontal scrolling.
Empty Table Text	Specify alternate text that you want to display if the table is empty. If no value is specified for this property and the query associated with the table's view object has not been executed, the default text, "No search conducted.", appears in the empty table. However, if a search has been conducted, but fetches no rows, the default text, "No data exists.", appears in the empty table.
Banding Type	Specify the type of banding for the table: noBanding, columnBanding, or rowBanding. The default is noBanding.
Banding Interval	If you specify row or column banding, you can specify a number to indicate the banding interval. The default is 1. As an example, if you specify rowBanding and the banding interval is 1, the table displays an alternating gray band across every other row.
Controller Class	(Optional) To handle specific table events and/or perform table-specific actions, centralize such code in a controller and specify the name of that controller in the `Controller Class` property for the table. Refer to Implementing the Controller for additional information and examples of Controllers.
Admin Personalization	Set to **True** or **False** to indicate whether an administrator can personalize this region.
User Personalization	Set to **True** or **False** to indicate whether a user can personalize this region.

Step 3: While no other properties are required, there are other optional properties you may set on the table to modify the default behavior of your table. You can see the full list of properties on a table in the OA Extension Property Inspector when you create a region of Region Style advancedTable. Some of these properties are discussed in more detail when you implement specific features in a table. You can also refer to the *OA Component Reference* in the JDeveloper Help.

Step 4: Define columns for your table by adding a column or columnGroup container to the advanced table. The column and columnGroup containers are indexed children of the advancedTable region. A column is the encapsulation of one column of a table. It includes an actual item web bean, such as `messageStyledText` or `messageTextInput`, as well as column format, column header, and column header data/format information. Any properties you set for a column override the table-level settings of the same properties for that column. To create a column container, select the advancedTable region in the Structure pane. Choose **New > column** from the context menu.

A columnGroup is a grouping of columns displayed under a common header. A columnGroup enables you to create a column containing "children" columns, also known as a column span. A columnGroup can contain columns and other columnGroups and encapsulates only the column header and column header data for the columnGroup, and not for the columns and columnGroups under it. To create a columnGroup, select the advancedTable region, choose **New > columnGroup** from the context menu. A child column is automatically added under the columnGroup. Create additional children under the columnGroup by selecting the columnGroup, then choosing **New > column** or **columnGroup** from the context menu.

Step 5: In the OA Extension Property Inspector, set the following optional properties on the column containers:

Optional Properties	Description
Total Value	Set this property to **True** to enable Totaling on this column. See the section on Totaling for additional information.
No Wrap	Specify whether the column content can wrap.
Banding Shade	Specify whether column banding should be dark or light in appearance.
Alignment	Specify whether the column content alignment is textFormat (Start), iconButtonFormat (Center), or numberFormat (Right). The default alignment is textFormat. (For old tables of the 'table' region style, the alignment defaulted to the data type of the item. Now with the 'advancedTable' regions, you must set the alignment yourself.) **Note:** The BLAF standards for the alignment of numeric column content has been revised. Read only and updateable numeric values that could be totaled or compared across cells in columns (like a currency formatted value or quantity) may be set as right-aligned together with their column headers. Numeric and alphanumeric values (such as, social security, passport, sku, or identification numbers) that are not used for totaling or comparisons across cells in columns, are left-aligned together with their related column headers. Therefore, if your column contains numeric content of the latter type, set the `Alignment` property to textFormat.
Grid Displayed	Specify whether the grid line to the left of the column should be displayed. This property acts at the column level, independently of the Display Vertical Gridlines property.
Display Vertical Gridlines	As of Release 12.2.5, specify **True** or **False** to indicate whether to display the vertical gridlines of the table. This property acts at the table level, independently of the Grid Displayed property. This property is useful for turning off gridlines in tables displayed on tablet clients.

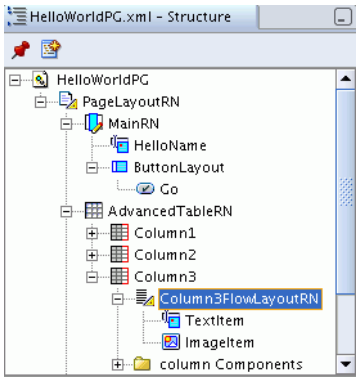
Width	Specify the width of the column in pixels or percentage.
Admin Personalization	Set to **True** or **False** to indicate whether an administrator can personalize this column.
User Personalization	Set to **True** or **False** to indicate whether a user can personalize this column.

Note: Refer to the Oracle JDeveloper OA Extension online help for information about other properties you can set on the columns.

Step 6: Specify the actual web bean you want to render under each column container. Select the column in the Structure pane, then choose **New** from the context menu and select one of the following:

- **item** - (leaf item). The item style you specify determines the properties you can set on the leaf item. See the full list of properties relevant to a specific leaf item in the OA Extension Property Inspector. Examples of leaf items that you can implement include a checkbox or a poplist.
- **switcher** - Defines a table content switcher. A table content Switcher contains two or more leaf items representing possible display alternatives. The item that displays in the Switcher column at runtime depends on the nested child item name that is returned from a specific view object attribute mapped to the Switcher column. Refer to the discussion on Table Content Switchers for more information.
- **hideShow** - Defines a Hide/Show region.
- **flowLayout** - Defines a flow layout region.
- **tableLayout** - Defines a table layout region.
- **stackLayout** - Defines a stack layout region.
- **messageComponentLayout** - Defines a messageComponentLayout region.

Note: The hideShow, flowLayout, tableLayout and stackLayout regions group their underlying leaf items into composite columns. An example of a composite column is a flowLayout region that displays an image and a description. Another example is a tableLayout or messageComponentLayout region that shows item prompts for the items that are rendered in a cell. In the case of the tableLayout region, it has a rowLayout child region, which in turn has a cellFormat child region. The cellFormat child region has two leaf items, both of which could be messageStyledText to render a prompt and a value for each cell under the column. The figure below shows an example of a composite column region (Column3FlowLayoutRN) containing two leaf items (TextItem and ImageItem).



Attention: Defining a graphTable region under an Advanced Table column or under a layout region beneath an Advanced Table column, is *not* supported.

Step 7: Set the following common properties for all leaf items of a column:

Properties	Description
View Attribute	Enter the name of the view object attribute that provides data for this column. The attribute should come from the same BC4J view object specified for the advancedTable region.
Read Only	Set this property to **False** if you want the column to be updateable.
Prompt	Set this property to display the prompt text for the leaf item if: • The leaf item is a child of a tableLayout or messageComponentLayout region. **Note:** If the leaf item is not under a container, then this Prompt property is ignored because the prompt text in this case is derived from the sortableHeader web bean. • The leaf item is a required field and you want the "Required Field Missing" error message to display this prompt. • You plan to allow users to export data from an advanced table. In the past, the Prompt property of the sortableHeader under the columnHeader grouping was not picked up during export. Therefore, if you did not specify a value for the Prompt property of a leaf item, the data was exported without a column header. If the Prompt property on the leaf item is null, OA Framework takes the Prompt value from the columnHeader's sortableHeader element. In other words, if you do not specify a value for a leaf item's Prompt property, the end user who performs a data export from the advanced table will see the same column header from the screen as in their exported data. If you set a value for the leaf item's Prompt property so that it is different from the columnHeader's sortableHeader Prompt property, the user will see the value from that leaf item's Prompt property in the exported data.
Admin Personalization	Set to **True** or **False** to indicate whether an administrator can personalize this leaf item.
User Personalization	Set to **True** or **False** to indicate whether a user can personalize this leaf item.

Note: The optional property Export View Attribute allows you to specify a different view attribute from which to export data for the leaf item. In other words, if you specify a value for this property, and a user selects an export function to export data from the table's parent page or region, the data that is exported for this leaf item is from the view attribute specified in the Export View Attribute property, and may not be identical to the data displayed.

Note: Refer to the Oracle JDeveloper OA Extension online help for information about other properties you can set on the leaf items.

Step 8: Both column and columnGroup containers encapsulate column header and column header data information. When you create a column or columnGroup container, a columnHeader grouping is automatically created under each column or columnGroup in the Structure pane. Add a sortableHeader child under the columnHeader grouping to encapsulate all the information related to the column header that you want to define.

Warning: You *must* create a sortableHeader child even if you do not want your column to be sortable.

Select the columnHeader in the Structure pane, then choose **New > sortableHeader** from the context menu.

Step 9: Set the following properties on the sortableHeader (* indicates a required property):

Properties	Description
`Prompt`*	Set the text for the column header area.
`No Wrap`	Specify whether the column header content can wrap.
`Required Indicator`	Set this property to **yes** if you want the 'Required' field indicator displayed in the column header of this column. To make a column actually perform client-side validations for the required field, the actual web bean added under the column bean must also have the `Required` property set to **yes**.
`Abbreviation`	Set this property to an abbreviation for the column header's text. It's read by screen readers when a user is navigating the cells within a table. This property is especially useful if you have a long table column heading and want to avoid having the screen reader read the long column heading repetitiously. **Note:** The abbreviation is rendered in HTML *only* when the Self Service Accessibility Features profile is set to **Screen Reader Optimized**.
`Sort Allowed`	Set this property to **yes** to enable sorting on the column. See the section on Sorting for more information. Refer to the Advanced Table example of the Sample Library for an illustration.
`Initial Sort Sequence`	Set this property to specify the level at which this column is sorted when the page first renders. Refer to the Sorting section for more information.
`Admin Personalization`	Set to **True** or **False** to indicate whether an administrator can personalize the column.
`User Personalization`	Set to **True** or **False** to indicate whether a user can personalize the column.

Note: Refer to the Oracle JDeveloper OA Extension online help for information about other properties you can set on the column headers.

Step 10: To optionally add a `formValue` web bean (a value that gets submitted with your form, but is not displayed to the user) to your Advanced Table, select the Advanced Table region, and choose **New > formValue** from the context menu.

Advanced Table Under a Query Region

If you define an advanced table under a query region, you should consider the following:

- You can prevent an end-user from personalizing a column in an advanced table by setting the `User Personalization` property of the column, its leaf node, and its sortable header to **False.**
- If you want your controller to hide a leaf node whose `User Personalization` property is set to **False**, set the column web bean's `Rendered` flag to **false** in the controller. This automatically hides the column, the leaf node, and the sortable header. If you have a columnGroup web bean, you can simply set its `Rendered` flag to **false** to hide all the columns under it.
- If you want your metadata to hide a column whose `User Personalization` property is set to **True**, set the `Rendered` property of the column and its leaf node to **False**. If the leaf node is a nested region, you can just set the `Rendered` property on the nested region; you do not have to set it on all the child nodes recursively.

  An end-user can later display the hidden column by using the Personalize Views Page to create a new view that adds the column to the list of displayed columns in her personalized view. OA Framework then sets the `Rendered` property to **True** and automatically propogates this change up the component hierarchy of the advanced table column for that user's view.

If you create an advancedTable region that is not defined under a query region:

- To hide a column, simply set the `Rendered` property of the column to **false**. This automatically hides the column, its leaf node, and its sortable header.

Runtime Control

When OA Framework first renders your table, it does not automatically execute the query on the view instance for performance reasons. To complete this implementation, you *must* perform one programmatic step. Include code in your controller to bind in your parameters and execute the view instance to populate the table with data. You can find an example of how this is done in the Framework Toolbox Tutorial: Create - Part 1.

Unlike tables implemented with `OATableBean`, no post-processing is required to set up the behavior of a table implemented with `OAAdvancedTableBean`. All properties and named children defined in OA Extension get handled before control is passed to your controllers. This means there is no need to use the `OATableBean prepareForRendering` method. Also note that from Release 12.2.5 you may either call the `setVerticalGridlinesDisplayed` API in `OAAdvancedTableBean` to set the `verticalGridlinesDisplayed` property at runtime, or call the `isVerticalGridlinesDisplayed` API to get the property value.

You may sometimes have a need to bind a column or a column's index child's rendered_attr to an VO attribute value, either through SPEL or by OADataBoundValueViewObject explicitly. In such cases, you need to ensure that the value returned for every row for the VO attribute is consistent. For example, this attribute should return either all 'False', or 'True' or 'Null' for every row. Otherwise, this may lead to an IndexOutOfBoundsException.

Event Handling

Table events are HTTP requests that are trapped and processed by OA Framework and handled during the `processFormRequest` phase.

The various table events are:

- Navigation - user selects the **Next** or **Previous** link to navigate between different ranges of rows.
- Sorting - user selects a beveled column heading to sort that column in ascending or descending order.
- Insertion of a new row - user selects the **Add Another Row** button.
- Recalculate column Total - user selects the **Recalculate** button to update the column total.
- Detail Disclosure - user selects the **Hide** or **Show** link to collapse or expand the detail disclosure region.
- Table Control - user selects the **Action/Navigation** button in the table Control bar.

Familiarize yourself with the set of UIX "hidden" fields so you can capture these events and implement custom behavior on them. The "hidden" fields are UIX attributes in the `UIConstants` interface. These parameters are set only when a form submit event is induced from a table. They are:

- `SOURCE_PARAM` - indicates the source of the event that is generating the current browser request. This maps to the name attribute of the web bean. If you wish to check whether a table generated an event, include the following in your code:

```
if (tableBean.getName().equals(pageContext.getParameter(SOURCE_PARAM)))
{
  ...
}
```

- `EVENT_PARAM` - indicates the event generated by a web bean (a table, in this case). The possible events generated by a table are:
  - `GOTO_EVENT` - when **Next** or **Previous** navigation links are selected.
  - `SORT_EVENT` - when a column header is selected to sort that column.
  - `HIDE_EVENT` - when the **Hide** link of a detail disclosure is selected.
  - `SHOW_EVENT` - when the **Show** link of a detail disclosure is selected.
  - `ADD_ROWS_EVENT` - when the **Add Another Row** button is selected.
  - `UPDATE_EVENT` - when the total row **Recalculate** button is selected.
- `VALUE_PARAM` - indicates a value that is relevant to a particular event:
  - When a detail disclosure Hide/Show is selected, the value parameter contains the row index corresponding to the row whose Hide/Show was selected.
  - When the **Next** or **Previous** link of table navigation bar is selected, the value parameter contains the index of the first row of the current range. For example, when the row range 1-10 is displayed, the value is 1 and when the row range 11-20 is displayed, the value is 11.
- `SIZE_PARAM` - indicates the number of rows currently displayed in the table (relevant only to the navigation event).
- `STATE_PARAM` - indicates the current sort state (ascending or descending) of the column on which sorting is invoked (relevant only for the sort event).

Example Usage

To check for the "Add Rows" event:

```
if (tableBean.getName().equals(pageContext.getParameter(SOURCE_PARAM)))
  && ADD_ROWS_EVENT.equals(pageContext.getParameter(EVENT_PARAM)))
{
  ...
}
```

Avoiding Stale Data Checks for Read-Only Tables

When you query a table, OA Framework calls `processFormData` on `OAAdvancedTableBean`, which in turn performs a primary key check between the rows currently displayed in the table and the rows in the view object result set. If a primary key check fails, a stale data error is thrown. In the case of a read-only table that has previously displayed search results, a stale data error is an indication that the submitted table rows from the displayed page cached by the browser and the latest view object rows are based on different query criteria. In other words, a scenario similar to the following took place:

1. The user performed a search to display results in the read-only table.
2. The user updated the search criteria and selected Go to requery the view object.
3. The user selected the browser's Back button to return to the previous results.
4. The user selected Go again, to requery the view object based on the newer search criteria specified in step 2.

For read-only tables, this kind of stale data check could be avoided for performance and browser Back button compliance reasons.

To avoid stale data checks for read-only tables use the following API on `OAAdvancedTableBean`:

```
setSkipProcessFormData(pageContext, true);
```

Warning: Do not call this method on updatable tables, as setting `skipProcessFormData` to true will clear all the changes done on the table.

Personalization Considerations

You can personalize the `Row Header View Attribute` property of an Advanced Table to render a separate column as a Row Header.

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Advanced Table Accessibility Support

To support accessibility for screen readers, you must specify a column in an advanced table as a row header. Every row in the table must refer to a column as the row header that uniquely identifies the row so the screen reader can read it. Screen reader users must also define their Accessibility mode by setting the profile Self Service Accessibility Features to **Screen Reader Optimized**.

The underlying UIX `oracle.cabo.ui.beans.table.TableBean` supports accessibility by defaulting a table's first column as the row header. However, you can set a different column as the row header, especially if a user is likely to personalize the table by turning off rendering of the first column or if the first column can return no data.

To declaratively set a column as a row header, in the OA Extension Property Inspector, specify a column ID (String) for the `Row Header Column` property (XML attribute: `rowheadercol`) of the advanced table region.

To programmatically set a column as a row header, use the following API on `OAAdvancedTableBean`:

```
setRowHeaderColumn(<columnName>);
```

As of Release 12.2, you may also personalize the `Row Header Col` property of an Advanced Table to set a column as a Row Header for screen reader accessibility.

Similarly, you must also specify column headers for your table. A data table needs column headers to meet accessibility guidelines. If you wish to implement a table without column headers, design it as a tableLayout region.

Note: To support accessibility for screen readers, the detail disclosure of a table row is created as part of that table row's cell rather than as a table within a separate detail row.

Partial Page Rendering (PPR) and Tables

In certain cases, you may want to include a mix of updatable and non-updatable rows in a table. This is accomplished by using partial page rendering at the row level for items in a table (as well as Table-in-Table). Refer to the Dynamic User Interface document for more details. You should first familiarize yourself with how to implement PPR in general, then refer to the section on PPR and Tables.

Table Features and Components

This section discusses the following:

- Row Headers
- Column Headers
- Column Span
- Row Span
- Navigation
- Selection and Control Bar
- Table Actions
- Sorting
- Adding Rows
- Totaling
- Detail Disclosure
- Advanced Table-in-Advanced Table
- Formatting a Table
- Table Row Dynamic Poplist
- Table Performance Issues
- Row-Level Error Prefix

Row Headers

The figure below illustrates a table with column headers and row headers.

	Column	Column
Row	Cell	Cell
Row	Cell	Cell
Row	Cell	Cell
Row	Cell	Cell
Row	Cell	Cell

Declarative Implementation

To implement a row header, you need only specify a value for the `Row Header View Attribute` property on the advancedTable region in OA Extension.

The `Row Header View Attribute` is the view attribute associated with the table view object that determines the values for the row headers. Just as the view attribute for a column leaf item supplies values for the cells of that column on query execution, the `Row Header View Attribute` supplies values for the row header cells.

Runtime Control

There is no runtime control code necessary to format a row header.

Personalization Consideration

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Column Headers

The figure below illustrates a table with column headers.

Select	First Name	Last Name	Salary	Start Date
☐	Peter	Smith	145000	17-Jun-1998
☐	James	Brown	152000	23-Apr-1994
☐	Lucy	Taylor	135000	13-Mar-1994
☐	Fiona	Lincoln	97000	12-Mar-1997
☐	Charles	Graham	56000	22-Nov-2001

Declarative Implementation

The header text of a column is set by the `Prompt` property value of the column's sortableHeader. If a column is a composite column of tableLayout region style, each child (leaf item) of the tableLayout region also displays a prompt when you specify a value in the `Prompt` property of the leaf item. The alignment of the column header is dependent on the value of the `Alignment` property set on the column container:

Value of `Alignment` Property	Resulting Column Header Alignment
iconButtonFormat	Center
numberFormat	Right
objectNameFormat	Left
textFormat	Left

Runtime Control

To change the column header text, please use the following code:

```
// Get a handle to the column's header
OAColumnBean columnBean =
  (OAColumnBean)tableBean.findIndexedChildRecursive("<columnBeanId>");
OASortableHeaderBean colHeaderBean =
  (OASortableHeaderBean)columnBean.getColumnHeader();
colHeaderBean.setText("<newText>");
```

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Column Span

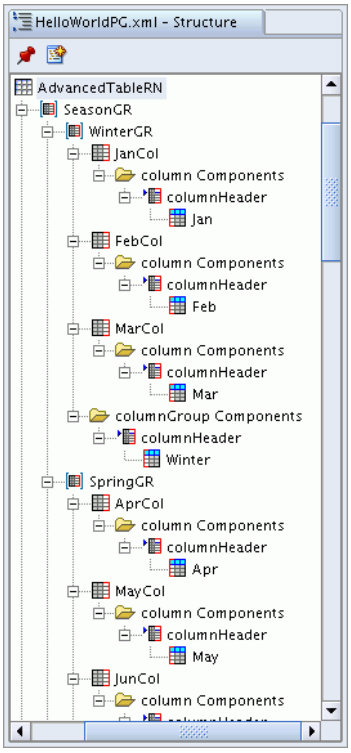
Use column span when you need to display a column that contains "children" columns. A column span is implemented in OA Extension as a columnGroup container. A columnGroup can contain columns and other columnGroups and encapsulates only the column header and column header data for the columnGroup. Not for the columns and columnGroups under it.

An example of column span is shown in the figure below, where the first column is actually a top level columnGroup labeled as Column Header. Beneath it are two "children" columns labeled Winter and Spring that are actually columnGroups. These columnGroups consist of another level of columns that are labeled for the months of each season. The figure shows that you can have a combination of columns with and without column span in a single table.

Column Header						
Winter			Spring			
Jan	Feb	Mar	Apr	May	Jun	Column Header

Declarative Implementation

Column span implementation is integrated in the advancedTable creation steps and is described specifically in Step 4 of Defining an Advanced Table.



To implement the column span example shown in the figure above, follow the steps below.

Note: These steps only highlight how to create the column span structure. They do not into detail about how to set the various properties of the advanced table region and its children.

Step 1: Select the advanceTable region and choose **New > columnGroup** from the context menu. Select the sortableHeader for this columnGroup and specify a value

for its `Prompt` property. In the figure above, the value is set to **Column Header**.

Step 2: Select the newly created columnGroup and choose **New > columnGroup** from the context menu to create the next level of columns that represent the seasons. Select the sortableHeader for this columnGroup and enter the value **Fall** for its `Prompt` property. Select the columnGroup you created in Step 1 again, and choose **New > columnGroup** from the context menu. Select the sortableHeader for this columnGroup and enter the value **Spring** for its `Prompt` property.

Step 3: Select the columnGroup you created for 'Fall' in Step 2 and choose **New > column** from the context menu. Select the sortableHeader for this column and enter the value **Sep** for its `Prompt` property. Repeat for the other two months in the 'Fall' columnGroup. Repeat this entire step to similarly create columns for each month in the 'Spring' columnGroup.

Step 4: Select the advanceTable region and choose **New > column** from the context menu. Select the sortableHeader for this column and specify a value for its `Prompt` property. In the figure above, the value for the column without column span is also set to **Column Header**.

Runtime Control

Although you should implement a column span declaratively, there may be occasions when you need to create a column span dynamically because you do not know the number of sub-columns until runtime.

First, perform the following steps to create the column web beans:

Step 1: Create a web bean of style `COLUMN_BEAN` and add it as an indexed child of the `advancedTable` bean in your controller. Set the mandatory property - `ID`.

Step 2: Create a web bean of style `SORTABLE_HEADER_BEAN` and set it as the column header (`setColumnHeader`) of the column web bean created in Step 1. Set the mandatory property - `Prompt`.

Step 3: Create the actual leaf item, such as messageStyledText, and add it as an indexed child of the column web bean created in Step 1. Set the mandatory property - `View Attribute`.

Second, create a column group web bean to achieve the column span as follows:

Note: Before proceeding, set the View Instance property on the advancedTable region declaratively.

Step 1: Create a web bean of style `COLUMN_GROUP_BEAN` and add it as an indexed child of the `advancedTable` bean in your controller. Set the mandatory property - `ID`.

Step 2: Create a web bean of style `SORTABLE_HEADER_BEAN` and set it as the column header (`setColumnHeader`) of the column group web bean created in Step 4. Set the mandatory property - `Prompt`.

Step 3: Create as many column web beans as needed following Step 1 to Step 3. Add them as indexed children of the column group web bean created in Step 4.

Step 4: You can add column groups under column groups following this outline of steps.

Example

The following code example creates a column group and that contains two columns.

```
...
// Get a handle to the advanced table
OAAdvancedTableBean tableBean = ...;

// Create a column group, create the set the column header,
// and add the column group under the advanced table
OAColumnGroupBean columnGroup = (OAColumnGroupBean)
  createWebBean(pageContext, COLUMN_GROUP_BEAN, null, "ColGroup");
OASortableHeaderBean columnGroupHeader = (OASortableHeaderBean)
  createWebBean(pageContext, SORTABLE_HEADER_BEAN, null, "ColGroupHeader"); columnGroupHeader.setText("Column Group");

// Retrieve from message dictionary
columnGroup.setColumnHeader(columnGroupHeader);
tableBean.addIndexedChild(columnGroup);

// Create a column, create the set the column header, and add the column
// under the column group
OAColumnBean column1 = (OAColumnBean)createWebBean(pageContext, COLUMN_BEAN,
  null, "Column1");
OASortableHeaderBean column1Header = (OASortableHeaderBean)
  createWebBean(pageContext, SORTABLE_HEADER_BEAN, null, "Column1Header");
column1Header.setText("Column 1");
column1.setColumnHeader(column1Header);
columnGroup.addIndexedChild(column1);

// Create the actual leaf item under the first column
OAMessageStyledTextBean leaf1 = (OAMessageStyledTextBean)
  createWebBean(pageContext, MESSAGE_STYLED_TEXT_BEAN, null, "Leaf1");
leaf1.setViewAttributeName("<viewAttr1>");
column1.addIndexedChild(leaf1);

// Create another column, create the set the column header, and
// add the column under the column group
OAColumnBean column2 = (OAColumnBean)
  createWebBean(pageContext, COLUMN_BEAN, null, "Column2");
OASortableHeaderBean column2Header = (OASortableHeaderBean)
  createWebBean(pageContext, SORTABLE_HEADER_BEAN, null, "Column2Header");
column2Header.setText("Column 2");
column2.setColumnHeader(column2Header);
columnGroup.addIndexedChild(column2);

// Create the actual leaf item under the second column
OAMessageStyledTextBean leaf2 = (OAMessageStyledTextBean)
  createWebBean(pageContext, MESSAGE_STYLED_TEXT_BEAN, null, "Leaf2");
```

```
leaf2.setViewAttributeName("<viewAttr2>");
column2.addIndexedChild(leaf2);
...
```

Personalization Considerations

When an end-user creates a personalized view of the advanced table region, the Available Columns/Columns Displayed shuttle in the Create/Update/Duplicate page appends the complete hierarchy of columnGroup names, if any are defined as the parent of the column, to the actual column name listed. This ensures that a user hides/shows the correct column, especially in the case where multiple columns of the same name may exist within different columnGroups.

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Row Span

The figure below illustrates a table with row spans. Since an advanced table requires a view usage, and the same view usage for all columns, it is difficult to support row spans that require different amounts of data under each column. In order to implement an advanced table with row spans, you *must* strip the advanced table's dependence on the view object.

The current support of rowspan is ideal only for the static display of a small amount of data. It currently has the following limitations:

- Does not scale when the data *must* come from a view object or if there are a large number of rows or columns.
- Supports read-only data.
- Does *not* support form data because without a view object, it is difficult to push back data.
- Does not support table events.

Note: The advanced table generates an exception in developer mode because the view object is not set on it. However, the actual row span renders when you run the page with the advanced table in normal mode.

Declarative Implementation

This feature must be enabled programmatically.

Runtime Control

In the column where you want to have row span, set the `setUseSeparateRows` attribute on the Column to **true**. On setting this attribute to true, the advanced table does *not* take the view instance (whether it is set declaratively or programmatically) into account. You must therefore provide all the data to the advanced table using UIX `DataObject` and `DataObjectList`, in a row-wise manner, in your controller. The following code example illustrates how you can programmatically implement the row span that is shown in the figure above:

```
// Set a dummy view usage on the table
OAAdvancedTableBean tableBean = ...;
tableBean.setViewUsageName("");

// Set text binding for the leaf items
OAMessageStyledTextBean item1 = (OAMessageStyledTextBean)
  tableBean.findIndexedChildRecursive("item1");
item1.setTextBinding("text1");
OAMessageStyledTextBean item2 = (OAMessageStyledTextBean)
  tableBean.findIndexedChildRecursive("item2");
item2.setTextBinding("text2");

// Set the column bean's properties
OAColumnBean column2 = (OAColumnBean)tableBean.findIndexedChildRecursive
  ("column2");
column2.setUseSeparateRows(true);
UINodeList col2List = new DataObjectListNodeList
  (item2,new DataBoundValue("childDataList"));
column2.setIndexedNodeList(col2List);

// Create the row data
DictionaryData rowData[] = new DictionaryData[3];
rowData[0] = new DictionaryData("text1", "value");
DictionaryData row1Col2Data[] = new DictionaryData[2];
row1Col2Data[0] = new DictionaryData("text2", "value");
row1Col2Data[1] = new DictionaryData("text2", "value");
rowData[0].put("childDataList", new ArrayDataSet(row1Col2Data));

rowData[1] = new DictionaryData("text1", "value");
DictionaryData row2Col2Data[] = new DictionaryData[3];
row2Col2Data[0] = new DictionaryData("text2", "value");
row2Col2Data[1] = new DictionaryData("text2", "value");
```

```
row2Col2Data[2] = new DictionaryData("text2", "value");
rowData[1].put("childDataList", new ArrayDataSet(row2Col2Data));

rowData[2] = new DictionaryData("text1", "value");
DictionaryData row3Col2Data[] = new DictionaryData[1];
row3Col2Data[0] = new DictionaryData("text2", "value");
rowData[2].put("childDataList", new ArrayDataSet(row3Col2Data));

tableBean.setTableData(new ArrayDataSet(rowData));
```

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

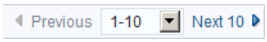
Navigation

The navigation bar allows you to traverse across different row ranges of a table and is rendered:

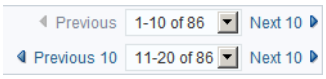
- At the top of the table if the number of rows in the table is less than 10.
- At both the top and bottom of the table if the rows in the table is equal to or greater than 10.

Note: No navigation bar is displayed if the number of rows in the view instance is less than the value specified for the advancedTable `Records Displayed` property.

When a user first navigates to the page, OA Framework does not know how many rows will be returned to the table. The navigation bar simply shows the **Previous** and **Next** links.



Once the user navigates through all the rows, the navigation bar displays the row range as a poplist so that the user can navigate directly to a specific range of rows, as shown below:



Declarative Implementation

There are no declarative steps necessary to implement the default behavior of the navigation bar.

Note: If you have a wide table that requires horizontal scrolling, you can prevent the **Next** and **Previous** links from appearing off the screen by setting the `Width` property of the advancedTable region to **100%**.

Runtime Control

For event handling, specify the following code in your controller's `processFormRequest` to check for a navigation event:

```
if (GOTO_EVENT.equals(pageContext.getParameter(EVENT_PARAM))

{
  ...
}
```

To check whether the **Next** link or **Previous** link is selected:

```
if ((tableBean.getName().equals(SOURCE_PARAM)) &&
  (GOTO_EVENT.equals(pageContext.getParameter(EVENT_PARAM)))
{
  String value = pageContext.getParameter(VALUE_PARAM);

  if (value != null)
  {
    int val = Integer.parseInt(value);
    int newRangeStart = val - 1;
    if (tableVO.getRangeStart() < newRangeStart)
    {
      // next pressed
      ...
    }
    else
    {
      // previous pressed
      ...
    }
  }
}
```

Navigation and Performance

The table rendering logic brings in *only* the rows you need in an incremental fashion. In other words, if your table display size is 10 rows, then only the first 10 rows from the query are brought in to the middle-tier. When you select the **Next** link, another 10 rows are brought in from the database through the JDBC cursor. This is called Incremental Fetch logic. Time spent on network round trips is conserved by bringing in rows only on demand.

This is also the reason why the total row count is unknown when the table is first rendered and the poplist in the table navigation area does not initially appear. Once

the fetch is complete, a poplist always displays even as you navigate back and forth with the **Previous** and **Next** links.

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

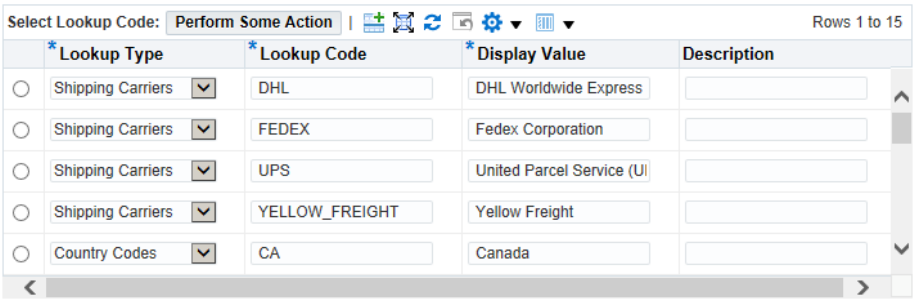
Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Selection and Control Bar

Table selection refers to the ability to select specific rows in a table. If single selection is enabled for a table, OA Framework renders a control bar and a Selection column that displays a radio button.

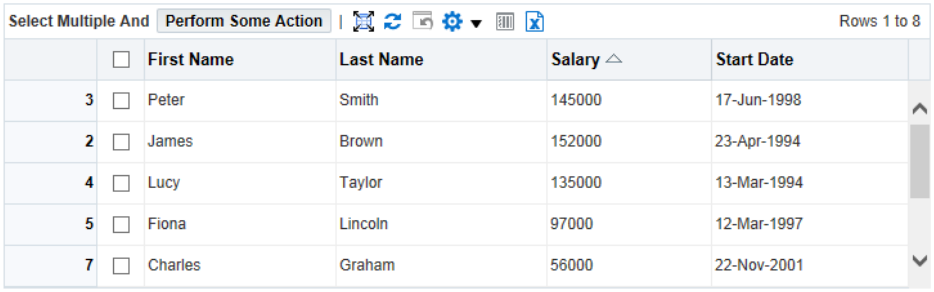
Advanced Table with Single Selection Radio Button



If multiple selection is enabled for a table, OA Framework renders a control bar and a Select column that displays a checkbox. Users can use the Select column to select a specific row or rows and then choose a control such as a button or poplist on the control bar to apply an action on the selected row(s).

Beginning in Release 12.2.6, for multiple selection, a new tri-state check box appears above the Select column to help you quickly select or deselect all or some of your rows as well as visualize the partial selection of your rows. Note that all disabled rows are excluded from selection.

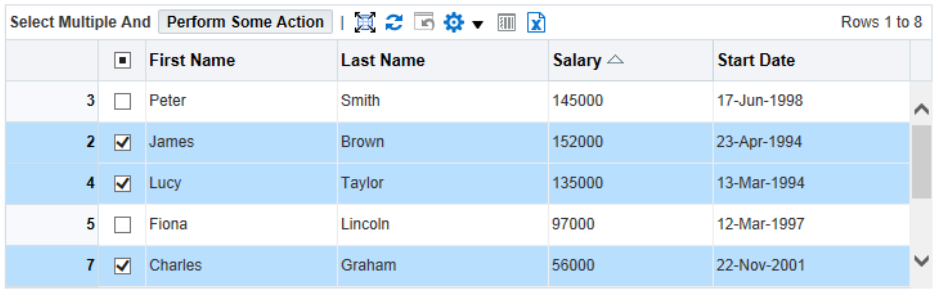
Advanced Table with Multiple Selection Checkbox Where All Rows Are Unselected



Advanced Table with Multiple Selection Checkbox Where All Rows Are Selected



Advanced Table with Multiple Selection Checkbox Where Some Rows Are Selected



Note: Each browser natively renders the partially-selected state of the tri-state checkbox so it's appearance will vary depending on the browser you use.

Accessibility Considerations

Release 12.2.6 supports the single selection radio button and the tri-state multiple selection checkbox in screen readers. The radio button or checkbox can be selected/unselected using the keyboard space bar. However, since the cell above the single selection column is empty and the cell above the multiple selection column displays the tri-state checkbox, there is no visual text to indicate the column scope for the Select columns.

Declarative Implementation

Step 1: To enable table selection, select your advancedTable region in the Structure pane of OA Extension. Display the context menu and under **New**, choose either the singleSelection or multipleSelection named child. If you choose singleSelection, OA Framework renders a Selection column with radio buttons so you can select a single row in the table. If you choose multipleSelection, OA Framework renders a Selection column with checkboxes so you can select multiple rows in the table.

Attention: The Selection column is always rendered as the first column in a table in accordance with the Oracle Browser Look-and-Feel (BLAF) Guideline: Tables. OA Framework does not support any other additional checkbox or radio button item in the table.

Step 2: For the singleSelection or multipleSelection named child, set the following properties:

Properties	Description
`ID`	Specify an identifier that uniquely identifies the item. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
`View Attribute`	Specify a view attribute for this selection named child. The view attribute must be a String, such as "Y" or "N".
`Text`	Set the text that appears in the Control bar. In the examples shown in the figures above, the control bar displays the text "Select Item(s) and ...".

Step 3: Add selection controls to the control bar. In the Structure pane, select the newly created selection named child. Display the context menu and under **New**, choose flowLayout. This creates a flowLayout region that is added as an indexed child of the selection bean.

Step 4: Under the flowLayout region, layout the children, such as buttons, submit buttons and poplists that you want to render in the control bar.

The **Select All** and **Select None** links in the table filter area appear when you choose multipleSelection as the table selection named child.

Runtime Control

OA Framework applies the check/unchecked values of the table selection to the corresponding view attributes. You can use your controller to identify which records are selected and then take programmatic action from there. A code example of how to locate rows with certain attribute values is available in the Iterating View Object Rows section of Chapter 5: View Objects in Detail.

To disable one or more rows from single or multiple selection in your table, add the following code to your controller `processRequest` method:

```
OAAdvancedTableBean tableBean = ...;
tableBean.setSelectionDisabledBindingAttr("Disabled");
```

"Disabled", in the example above, represents a Boolean view attribute that returns 1/"Y" if one or more rows is (are) to be disabled, otherwise it returns 0/"N". The view attribute "Disabled" *must* belong to the same view object as the one associated with the advanced table items.

You can also programmatically set the text in your table control bar. As an example, you can add the following code in your controller `processRequest` method:

```
// Can be OASingleSelectionBean or OAMultipleSelectionBean

OASingleSelectionBean singleSelection = (OASingleSelectionBean)
  advTableBean.getTableSelection();
singleSelection.setText("<newText>");
```

Beginning in Release 12.2.6, OA Framework maintains, in both the server and client side, the user interaction states of the column header tri-state checkbox as well as the individual table row Select checkboxes.

State Transition of the Column Header Tri-state Checkbox

Present State	Event Trigger	New State
SELECTED	click (mouse) space bar (keyboard) tap (touch device)	UN_SELECTED
UN_SELECTED	click (mouse) space bar (keyboard) tap (touch device)	SELECTED
PARTIALLY_SELECTED	click (mouse) space bar (keyboard) tap (touch device)	UN_SELECTED

State Transition of Table Row Checkboxes

- SELECTED - Set when all rows of the table are selected excluding disabled rows.
- UN_SELECTED - Set when all rows of the table are not selected excluding disabled rows.
- PARTIALLY_SELECTED - Set when some rows of the table are selected excluding disabled rows.

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- In accordance with the BLAF guidelines, the selection bean (column) is always rendered with a default column header text of **Select** for the single or multiple selection. This default column header text cannot be changed.
- The **Select All** and **Select None** links in the table filter area that appear when the table selection named child is multipleSelection cannot be changed.
- A table can only have either a single selection (radio group) or a multiple selection (check box group), but never both.
- Further, if a table has either a single selection or multiple selection, then no other column in the table can have a radio button or a checkbox item.
- A table can contain a radio button or checkbox only in the first column, and not in any other column.

Table Actions

Table actions are global table-level action components that are rendered in the control bar of the table. Previously, you had to define tableLayout-rowLayout-cellFormat web beans above the table to include global action items for the table. Now, using the UIX `tableActions` named child, you can define any number of items under a flowLayout or rowLayout. Generally, submitButtons and on rare occasions, poplists, are defined in this area.

The tableAction component is also supported in HGrid and Gantt regions.

Declarative Implementation

Step 1: To define table actions, select your advancedTable region in the Structure pane of OA Extension. Display the context menu and under **New**, choose the tableActions. This automatically creates a tableActions named child consisting of a flowLayout region.

Step 2: Specify a standards-compliant ID for the region and leave the `Region Style` as flowLayout or set it to rowLayout.

Suggestion: If you have only buttons to add to the table actions area, then you can use either layout styles, flowLayout being preferable. However, if you are adding message web beans such as `messageChoice` or `messageTextInput`, along with buttons to the table action area, then use the rowLayout style. Using a flowLayout instead of a rowLayout in this case may cause alignment problems.

Step 3: Under the Layout region, layout the children you want to render as table actions, such submitButton or messageChoice. Select the Layout region, and choose **New > Item** from the context menu. Select the new Item that is created and set the item style as appropriate.

Runtime Control

There is no runtime control code necessary to implement table actions.

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Sorting

An advancedTable column can be sorted by a user at any time, and can also be automatically sorted when the page is initially rendered, (known as initial sorting). In the case of initial sorting, you can sort your table based on one, two or three columns of data. When a column is sortable, the column heading appears beveled, but does not display the sort direction arrow icon until the user selects the column heading to sort the column.

If a user selects a beveled column heading to sort, the sort direction is ascending and displayed as such through the sort direction arrow that appears. If the user selects the sort direction arrow, it toggles to the opposite direction, in this case, sorting the column in descending order.

If a table is initially sorted, the column set as the first level of sorting displays the sort direction arrow icon until the user sorts another column. At this point the sort direction arrow icon appears in that column header.

You can also perform a middle tier sort of an advanced table.

Note: Sorting is case sensitive, where all uppercase values precede lower case values.

Declarative Implementation

To enable sorting on an advancedTable column, you need to set the `Sort Allowed` (mandatory) and `Initial Sort Sequence` (optional) properties on the column's sortableHeader. The following list identifies the possible combination of values you can set on these two properties and how they affect sorting:

Sort Allowed	Initial Sort Sequence	Result
no	none	Column is not sortable and is not initially sorted when the table is rendered.
yes	first,second, or third	The column is sortable and is also initially sorted as either the first, second, or third level of sorting performed on the table, in ascending direction (default) when the table is rendered.
yes	none	The column is available for sorting by the user, but is not initially sorted when the table is rendered.
ascending or descending	none	The column is available for sorting by the user, but is not initially sorted when the table is rendered. This usage should not be used. Instead, set **Sort Allowed** to yes, and **Initial Sort Sequence** to none to achieve the same result.
ascending or descending	first, second, or third	The column is sortable by user in the specified direction and is also initially sorted as either the first, second, or third level of sorting performed on the table.

Enabling Internal Sorting On Another View Attribute

Generally, you can specify sorting on a table when you define the table. In some cases, especially when you use a raw HTML column with a HTML link, you may want the sorting to occur on the link value displayed in the user interface rather than on the actual link value itself (with <a href ...). To accomplish this, set the `Sort By View Attribute` property on the column container sortableHeader, to the name of the view object attribute on which you want to sort.

If this property is not set or set to null, sorting is performed on the view attribute name associated with the column child.

Attention: If a table column contains a nested region, such as a switcher region, and you want to enable sorting on this column, it is mandatory that you set the Sort By View Attribute property on the column container's sortableHeader.

Enabling Middle Tier Sorting

To enable middle tier sort on the Advanced table, set the Dirty Row Sort Enabled property to true. The following table shows the possible outcomes of the `DirtyRowSort` Enabled property on the Advanced table.

Table State	DirtyRowSortEnabled(**true**)		DirtyRowSortEnabled(**false**)	
	Fetch Complete	*Fetch Incomplete*	*Fetch Complete*	*Fetch Incomplete*
Table with no pending changes	Sort in the middle-tier	Sort in the **database**	Sort in the **middle-tier**	Sort in the **database**
Table with pending changes	Sort in the middle-tier	No sort is done and a warning message is displayed	No sort is done and a warning message is displayed	No sort is done and a warning message is displayed
Table with transient EO/VO columns	Sort in the middle-tier	No sort is done and a warning message is displayed	No sort is done and a warning message is displayed	No sort is done and a warning message is displayed

Note: When the data is sorted in the middle-tier, the data is sorted based on unicode character values and does not support any other character set. Hence middle-tier sorting is not supported for any language other than English.

Runtime Control

The table bean invokes the following two methods on the view object to perform sorting:

- `viewObject.setOrderByClause(orderByClause);`
- `viewObject.executeQuery();`

Debugging can be performed by setting a break on these methods on the view object, or a superclass of the view object.

Programmatically make a column sortable by using the `isSortable` and `setSortable` methods on `oracle.apps.fnd.framework.webui.beans.OAWebBeanDataAttribute`. Refer to the following code as an example:

```
// We assume that all children of the advanced table are those that implement
// OAWebBeanDataAttribute
// Alternately, the exact bean type can be used, such
// as OAMessageStyledTextBean.
OAWebBeanDataAttribute webBean =
  (OAWebBeanDataAttribute)advtableBean.findIndexedChildRecursive("<columnName>");
webBean.setSortable(true);
```

Enabling Internal Sorting On Another View Attribute

You can programmatically enable internal sorting on another view attribute.

For example, you have a view attribute named Emplink. When you click on the sort icon of the table column for Emplink, you want the actual internal sorting to occur on the Ename attribute. Programmatically control this as follows:

```
OAAdvancedTableBean advtable = ()createWebBean(pageContext,  webBean, "EmpTable");
OAAdvancedTableBean
OAWebBeanDataAttribute empno =
  (OAWebBeanDataAttribute)advtable.findChildRecursive("<EmpnoItemID>");

empno.setSortByAttributeName("Ename");
```

If this sort by attribute is not set or set to null, sorting is performed on the view attribute name associated with the column child.

Enabling Sorting By View Attribute On A Column sortableHeader

As of Release 12.1 RUP4, you can use the following API on OASortableHeaderBean to programmatically enable sorting by view attribute on a column sortableHeader:

```
public void setSortByViewAttributeName(String viewAttrName)
```

The getter method for this is:

```
public String getSortByViewAttributeName()
```

Middle Tier Sorting

To enable middle tier sorting, call the following method on the OAAdvancedTableBean.

```
tableBean.setDirtyRowSortEnabled(true);
```

Implementing Table Selection and Hide/Show

By default, when the selector item is checked in a table that supports table selection, the underlying view object attribute value is updated. This action leaves the view object with pending changes (even for a view-only table). When the view object is in this "dirty" state, operations such as table sorting or personalization are disabled. To implement a selector item without affecting the view object state, follow these steps:

Step 1: Define a transient view attribute for the selector item using the BC4J view object wizard in JDeveloper. Do *not* add a dynamic attribute by calling `ViewObject.addDynamicAttribute`.

Step 2: Following the OA Framework coding standards, always generate an `OAViewRowImpl` for each view object so you can call named accessors. In this case, override your `set<AttributeName>` method as shown below:

```
public void setSelectFlag(String val)
{
  populateAttribute(SELECTFLAG, val);
}
```

This example assumes the selector's view attribute is named "SelectFlag" and the code is calling the `populateAttribute` method to set its value without marking the view object as being "dirty."

A similar situation occurs when Hide/Show is present in a table. See Entity Object and View Object Setters in Chapter 5 for additional information.

Personalization Considerations

If you expect your users to personalize the sort order of a table, do *not* set explicit order by statements in your view objects. Instead, use the `Initial Sort Sequence` and `Sort Allowed` properties in OA Extension. When the table is rendered the first time, OA Framework provides an order by statement on the view object based on how you set these properties. For more information on how to personalize the sorting of data in a table, refer to Admin-Level Personalizations in the Personalization Guide.

When you define an end-user personalizable page, where the advanced table under the query region contains a column with a nested region (such as a table content switcher), users will be able to sort on that column in the Create/Update/Duplicate View page. Previously, users were unable to set a column with a nested region as a sort column in the **Sort Settings** because that column was not exposed in the **Column Name** poplist.

Notes:

- When a user specifies the column with the nested region as the column to sort, the actual content that is sorted will depend on what you set the Sort By View Attribute property to on the sortableHeader for the column container.
- If a user sets sorting on a column that contains a nested region using the Create View page, but finds that the change is not taking effect on the table, it is likely that the `Sort By View Attribute` property on the sortableHeader for that column was *not* set. In this case, the personalization administrator *must* set the `Sort By View Attribute` property on that column using the Admin Personalization UI.
- If you do not want sorting on the nested region to be modified by users, set the `User Personalization` property to **False** so that the nested region does not display in the **Column Name** poplist of the Create View page's **Sort Settings**.

See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

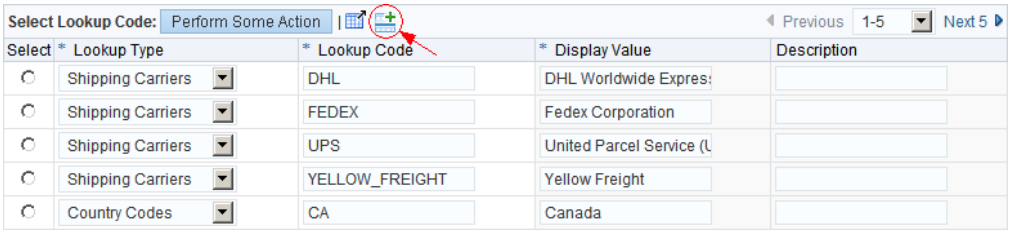
Usage Restrictions

- Performed by re-querying the database.
- May not work well with view objects that contain custom 'expert-mode' SQL. Basically the view object `setOrderByClause` is invoked on the view object using the column name associated with this web bean. An alternative for 'expert-mode' SQL may involve overriding the view object `setOrderByClause` and performing custom logic. The `orderByParameter` contains the column name and either **desc** or **asc**.
- Does not work with view objects that contain the view attribute expression "See the SQL...". To sort on these view attributes, modify the view object XML directly and change the view attribute expression to the SQL column name.
- Not allowed for tables that allow inserts.
- Not supported for tables containing updateable columns unless the updateable columns are mapped to transient view object columns. No exception is thrown if the table contains updateable columns since there may be a rare case when it makes sense, as in a table where the contents fit on one page.
- Modified transient columns are reset. This is normal and expected behavior.
- Not supported on the Select column, which is the optional first column of a table that contains a checkbox or radio button.
- Disabled when the Select column is checked for a row in a table or when **Hide/Show** is present in a table. See Implementing Table Selection and Hide/Show for more information.
- Not supported on table rows created in the middle-tier.

Adding Rows

OA Framework displays an **Add Another Row** button to the table footer if a table is insertable, as described in the Oracle Browser Look-and-Feel (BLAF) Guidelines: Tables (for Release 12.1.3 and earlier). Refer to the Advanced Table example of the Sample Library for an overview of an implementation of this feature.

Note: As of Release 12.2.3, instead of an **Add Another Row** button appearing at the table footer, an **Add Another Row** icon now appears on the table's control bar as shown in the figure below. The button label that you can customize for the previous Swan Look and Feel now appears as a tool tip for the icon.



Note: Also in Release 12.2.3, if you implement the table to add more than one row, then an **Add Multiple Rows** icon appears in place of the **Add Another Row** icon in the control bar.

Note: When you add another row, the row is added as the last row in the current range. The existing last row in the current range is pushed into the next range.

Note: When you select the Add Another Row button/icon, the cursor is placed on the first updateable field of the new row that was added. If you implement an **Add *n* Rows** button/icon, then when you select this button/icon, the cursor is placed on the first updateable field in the first of the new set of n rows added.

Add new rows to a detail table that is associated to some master row via a detail view instance. In the past, you had to `setAutoInsertion(false)` on such a table and handle row insertions yourself. Now, you can add rows automatically to detail tables because foreign keys are automatically populated for a table associated

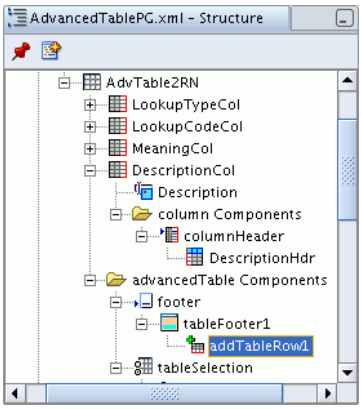
with a detail view instance of some master row. You *must* still `setAutoInsertion(false)` if there is custom SQL in the view link or if you have to perform some custom population of detail row data.

Declarative Implementation

The following steps describe how to display an **Add Another Row** button/icon to an advanced table.

Step 1: In the Structure pane of OA Extension, select your advancedTable region and choose **New > footer** from the context menu. OA Extension creates an advancedTables Components folder containing a footer named child, that contains a tablefooter container (labeled tableFooter1).

Step 2: Select the tableFooter and choose **New > addTableRow** from the context menu.



Step 3: In the Structure pane, select the newly created addTableRow item, as shown in the figure above, and use the Property Inspector to set its following properties (* Required):

Properties	Description
`ID`*	Specify an identifier that uniquely identifies the addTableRow item in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
`Add Rows Label`	For Release 12.1.3 and earlier, specify text to override the "Add Another Row" default text that appears in the **Add Another Row** button. For Release 12.2.3 and above, this text appears as a tool tip for the the **Add Another Row** icon.
`Rows to Add`	Specify the number of rows to add each time a user chooses the **Add Another Row** button. Do not set the value for this property to be greater than the value set for the `Records Displayed` property on the advanced table. The default is 1. In Release 12.2.3, if you set the value greater than 1, the **Add Multiple Rows** icon renders, otherwise, the **Add Another Row** icon renders. **Note:** This property is valid only if the `Insert Rows Automatically` property is set to **True**.
`Insert Rows Automatically`	Specify **True** to indicate rows are to be added automatically when the **Add Another Row** button is chosen. Specify **False** if you want to respond to the event in your controller so that you can, for example, set the primary keys or set some default row attributes. **Note:** The `Insert Rows Automatically` property is equivalent to the `autoInsertion` property that was set in your controller using the `setAutoInsertion` API for classic tables.

Step 4: To allow new rows to be added to a detail table, such as with an Advanced Table-in-Advanced Table, Detail Disclosure drilldown, or a master-detail template, then in addition to setting `Insert Rows Automatically` to **True**, you *must* also attach the master and detail tables together by specifying an identical value in the `View Link Name` property of both the master and detail tables.

Note: Auto-insertion in detail tables only work for cases where the view link has no custom SQL.

Runtime Control

There is no runtime control code necessary to implement the **Add Another Row** button/icon. However, to do special handling when a user chooses the Add Another Row button/icon, you *must* first declaratively set the `Insert Rows Automatically` property to **False** for the addTableRow item. This is performed in the `processFormRequest` method of your controller code.

You can also modify properties of the Add Another Row feature in your controller, as shown in the following example:

```
// Get a handle to the table footer bean
OATableBean tableBean = ...;
OATableFooterBean tableFooterBean = tableBean.getFooter();
if (tableFooterBean != null)
{
  // Get a handle to the add table row bean
  OAAddTableRowBean addTableRowBean = tableFooterBean.findIndexedChild("<addRowBeanId>");

  // Set the NEW_ROW_INITIALIZED property on the addTableRowBean to ensure that
  // the state of the new rows added by OAFramework (Insert Rows Automatically set to TRUE)
  // is STATUS_INITIALIZED. This would ensure that the rows that are not edited
  // by the user are not posted to the database on a commit.
  addTableRowBean.setAttributeValue(OATableUtils.NEW_ROW_INITIALIZED, Boolean.TRUE);

  // Add 5 rows everytime add row button is clicked
  addTableRow.setText("Add 5 Rows");
```

```
  addTableRow.setRows(5);

  // Handle add row insertion yourself
  addTableRow.setAttributeValue(AUTO_INSERTION, Boolean.FALSE);
}
```

Note: If you add multiple rows to a table, the new rows are shown at the bottom of the current range, pushing any existing rows into the top of the subsequent range. Only add, at most, a number of rows that is less than or equal to the number of rows displayed in your table.

Important: If a product team decides to handle the Add Rows event in its controller, the product team needs to get the actual number of records to be inserted from the Add Row web bean (which includes the value of the Admin personalizable `Rows to Add` property merged with the seeded meta data value), and insert that many number of rows into the view object. If the product team code functionality is such that it cannot use the personalizable `Rows to Add` property, the product team must provide a warning or exception that the personalization to the `Rows to Add` property does not effect the page and instruct the personalization administrator to revert the `Rows to Add` property back to its original value. If, however, OA Framework handles the Add Rows event, OA Framework gets the value of the `Rows to Add` property from the web bean (which includes the value of the Admin personalizable `Rows to Add` property merged with the seeded meta data value), and adds that many rows into the view object.

Add Another Row and View Object Execution

If the view object associated with the advanced table is *not* executed and you select **Add Another Row**, you get a state loss error on the page. For any successful table event, the table view object query *must* be executed before a table event occurs. In the case where you do not want to display any rows when the page renders, yet want to let the user select **Add Another Row** to add rows into the table, then before you add rows to the table, you must properly initialize the view object as described in the Initialization Guidelines.

Empty Rows In A Table

Multiple empty rows can be added into a table when any of the following occur:

- Your code inserts rows into the view object before the table renders, so the table displays new blank rows.
- A user selects the **Add Another Row** button multiple times.
- A user selects the **Add n Rows** button, which you have programmatically implemented to add 'n' rows into your view object.

In any of these cases, if the user does *not* enter values into some of the rows, you *must* include logic to recognize those empty rows (if any), and remove them from the view object. For example, if you pre-populate ten new empty rows in an expense reports table and the user enters only three expense lines, you *must* provide code to recognize and remove the remaining seven empty rows from the view object.

Personalization Consideration

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

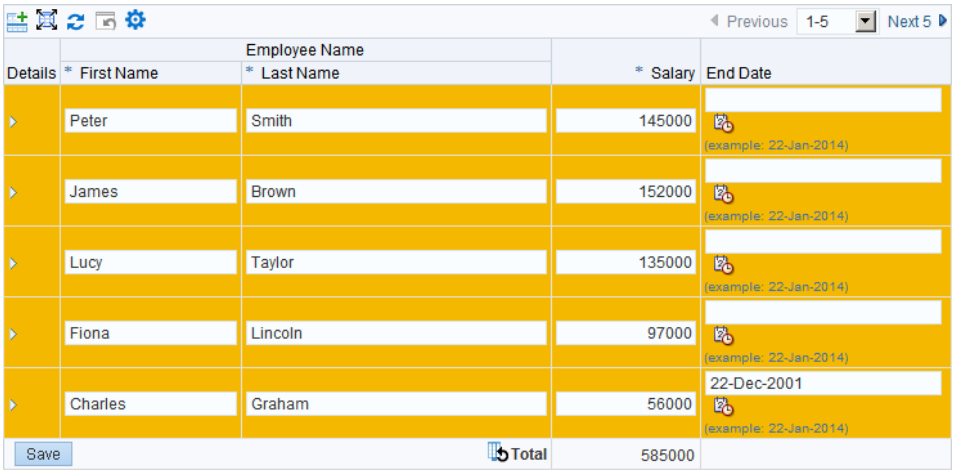
Totaling

OA Framework supports the Totaling of values in any numeric table column. If Totaling is enabled in a table, the table footer displays a column total and a **Recalculate** button, as shown below. The total displays a double precision summation of all visible rows in the table.

Warning: The total reflects only the current visible records and not all the records queried.

Note: Totaling can be enabled for any column except for the first column in a table.

Note: As part of the Release 12.2.4 Look and Feel, the **Recalculate** button appears as an icon next to the **Total** text. The button label that you can customize for the Swan Look and Feel now appears as a tool tip for the icon.



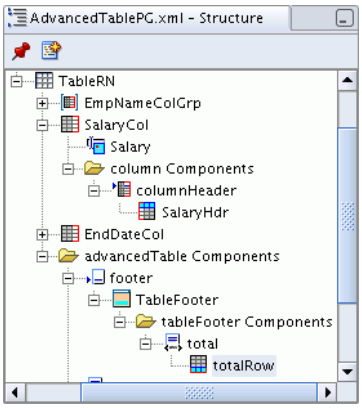
Declarative Implementation

The following steps describe how to enable Totaling for a column in the footer of an advanced table.

Step 1: In the Structure pane of OA Extension, select the column container for which you want to enable Totaling Any column except for the first column can be totaled. Set the `Total Value` property for this column container to **True.**

Step 2: In the Structure pane, select your advancedTable region and choose **New > footer** from the context menu. OA Extension creates an advancedTables Components folder containing a footer named child, that contains a tablefooter container (labeled tableFooter1).

Step 3: Select the tableFooter container and choose **New > total** from the context menu. OA Extension creates a tableFooter Components folder containing a total named child, that contains a new totalRow item as shown in the figure below.



Step 4: In the Structure pane, select the totalRow item and use the Property Inspector to set its following properties (* Required):

Properties	Description
`ID`*	Specify an identifier that uniquely identifies the totalRow item in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
`Recalculate Totals Label`	In Release 12.1.3 and earlier, specify text to override the "Recalculate" default text that appears in the **Recalculate** button. In Release 12.2.3 and above, the specified text appears as a tool tip for the **Recalculate** icon.
`Disable Total Recalculation`	Specify **True** or **False** to indicate whether or not to render the **Recalculate** button/icon. The default is **False**.

Note: Although the `Rendered` property on the totalRow item may be changed, always set this property to **True** so you can render and see the total. If you do *not* want the `totalRow` web bean to render, then do not set the `Total Value` property to **True** for any of the columns in your advanced table.

Runtime Control

There is no runtime control code necessary to implement Totaling in the advanced table footer. However, you can use programmatic control to accomplish some additional formatting or modifications to the Totaling feature.

Modify Properties of Totaling

Modify properties of the Totaling feature in your controller, as shown in the following example:

```
// Get a handle to the table footer bean
OATableBean tableBean = ...;
OATableFooterBean tableFooterBean = tableBean.getFooter();
if (tableFooterBean != null)
{

  // Get a handle to the total row bean
  OATotalRowBean totalRowBean = tableFooterBean.getTotal();

  // If you want to set text for the total button/icon
  totalRowBean.setText("Re-Total");

  // If you want to hide the recalculate total button/icon
  totalRowBean.setReadOnly(true);
}
```

Format totalRow Item

You can format the content of the totalRow item programmatically. However, setting a formatting option such as CURRENCY_CODE, on an individual web bean's text attribute, under the total column, does *not* format the total row's values. Instead, to format the total row's value, use the `setAttributeValue(AttributeKey key, Object value)` method of `oracle.apps.fnd.framework.webui.beans.table.OAColumnBean` in your controller, as shown in the following example:

```
OAColumnBean columnBean = (OAColumnBean)webBean.findIndexedChildRecursive("Salary");
columnBean.setAttributeValue(CURRENCY_CODE,"USD");
```

User should also be able to format Total column values in Table using formatter like:

```
oracle.cabu.ui.validate.Formatter formatter = new OADecimalValidater("###0.###;-###0.###","#.##0.###;-#,##0.###");
OAColumnBean columnBean = (OAColumnBean)webBean.findIndexedChildRecursive("Salary");
columnbean.setAttributeValue(ON_SUBMIT_VALIDATER_ATTR, formatter);
```

Control Total Value Generation

To bypass OA Framework's mechanism to control the value generation for a table column footer so you can provide your own value such as a percentage, set the `oracle.apps.fnd.framework.webui.OAWebBeanConstants TABULAR_FUNCTION_VALUE_ATTR` on the appropriate table column. This overrides OA

Framework's internal behavior of footer value generation. Like all web bean attributes, you can data bound this attribute if required.

Sample usage:

```
OAAdvancedTableBean advtableBean = ...;
OAMessageStyledTextBean salaryBean = advtableBean.findChildRecursive(...);

String formattedTotal = ...;
// compute the total
salaryBean.setAttributeValue(TABULAR_FUNCTION_VALUE_ATTR,formattedTotal);
```

or:

```
salaryBean.setAttributeValue(TABULAR_FUNCTION_VALUE_ATTR,
  new OADataBoundValueViewObject(...));
```

Grand Totals

Declarative or programmatic support for Grand Totals is currently not available in advanced tables. To implement grand totals, you *must* build your own layout below the advanced table in accordance with the Oracle Browser Look-and-Feel (BLAF) Guidelines: Tables: Grand Totals. Use the messageComponentLayout region style to build this layout. In addition, you *must* programmatically provide the values for the grand total fields. Since these fields are like any other form fields, you can first calculate the grand total value in your controller code by looping through the entire view instance row range, and then call `setText()` for the grand total field to set it with the calculated grand total value.

Personalization Consideration

As mentioned earlier, you can not enable Totaling for the first column in a table. If you attempt to personalize the first column of an advanced table by setting the `Total Value` property to **True** in the Personalize page, it is simply ignored. To total the contents of that first column, you *must* use the OA Personalization Framework to reorder the column within the advanced table so that it is no longer the first column in the table, then set the `Total Value` property to **True** for that column.

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Totaling is supported *only* on leaf items of `Data Type` **NUMBER** immediately under a column container of an advanced table. It is *not* supported on regions, such as a messageComponentLayout region, under a column container of an advanced table.

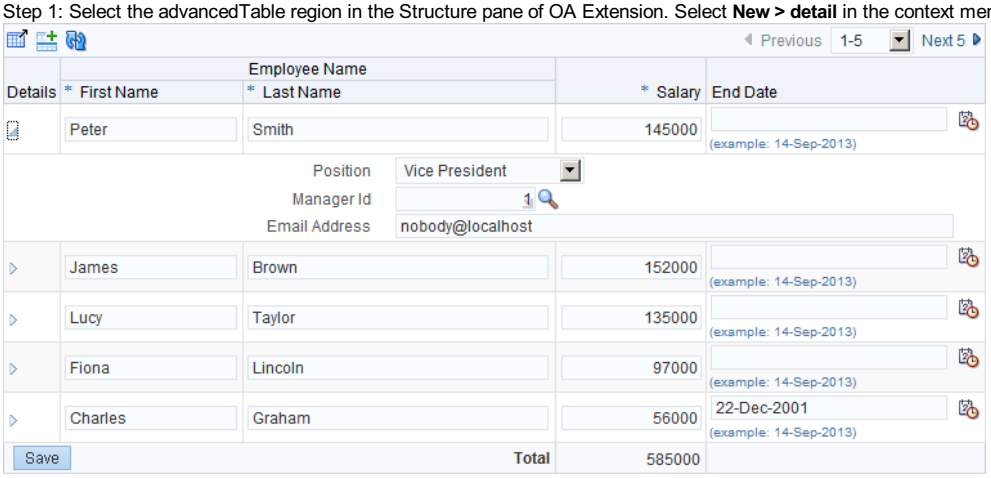
Detail Disclosure

Detail disclosure is also known as a row-level Hide/Show in a table.

Declarative Implementation

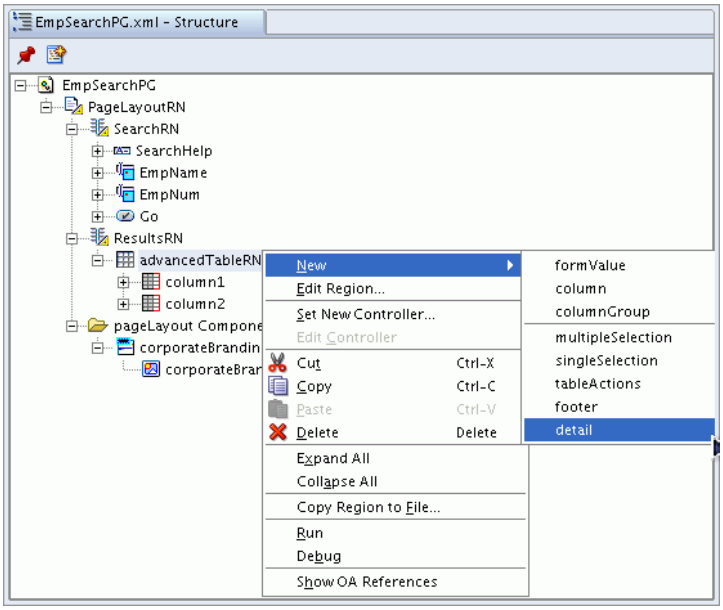
Follow the steps given below to implement the detail disclosure feature. Refer to the Advanced Table example of the Sample Library for an example of this feature.

Step 1: Select the advancedTable region in the Structure pane of OA Extension. Select **New > detail** in the context menu. This creates

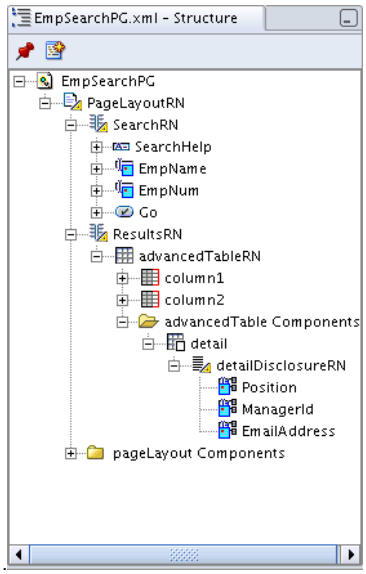


a detail named child that you can use to show additional information for any row in the table.

Step 2: The default region style of the detail region is **header**, but you can change the style, if required, and add children to the region. The recommended layout is messageComponentLayout.



Step 3: Select the advancedTable region in the Structure pane. Set the `Detail View Attribute` property to the name of the Boolean/String ("Y" or "N") attribute from the view object specified for the advancedTable region. This determines the shown or hidden state of the detail child of the rows



Note: Specifying the named child detail region alone is sufficient to enable the addition of a row-level Hide/Show. Therefore, you do *not* need to explicitly add a hideShow as the detail region.

Attention: The leaf items in the detail disclosure named child region *must* be associated with the same view object as the leaf items in the other columns of the advanced table except when your detail disclosure named child region contains another advanced table (Advanced Table-in-Advanced Table). In this case the outer and inner tables *must* be associated with master and detail view objects connected via a view link.

Runtime Control

Set the properties discussed in the Declarative Implementation section above in your controller:

```
import oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean;
...
OAAdvancedTableBean advtableBean =
  (OAAdvancedTableBean)webBean.findIndexedChildRecursive("<advtableBeanName>");

// Set the detail named child
OAWebBean detailNode = (OAWebBean)...

// Get a handle to the detail node
advtableBean.setDetail(detailNode);

// Set the detail view attribute name
advtableBean.setDetailViewAttributeName("<detailViewAttributeName>");
...
```

The use of the **Hide All Details|Show All Details** link above a table with Detail Disclosure feature can only be enabled by using the UIX `TableBean`

`setAllDetailsEnabled` method in your `processRequest` method.

```
// Turn on the "Hide All Details | Show All Details" links
advtableBean.setAllDetailsEnabled(true);
...
```

For information on nested tables or table inside detail disclosure, refer to table-in-table.

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

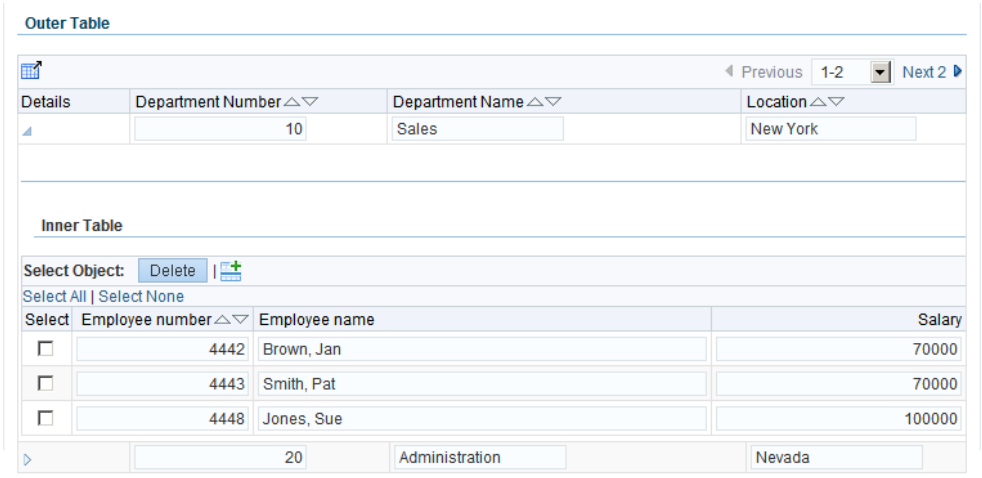
Usage Restrictions

- The BLAF Hide/Show guidelines for Tables do *not* allow embedding secondary or children objects in the details of a table row. In other words, do *not* create a hideShow or hideShowHeader region under the detail region of a table. See HGrids for information about how to display hierarchical information in a table, Similarly, do *not* create a hideShow region under an inner table of a Table-in-Table or implement detail disclosure inside the inner table of a Table-in-Table.
- In accordance with BLAF guidelines, the **Detail** column header text for the detail disclosure column and the **Show/Hide** link texts within the cells of the detail disclosure column can not be modified.
- The `All Details Enabled` property for the advanced table renders the **Hide All Details|Show All Detail** link above the advanced table. Selecting **Show All Details** expands all the detail disclosure regions and selecting **Hide All Details** collapses all the detail disclosure regions. To enable this property, you must do so programmatically using the `setAllDetailsEnabled` method. Refer to the Runtime Control section for an example.
- The **Hide All Details|Show All Detail** link text cannot be changed.

Advanced Table-in-Advanced Table

If you have number of records that are interrelated, you can display them in tabular form using an `advancedTable` bean. This is an intuitive way to display data that is independent of other entities. Advanced Table-in-Advanced Table allows all interrelated information, typically master-detail relationships, to be seen and updated on one page by embedding an inner table within an outer table. This is possible because you can add an `advancedTable` bean as a named child of another `advancedTable` bean. The information in the inner table is based on the corresponding master row in the outer table. The inner table associated with each row is opened/closed using the **Hide/Show** links that appear in the Details column as shown in the sample table-in-table user interface below.

Note: Advanced tables *only* support other advanced tables as the inner or outer table.



The inner table bears a master-detail relation with the outer table. For each master row, corresponding detail rows are displayed in an inner table. Users can control the rows in the inner table by defining additional search criteria to the criteria already defined in the view link, which relates the outer view object with the inner view object.

Warning: A table-in-table user interface can be costly in terms of performance because multiple view objects and the bean hierarchy are created and replicated at runtime. Performance cost is reduced when the inner table is read-only because the same view object can be reused.

The creation of the view object attached to a detailed section is done during rendering time. If you never open the detailed section of a row, the view object is *not* created. Therefore, if you have ten master rows in the outer table and you open the detail section of only two rows, the performance cost is only for those two rows.

You can make the inner table editable by adding form-elements in the inner table so the data is pushed to the view object just like a regular table. The inner table also supports navigation, sorting, adding another row, Totaling, single and multiple selection, row-level and column-level banding and exporting to a Microsoft Excel spreadsheet.

Declarative Implementation

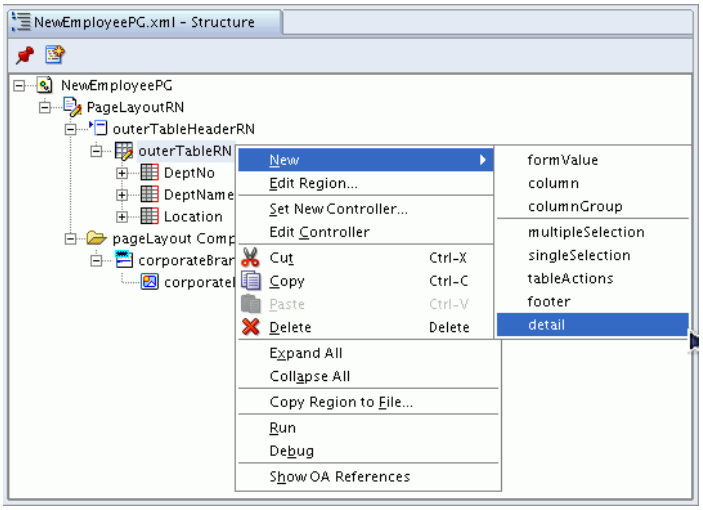
Note: The BLAF Hide/Show guidelines for Tables do *not* allow embedding secondary or children objects in the details of a table row. In other words, do *not* create a hideShow or hideShowHeader region under the detail region of a table. See HGrids for information about how to display hierarchical information in a table. Similarly, do *not* create a hideShow region under an inner table of an Advanced Table-in-Advanced Table or implement detail disclosure inside the inner table of a Advanced Table-in-Advanced Table.

To create a table similar to the one shown above, define the region definition in XML using OA Extension and add code in your controller.

Xml Structure

Step 1: Create an advancedTable region.

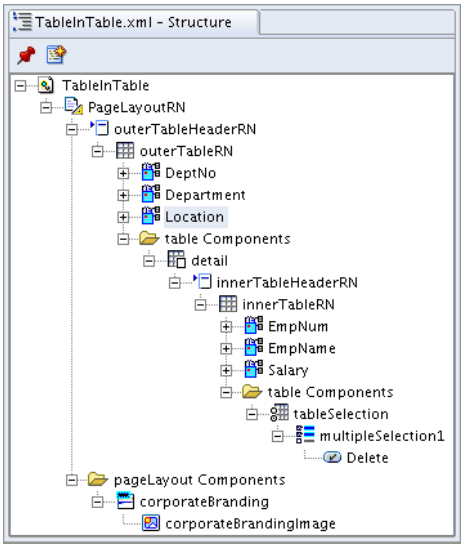
Step 2: Select the (outer) advancedTable region in the Structure pane of OA Extension. Select **New > detail** from the context menu.



This creates a detail named child that you can use to show additional information for any row in the table. The default style of the detail region created is **header**. Set this detail region to be the header for your inner table. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an `ID`.

Step 3: Select the inner table header region, select **New > Region** from the context menu to create an advancedTable region that is to become your inner table.

Step 4: Follow the standard instructions for defining an advanced table to create your inner table.



Step 5: Select the (outer) advancedTable region in the Structure pane. Set the `Detail View Attribute` property for this region to the name of the Boolean/String ("Y" or "N") attribute of the same view object as that of the other table columns. This determines the shown or hidden state of the detail child of each row.

Runtime Control

After creating the XML page structure, you need to initialize some properties on the beans in your code.

As a example, suppose your BC4J setup is as follows:

- View object 1 = DeptVO.
- Primary Key of DeptVO = Deptno.
- View object 2 = EmpVO.
- DeptVO and EmpVO are connected by a view link called DeptEmpVL.
- DeptVO and EmpVO are connected by the query *WHERE DEPTNO = :1*.
- Application module contains the two view objects and the view link.

Note: The view link is used in the master-detail relationship to display the correct detail rowset.

To achieve the advanced table-in-advanced table display, add these lines in your controller:

```
public void processRequest(...)
{
  OAWebBean outerTable = (OAWebBean)webBean.findChildRecursive("outerTable");
  OAWebBean innerTable = (OAWebBean)webBean.findChildRecursive("innerTable");
  if (outerTable != null)
  {
    outerTable.setAttributeValue(CHILD_VIEW_ATTRIBUTE_NAME,"Deptno");
    outerTable.setAttributeValue(VIEW_LINK_NAME,"DeptEmpVL");
  }
  if (innerTable != null)
```

```
  {
    innerTable.setAttributeValue(CHILD_VIEW_ATTRIBUTE_NAME,"Deptno");
    innerTable.setAttributeValue(VIEW_LINK_NAME,"DeptEmpVL");
  }
  ...
  ...
}
```

Note: The two properties that are set in the example above are required for the proper functioning of the bean ID generation and master-detail relationship. If you omit any of the above, you may end up with a JavaScript error or see incorrect records in the inner table.

RowSets

The advanced table-in-advanced table detail template (inner table) uses the `RowSets` interface to interact with the OA Framework Model. When you query data using a view object, the results of the query are stored in a RowSet. Each RowSet can have multiple iterators, which allow scrolling through the set of rows. Each view object can have multiple RowSets.

As a result of the `RowSet` implementation, to handle rows manually in an advanced table-in-advanced table, you need to go about it differently than when you handle rows in just an advanced table. Generally, with a advanced table, you can get a handle to the advanced table view object and manipulate it, such as by adding or deleting a row, and see the change reflected on the UI. This is *not* the case for the inner table of an advanced table-in-advanced table. Any changes that you make to the detail view object is never even considered.

To insert a new row or update the state of an existing row in an inner table, you *must* create an `OAInnerDataObjectEnumerator` and use it to get a handle to the RowSet attached to the inner table. Once you have the RowSet, you can manipulate it in any manner and the changes are reflected in the UI. In previous versions of OA Framework, `oracle.apps.fnd.framework.webui.OAInnerDataObjectEnumerator` used to scroll through the different view objects associated with each inner table. Now, since a RowSet is associated with each inner table, the behavior of this enumerator has been updated. Following is an example of how to use the enumerator to go through the rows of a RowSet attached to an inner table:

```
// get a handle to inner table
OATableBean innerTable =
  (OATableBean)webBean.findChildRecursive("InnerTableBean");

// create an enumerator
OAInnerDataObjectEnumerator enum =
  new OAInnerDataObjectEnumerator(pageContext, innerTable);

while (enum.hasMoreElements())
{
  RowSet innerRowSet = (RowSet) enum.nextElement();

  // get all rows
  Row []rowsInRange = innerRowSet.getAllRowsInRange();
  for (int i = 0; i < rowsInRange.length; i++)
  {
    Row nextRow = (Row) rowsInRange[i];

    if ("Y".equals(nextRow.getAttribute("DeleteFlag")))
    {
      // delete the marked row
      nextRow.remove();
    }
  }

  // In case you want to add new rows in this RowSet, you can do the same
  OARow newRow = (OARow) innerRowSet.createRow();

  // initialize value for some attribute and insert the row
  newRow.setAttribute("SomeAttr", "SomeValue");
  innerRowSet.insertRow(newRow);

}

// In case you want to change the WhereClause of the containing view object
OAViewObject innerViewObject = (OAViewObject) innerRowSet.getViewObject();

String newWhereClause = "DEPT.LOC = :1";
innerViewObject.setWhereClause(newWhereClause);

// Each RowSet can now bind parameter specific to each inner web bean
```

The association of a RowSet instead of a view object to the inner table improves the performance of a advanced table-in-advanced table. The amount of memory and other JDBC resources required is limited because all RowSets of an inner table are part of the same view object, and therefore share the same query. Also, the implementation of an advanced table-in-advanced table is now consistent with other web beans of similar hierarchical nature. The master table is always in synchronization with the inner table because any change in the master table triggers changes in the inner table via a view link.

Note: If you use the `OAInnerDataObjectEnumerator` in the `processRequest` method, you may not be able to scroll through the RowSets because they may not yet be created. A RowSet for an inner table is created only when a user actually displays the inner table in the UI.

If you want to iterate through all fetched inner rows instead of the rows in the current range, create OAInnerDataObjectEnumerator in the following way:

```
OAInnerDataObjectEnumerator enum =
      new OAInnerDataObjectEnumerator(pageContext, innerTable, true);
```

The third argument indicates that the enumerator should go through all fetched rows. The default behavior is to fetch the rows in the current range.

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- The BLAF Hide/Show guidelines for Tables do *not* allow embedding secondary or children objects in the details of a table row. In other words, do *not* create a hideShow or hideShowHeader region under the detail region of an advanced table. See HGrids for information about how to display hierarchical information in a table. Similarly, do *not* create a hideShow region under an inner table of a Advanced Table-in-Advanced Table or implement detail disclosure inside the inner table of a Advanced Table-in-Advanced Table.
- The inner table does not support LOVs. You must use an alternative UI.

Formatting a Table

There are several ways to format an advanced table. They are as follows:

- Full Table Formatting - affects the table as a whole.
- Column Formatting - affects only the columns of the table.
- Row Formatting - affects only the rows of the table.
- Column Header/Row Header Formatting - affects only the column headers or row headers of the table.

Refer to the section of interest for additional information.

Full Table Formatting

There are two types of formatting that you can apply, which affect the table as a whole:

- Banding: In a table, no banding, (no alternating colors), is rendered in the columns and rows of a table by default. The figure below illustrates a table with horizontal banding and vertical and horizontal grids, which appear to the left of each column, and above each row.
- Setting the width of the table.

Column 1	Column 2	Column 3
cell	cell	cell
cell	cell	cell
cell	cell	cell
cell	cell	cell
cell	cell	cell

Refer to the Advanced Table example in the Sample Library to see how row banding is implemented.

Declarative Implementation

Change the banding and width of a table by setting the `Banding Interval`, `Banding Type`, and `Width` properties of the advancedTable region in OA Extension, as described in Step 2 of Defining an Advanced Table.

For row or column banding, set `Banding Type` to rowBanding or columnBanding, respectively. Use the `Banding Interval` property to specify the interval of banding you want to render. For example, set Banding Type to columnBanding and Banding Interval to 2, to get a table that alternates two dark columns with two light columns over the entire table.

Although the width can be specified by pixels or percentages of the width of the parent element, percentages are commonly preferred. If you set the `Width` to 100%, the **Next** and **Previous** links that allow you to navigate among the rows in a table, remain visible above the table, without horizontal scrolling. This is especially useful when you have a wide table that requires horizontal scrolling.

Note: No matter what value you specify for the table width, if the width is not large enough to accommodate the table content, the value is overridden at display time to take up the minimum amount of necessary space.

Runtime Control

No runtime code is required to format a table.

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Column Formatting

You can modify the formatting of a table's columns as follows:

- Turn on/off the line wrap within the cells of a column.
- Alter the alignment of a column's content.
- Turn on/off the display of the grid line to the left of a column.
- Alter the width of a column.
- Set the column banding shade to dark or light.

Declarative Implementation

You can change the formatting of a column by setting the `No Wrap`, `Alignment`, `Grid Displayed`, `Banding Shade`, and `Width` properties of the column container in OA Extension, as described in Step 5 of Defining an Advanced Table.

Step 1: Set the `No Wrap` property to **False** (default) to allow wrapping of a column's cell content.

Step 2: Unlike earlier tables, the alignment of data in the columns of an advancedTable region is no longer automatically determined by the data type of the column's child. You *must* set the alignment using the `Alignment` property of the column. The default is textFormat, which aligns the data to the Start position. You can also

set the alignment to iconButtonFormat (Center) or numberFormat (Right).

Note: The BLAF standards for the alignment of numeric column content has been revised. Read-only and updateable numeric values that could be totaled or compared across cells in columns, (such as currency formatted values or quantity), may be set as right-aligned together with their column headers. Numeric and alphanumeric values, (such as social security, passport, sku, or identification numbers), that are not used for totaling or comparisons across cells in columns are left-aligned together with their related column headers. Therefore, if your column contains numeric content of the latter type, set the `Alignment` property to textFormat.

Step 3: Set the `Grid Displayed` property to **True** to render grid lines to the left of each column.

Step 4: Use the `Width` property on the column to set the width of the column in terms of pixels or percentage of the parent element (advancedTable). To specify a percentage, include the percent symbol after the value, as in 50%.

Runtime Control

No runtime code is required to format a column, but if you wish to change the alignment of a column programmatically, you can include code in the `processRequest` method of your controller, similar to the example shown below:

```
{
  OAColumnBean columnBean = ...;

  // Get the column format if already existing
  DictionaryData columnFormat = (DictionaryData) columnBean.getColumnFormat();
  if (columnFormat == null)
    columnFormat = new DictionaryData();

  // This end-aligns a column
  columnFormat.put(COLUMN_DATA_FORMAT_KEY, NUMBER_FORMAT);
  columnBean.setColumnFormat(columnFormat);
}
```

Attention: If your table includes a column that contains a messageLovInput item with a `Data Type` of NUMBER, the data is always right-aligned. However, for the column containing the messageLovInput item as its child, only the column header and not the column data can be left-aligned. In Release 12, you can now use the `oracle.apps.fnd.framework.webui.beans.message.OAMessageLovInputBean` API `setDataAlignment(String dataAlignFormat)` to specify the data alignment for a messageLovInput item. The valid values for the `dataAlignFormat` parameter are `NUMBER_FORMAT` and `TEXT_FORMAT`.

For example, to left-align the data in a messageLovInput item with data type NUMBER, you would include code such as the following in your controller:

```
OAMessageLovInputBean lovBean =
  (OAMessageLovInputBean)webBean.findChildRecursive(...);
lovBean.setDataAlignment(TEXT_FORMAT);
```

To right-align the data in a messageLovInput item with data type NUMBER, you would include this code in your controller (Note: The default alignment is right-align, so normally you wouldn't need to include this code, but it is shown here to simply illustrate a point):

```
OAMessageLovInputBean lovBean =
  (OAMessageLovInputBean)webBean.findChildRecursive(...);
lovBean.setDataAlignment(NUMBER_FORMAT);
```

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Row Formatting

OA Framework does *not* set any default row formatting, except to render a horizontal grid line above each row.

Declarative Implementation

There is currently no declarative support for formatting the rows in a table.

Runtime Control

The UIX `TableBean rowFormats` object determines the row format of a table. To modify a row's formatting, you can use the `TableBean getRowFormats` method. This method returns a `DataObjectList`. Each `DataObject` in the list corresponds to one row from top to bottom in the table format. Specify a format inside each `DataObject` to alter the formatting of each row accordingly. These formatting `DataObjects` store their values under the known, publicized key (UIConstants) `DISPLAY_GRID_KEY`. UIX queries each of the `DataObjects` for the `DISPLAY_GRID_KEY` property to determine if it returns `BOOLEAN.TRUE` or `BOOLEAN.FALSE`. Each one that returns `BOOLEAN.FALSE` from this query suppresses a grid line above that row.

Note: If a `DataObject` returns a value for a key, the value is used to change the row format. If no value is returned for a key, the table assumes the default format.

The following code example illustrates how you can override the default row format by suppressing a row's grid line.

```
public void processRequest (...)
{
  ...

  // get a handle on the array of Row Format Dictionaries
  DataObjectList myRowFormats = tableBean.getRowFormats();
  if (myRowFormats = null)
    DictionaryData[] myRowFormats = new DictionaryData[size];

  // set the DISPLAY_GRID_KEY property to FALSE for the second row
  // which corresponds to index 1 in the DictionaryData array
  rowFormats[1] = new DictionaryData(DISPLAY_GRID_KEY, Boolean.FALSE);
```

```
  //Modify any other row formats
  ...

  tableBean.setRowFormats(rowFormats);
  ...
}
```

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Column Header/Row Header Formatting

The only formatting you need to perform on a column header or a row header is to control its ability to wrap its content and its alignment. The default is to allow wrapping because not doing so can cause the table to grow too wide.

Declarative Implementation

Format a column header by setting the `No Wrap` property on the column or columnGroup header's sortableHeader component. The default is **False** to allow wrapping.

There is currently no declarative support for formatting row headers.

Runtime Control

The only programmatic column header formatting that you can do is change the wrap setting for the column header. To do this, add the following code to your controller:

```
...
public void processRequest (...)
{
  OAColumnBean columnBean = ...;

  // Get the column header format if already existing
  DictionaryData columnHeaderFormat = columnBean.getColumnHeaderFormat();
  if (columnHeaderFormat == null)
    columnHeaderFormat = new DictionaryData();

  // This ensures that the column header never wraps.
  columnHeaderFormat.put(CELL_NO_WRAP_FORMAT_KEY, Boolean.TRUE);
  columnBean.setColumnHeaderFormat(columnHeaderFormat);
}
```

To format a row header, use the `TableBean getRowHeaderFormats` method. This method returns a `DataObjectList`. Each `DataObject` in the list corresponds to one row header from top to bottom, in the table format. You specify formats inside each `DataObject` to alter the formatting of each row header accordingly. These formatting `DataObjects` store their values under the known, publicized key (UIConstants) `CELL_NO_WRAP_FORMAT_KEY`. This property determines whether the row header should be rendered with line wrapping disabled. To override the default, return `Boolean.TRUE` to render the row header without wrapping the contents.

Note: If a `DataObject` returns a value for a key, the value is used to change the row format. If no value for a key is returned, the table assumes the default format.

Personalization Considerations

- See a summary of Advanced Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Table Row Dynamic Poplist

OA Framework provides programmatic support for the implementation of a choice list (poplist or pulldown) in an updatable multi-row table, so that its poplist values can vary on a row-by-row basis.

Declarative Implementation

This feature can not be declaratively implemented.

Runtime Control

Refer to Dynamic Poplists in Standard Web Widgets for programmatic instructions.

Personalization Considerations

See a summary of Standard Web Widgets personalization considerations in the Oracle Application Framework Personalization Guide if you plan to implement a dynamic poplist in an advanced table.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Table Performance Issues

There are a number of issues in BC4J that could trigger all the rows to be brought in. Be aware of the following issues with your code:

- Do *not* issue `getRowCount()/last()/setFetchMode(FETCH_ALL)` on your table view object if you want to use incremental fetch logic.
- Avoid `setRangeSize(-1)`. If your view object already contains some rows and you issue `setRangeSize`, BC4J attempts to fill up the range by bringing in rows from the database if the number of rows in the view object is less than the range size that you try to adjust to. In this case, `setRangeSize(-1)` would cause all the rows to be brought in.
- If you need to use the table view object to iterate the rows before rendering the table, use it with caution. Also, be careful using the `setRangeSize` and `setRangeStart` methods on a table view object.

  Note: There may be cases when you would set the "range start" to 0 and the "range size" to be equal to the total row count to iterate all the rows in the view object. After the iteration, reset the range start and size to the original values.

  - When you set the "range size" to a size that is potentially larger than the current size, set the "range start" first to 0, then set the "range size". If you set the "range size" first and your current "range start" is greater than 0, BC4J faults in some rows from the database to fill up the range when it sets the new "range size". BC4J then resets the "range start" to 0 as well as part of the `setRangeSize` operation if the new "range size" is different from the current range size.
  - When you set the "range size" to a size that is potentially smaller than the current size (usually when you restore the original "range size" and start), then reverse the order -- set the "range size" first and then the "range start" because you may want to set the "range start" to a value greater than 0).
- The table rendering logic makes the following three assumptions to render a table without any extra logic that would degrade performance and avoid unnecessarily re-setting and adjusting the "range size" multiple times:

  1. For the table to render properly, make sure the table view object "range start" is the default value 0 before the table bean is created and rendered for the first time. If your code performs row navigation through methods like the view object `next` or `last` methods before the table bean is rendered, these methods move the "range start" to a value greater than 0. (The view object methods `first` or `previous` moves up the range.) Perform the following in this case:

     Option 1:

     ```
     int fetchedRowCount = vo.getFetchedRowCount();
     if (fetchedRowCount > 0)
     {
       // Save the range start and size.
       int savedRangeStart = vo.getRangeStart();
       int savedRangeSize = vo.getRangeSize();

       // When you set the range size to a size that
       // is potentially larger than the current size, you should
       // set the range start first to 0 and then set the range size.
       vo.setRangeStart(0);
       vo.setRangeSize(fetchedRowCount);
     ```

     try:

     ```
     {
       for (int i = 0; i < fetchedRowCount; i++)
       {
        // Or you can use getAllRowsInRange() outside the for loop.
        Row row = vo.getRowAtRangeIndex(i);
        ...
        // Perform row operation.
       }
     }
     ```

     finally:

     ```
     {
       // Restore the original range - do in finally clause,
       // in case action on row throws an exception.
       // When you set the range size to a size that is
       // potentially smaller than the current size
       // (usually when you restore the original range size and start),
       // then reverse the order
       // -- that is, set the range size first and then the range start.
       vo.setRangeSize(savedRangeSize);
       vo.setRangeStart(savedRangeStart);
     }
     ```

     Option 2:

     Alternatively, use a secondary iterator if you do not want to risk overriding the default row set range or row currency:

     ```
     RowSetIterator deleteIter = vo.createRowSetIterator("deleteIter");
     int fetchedRowCount = vo.getFetchedRowCount();
     if (fetchedRowCount > 0)
     {
       deleteIter.setRangeStart(0);
       deleteIter.setRangeSize(fetchedRowCount);
       for (int i = 0; i < fetchedRowCount; i++)
       {
         Row row = deleteIter.getRowAtRangeIndex(i);
         ...
       }
     }
     ```

finally:

```
{
if (deleteIter != null)
  deleteIter.closeRowSetIterator();
}
```

Note: You can also use the following convenience methods if you want to find rows matching some criteria: `OAViewObject.getFilteredRows` and `OAViewObject.getFilteredRowsInRange`.

2. Exception to assumption 1: If you redirect to current page with `retainAM=Y` in the URL, the "range start" can be greater than 0 if the user is already in some range other than the first range. To preserve the current table range upon redirect, place the following check on the `executeQuery` call:

   ```
   if (!vo.isPreparedForExecution())
     vo.executeQuery();
   ```

   This is because `executeQuery` resets the "range start" to 0, which may not be what you want when you redirect with `retainAM=Y`.

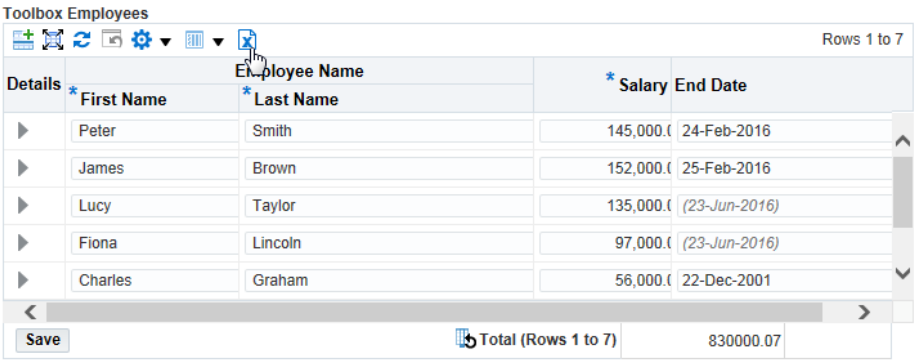
3. If you programmatically set the number of rows displayed for the table, it takes effect *only* if the table view object "range start" is initially 0.

Row-Level Error Prefix

You can now override the default row prefix that OA Framework displays for row and attribute error messages in a table. See the Overriding the Row-Level Error Prefix in the Error Handling document for additional information.

Export Table Data

As of Release 12.2.6, you can enable the export of table data from tables by simply setting the profile option FND: Enable BNE Table Integration. When you enable this feature and select the export table data icon in the table control bar, the data in the view object of the associated table bean exports to a Microsoft Excel spreadsheet that you can download to your machine. This feature invokes Oracle Web Applications Desktop Integrator (Oracle WebADI) to create the Excel spreadsheet. For more information about Oracle WebADI, refer to the "Accessing Oracle Web Applications Desktop Integrator" topic in the "Introduction to Oracle Web Applications Desktop Integrator" chapter of the *Oracle Web Applications Desktop Integrator Implementation and Administration Guide*.



The following configurations do not support the export table data feature:

1. Tables rendered in the email facet.
2. Tables rendered in printable pages.
3. Inner Tables (table-in-table).
4. Tables in pages running in embedded mode, such as tables inside a JTT skin.
5. Tables in pages running in portlet mode.
6. List of values table web beans.
7. Attachment tables.
8. Operating systems other than Windows.

Exporting Tables With Switcher Columns

Be aware of the following important requirements when exporting tables that have switcher columns.

Scenario	Expected Result
One of the columns and the the child 0 under column is a switcher, and all the cases children are non web bean data attributes such as image beans.	The column does not appear in Excel.
One of the columns and the child 0 under column is a switcher. One case is a non web bean data attribute (image) and another case is a web bean data attribute (link or message styled text).	The column appears in Excel. For rows that evaluate to the case non web bean data attribute, you must observe blank cells. For rows that evaluate to the case web bean data attribute, you must observe the data in Excel.
One of the columns and the child 0 under column is a switcher, and all the cases are web bean data attributes (link or message styled text). We assume that case 1 has the view attribute va1, and case 2 has the view attribute va2.	The column appears in Excel. For rows that evaluate to the case va1, you must observe data that is obtained from row.getAttribute(va1). For rows that evaluate to case va2, you must observe data that is obtained from row.getAttribute(va2).

One of the columns is a switcher, and all the cases are web bean data attributes (link or choice). We assume that case 1 has the view attribute va1, and case 2 has the view attribute va2 if the second case is a choice.	The column appears in Excel. For rows that evaluate to the case va1, you must observe data that is obtained from row.getAttribute(va1). For rows that evaluate to the case va2, you must observe data that is obtained from row.getAttribute(va2) . As the second case is a choice, you must see the name (Pick List Display Name) and not the value (Pick List Value attr) of the option.

Rich Table Interactions

To enable the rich table interactions introduced in Release 12.2.3, set the profile **FND: Enable Rich Table Interactions** to **True**.

Note: If you set the profile **FND: Enable Rich Table Interactions** to **True**, OA Framework creates a user view of any rich modifications you make. The user view always takes highest precedence when OA Framework displays the table. As a result, if you reorder a table using the admin Personalization Framework UI, the saved admin personalization will not take effect. If you wish to display the table columns as ordered according to your admin personalization, you can set the **Enable Column Reorder** property to false for those tables for which you feel admin personalization should take precedence. If you set the profile **FND: Enable Rich Table Interactions** to **False**, you disable all rich interactions across all products and tables, as well as disable mobile gestures.

Known Issues

- See a summary of key table issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
  - Tables Overview
  - Table Layout
  - Table Subcomponents
  - Manipulation of Table Display
  - Common Table Actions
  - Table Action and Navigation Options
  - Personalization of Table Views
- Javadoc File(s)
  - `oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean`
  - `oracle.apps.fnd.framework.webui.beans.OAWebBean`
  - `oracle.apps.fnd.framework.webui.beans.OAWebBeanDataAttribute`
  - `oracle.apps.fnd.framework.webui.OAWebBeanConstants`
  - `oracle.apps.fnd.framework.webui.beans.table.OAAddTableRowBean`
  - `oracle.apps.fnd.framework.webui.beans.table.OAColumnBean`
  - `oracle.apps.fnd.framework.webui.beans.table.OATotalRowBean`
  - `oracle.apps.fnd.framework.webui.beans.table.OATableUtils`
  - `oracle.apps.fnd.framework.webui.OAInnerDataObjectEnumerator`
  - `oracle.cabo.ui.beans.table.TableBean`
- Lesson(s)
  - Framework Toolbox Tutorial: Create - Part 1
- ToolBox Tutorial / Sample Library
  - To view Advanced Table examples implemented in the Sample Library project of the Toolbox workspace, run the `toolbox.jws` -> `SampleLibrary.jpr` -> `SampleBrowserPG.xml` file in OA Extension. Select the *Advanced Table* link in the *SampleBrowserPG* page that displays in your browser to view the implementation of the Column Span, Table Actions, Required Fields indicator, Row Banding, Detail Disclosure, and Add Another Row UI features.
  - The following package files in the Sample Library display the declarative and runtime implementation of the Advanced Table examples:
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.SampleBrowserPG.xml`
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.AdvancedTablePG.xml`
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.AdvTablePageCO.java`
- FAQs
  - Tables and Advanced Tables

Attachments

Overview

As described in the Oracle Browser Look-and-Feel (BLAF) UI Guidelines: Attachments Templates and Attachments Flows, use the attachments feature to associate a URL, file content, or text with an object, such as an expense report, contract, or purchase order.

To enable the attachments feature, you should first understand the concept of an entity. An entity is an object within Oracle E-Business Suite data, such as an item, an order, or an order line. (An entity is *not* related to BC4J Entity Objects.) The attachments feature *must* be enabled for an entity before users can link attachments to the entity. In the context of attachments, an entity can be considered either a base entity or a related entity. A base entity is the main entity of the region. A related entity is an entity that is usually related to the region by a foreign-key relationship.

An entity consists of a set of view attribute names (declared through your OA Extension region items), which are the primary keys for the view object related to your region. When an attachment is stored, the values of these primary keys and the Entity ID are stored so the attachments can be uniquely retrieved. Several regions can access the same entity, providing a consistent view of attachments throughout your application.

OA Framework also supports multiple entities associated with a region item, with functionality similar to the core functionality of attachments in Oracle E-Business Suite Forms-based functions.

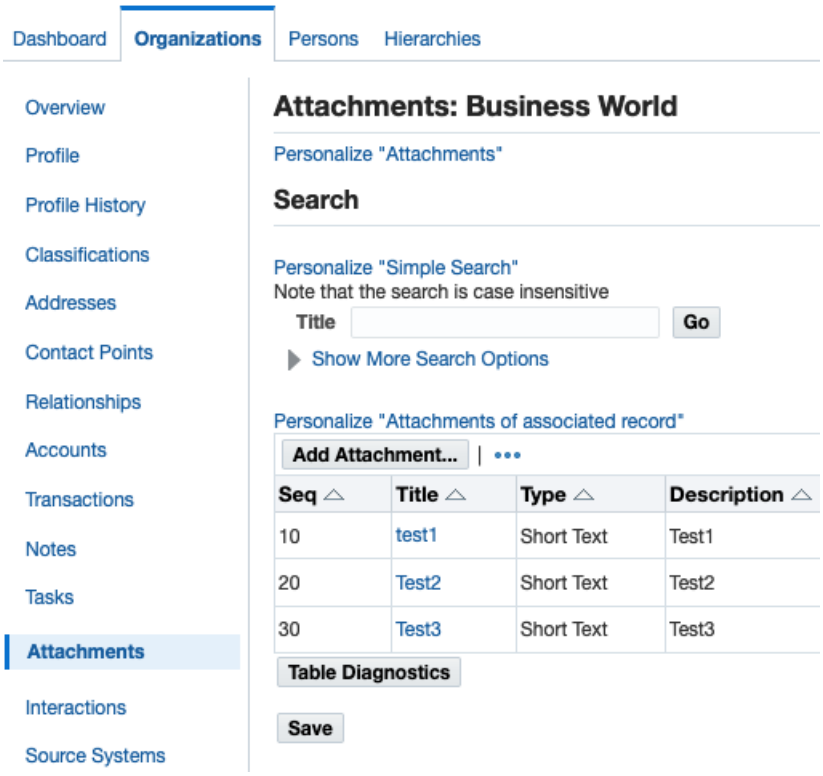
For instance, a purchase order has a PO Header and PO Lines. At the line level, you can add attachments pertaining to the entity `PO_LINES`. Note that you usually enter an item number in PO Lines. When you create the item in the master Item form, you can add an attachment, such as a picture of the item, that gets stored with a different entity, such as `MTL_SYSTEM_ITEMS`. With multi-entity support, when you view the attachments for a PO Line, you can see the attachments for both entities, that is, `PO_LINES` and `MTL_SYSTEM_ITEMS`.

Contents

- Release 12.2.10 Enhancements
- Release 12.2.7 Enhancements
- Release 12.2.6 Enhancements
- Release 12.2.5 Enhancements
- Attachment Styles in a Region
- Attachments Page User Interface
- Inline Attachments
- Enabling the Attachments Feature for an Entity
  - Declarative Implementation
  - Runtime Control
  - Enabling Virus Scanning
- Personalization Considerations
- Services Exposed
- Known Issues
- Related Information

Release 12.2.10 Enhancements

From Release 12.2.10, attachment inline and page mode displays an attachment sequence number similar to that used in Oracle Forms. The first column where attachments are displayed is "Seq", which refers to the sequence number. This is read only from Oracle Application Framework, and the value is incremented by 10 from the largest available sequence number.



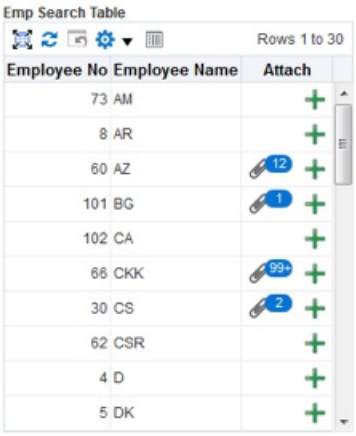
Release 12.2.7 Enhancements

In Release 12.2.7:

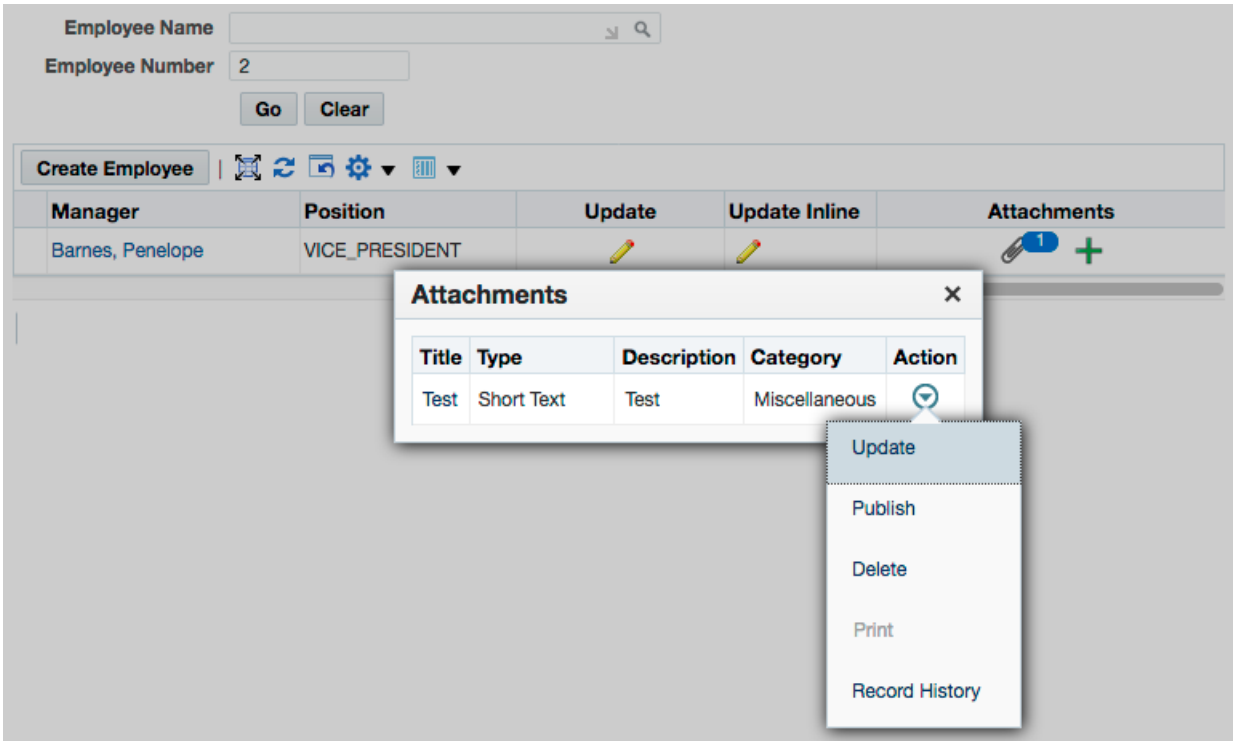
- The Add/View Attachment icons for the Attachment UI have been updated:



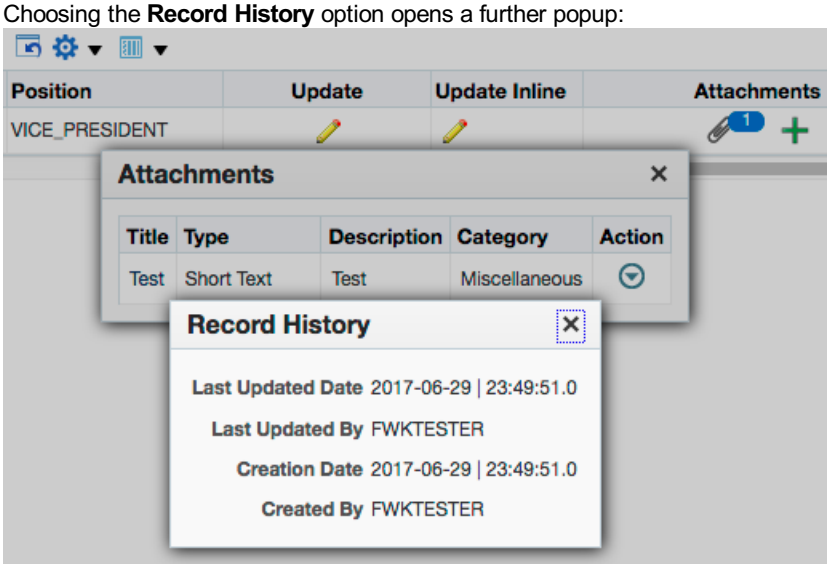
- The total number of attachments is now shown, in a badge next to the View Attachment icon. If there are more than 99 attachments, the attachment count is shown as 99+:



- An **Attachments** popup window gives a summary of any attachments, and allows various actions to be chosen:



- Choosing the **Record History** option opens a further popup:



- A size constraint for the Inline Attachment UI enables it to provide the optimum experience for mobile device users as well as desktop users.

Release 12.2.6 Enhancements

In Release 12.2.6:

- You can set a default attachment type for the Inline Attachment UI.
- You can restrict the attachment types that are allowed in the Inline Attachments or Attachments Page UI.
- Inline Attachment pop-ups now support drag and drop for keyboard users.
- Oracle E-Business Suite integration with AutoVue Document Print Service is now certified only with Oracle Autovue Release 20.2.3.
- You can set the new profile option FND Attachment URL Validation to enable URL-type attachment validation, such that if a URL directs to an "unsafe" external site, a confirmation dialog prompts the user to confirm before redirecting.

Release 12.2.5 Enhancements

To enhance productivity, users by default will be able to add, update, copy or view attachments through Inline Attachment pop-ups

rather than having to navigate through the former Attachments Page UI. In Release 12.2.5, the Inline Attachments UI renders as modal pop-ups. The "Add", "Update" and "View" attachment pages are suppressed as users can now accomplish all attachment-related functionality using these Inline Attachment pop-ups. For example, users can now select an attachment from a document catalog as well as add or edit a description for an attachment from the Inline Attachment pop-up UI.

You can, however, revert back to the old behavior of interacting with the Attachments page UI by disabling Inline Attachments.

Attachment Styles in a Region

When a region is attachment-enabled, a user can add, view, and delete attachments associated with the records in that region. The attachments are generally displayed in an Attachments table or Attachments page, although you can also display inline links of attachments in a single row region or enable a feature that allows inline viewing and addition of attachments.

Note: You can also render an Attachments table as read-only.

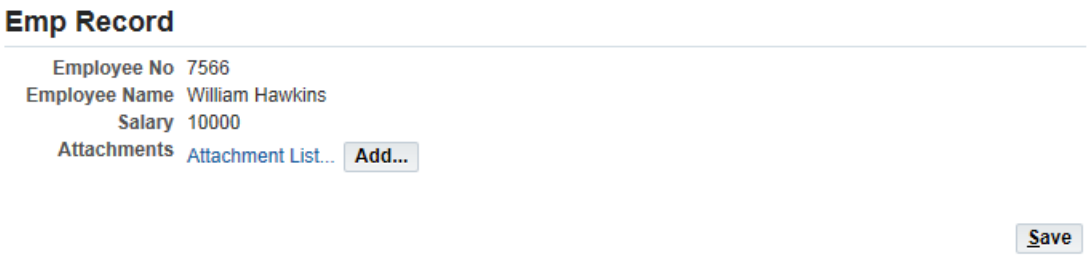
There are four attachment styles that you can render in a region:

- Display a View List Link and Add Button in a Single Row Region
- Display an Attachments Table on the Page of a Single Row Region
- Display Inline Links of Attachments in a Single Row Region
- Display an Attachments Column in a Multi-Row Table Region

Display a View List Link and Attachment Icon in a Single Row Region

In the case of a single-row region, you can implement attachments by displaying a **View** link, whose label you may change. In the example shown below, the default **View** link is overridden by the value "Attachment List". Beginning in Release 12.2.5, the user may select the link to display the Inline View Attachments pop-up or select the **Add** button to display the Inline Add Attachment pop-up.

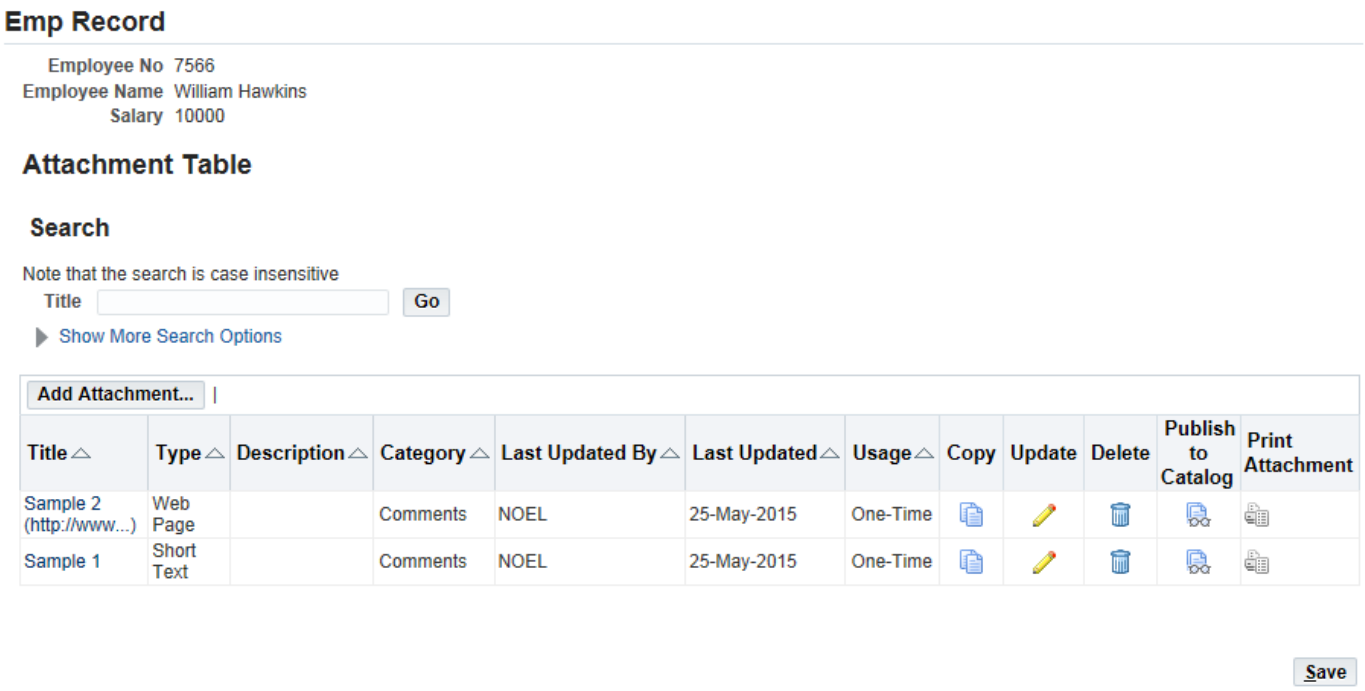
Figure 1. Example of an Employee Record page with an attachment View List link.



Display the Attachments Table on the Page of a Single Row Region

For a single-row region, you can choose to render the Attachment Table region on the product page directly as shown in the figure below.

Figure 2. Example of an Employee Record page containing an Attachment table.



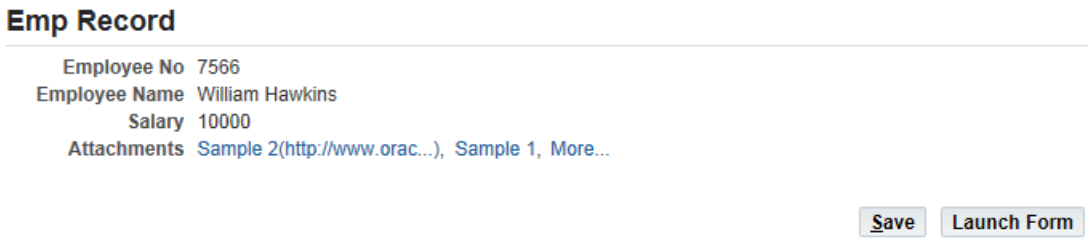
As of Release 12.2, if you enable the Copy Attachments feature for an Attachment Table in a single row region, a **Copy** column appears in the Attachment Table. A user may select the **Copy** icon to copy an attachment of any type to another entity.

Display Inline Links of Attachments in a Single Row Region

For a single-row region, rather than display a **View List** link or an Attachments table in the region, you may choose instead to display inline links to the first few attachments for the record as shown in the figure below. The inline links allow users to easily view the first few attachments without navigating from the current page and is generally used in the Workflow Notification Details page, as described in the Oracle Browser Look-and-Feel (BLAF) UI Guidelines: Notification Page Templates.

Users can also view additional attachments if there are any, or perform further operation on the attachments by selecting the **More ...** link that follows the attachment links. The **More ...** link displays the Inline View Attachments pop-up. If the attachments are not updateable, and the number of attachments that exist is equal to or less than the number of links that are allowed to be displayed (as set when you enable this feature), the **More ...** link does not appear.

Figure 3. Example of an Employee Record page displaying inline links of attachments.



If the attachment type of the inline link is File, then when you select the link, a dialog box opens in your Browser. You can either open the file and display the content or save the file. If you choose **Save**, the file is created and saved to your client machine.

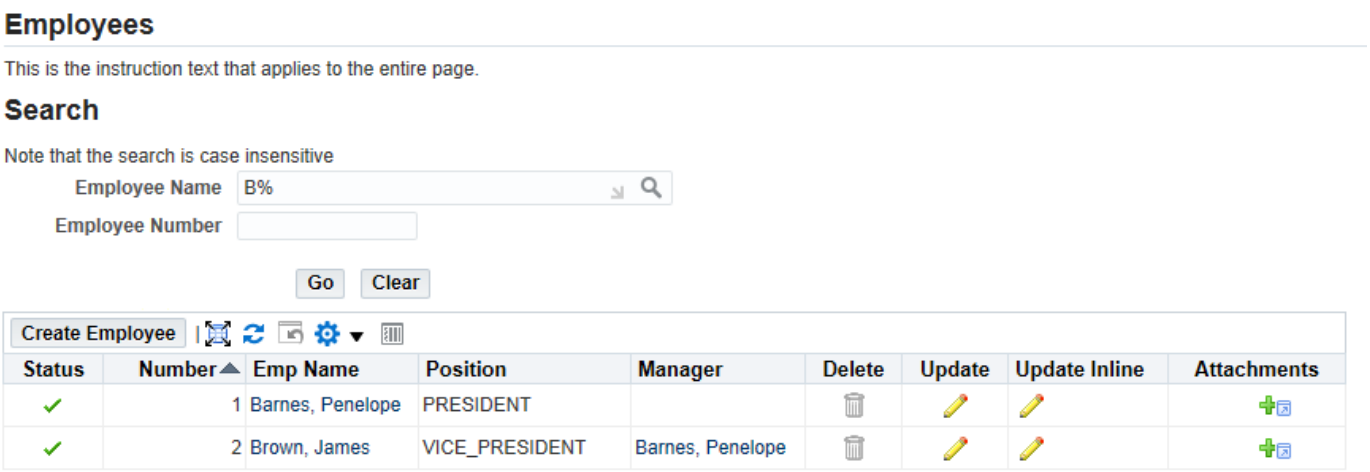
If the attachment type of the inline link is Short Text or Long Text, then when you select the link, you navigate to the Inline View Attachments pop-up, where you can view the content of the text.

If the attachment type of the inline link is URL, then when you select the link, you navigate to the destination URL.

Display an Attachments Column in a Multi-Row Table Region

You may also enable attachments in a multi-record table region by displaying an **Attachment** column at the end of the table. Each row in the table that contains an attachment displays an **Attachments** icon (a paperclip attached to paper), that displays the Inline View Attachments pop-up when selected, and an **Add** icon (a plus sign), that displays the Inline Add Attachment pop-up when selected.

Figure 4. Example of a multi-row region with an Attach column, containing Add icons in the table.



Attachments Page User Interface

The following pages or functions are available in the traditional Attachments flow:

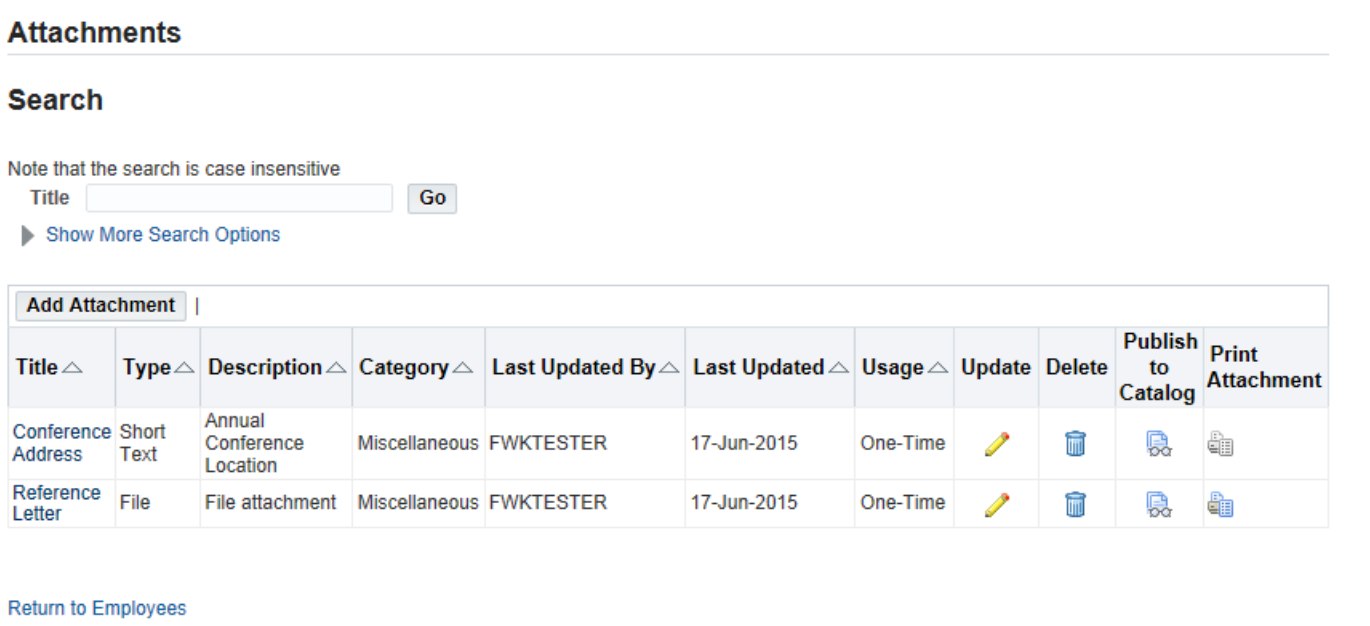
- Attachments Page
- View Attachment Page
- Adding Attachments
  - Add Attachment Page
- Editing an Attachment
- Deleting an Attachment
- Publishing a Document to the Document Catalog
- Printing File-Type Attachments

Note: As of Release 12.2.5, the Attachments Page UI appears only if the combined values of the `Inline Attachment Enabled` property of the Attachment region and the **FND: Disable Inline Attachment** profile option value are set accordingly.

Attachments Page

When you select the Attachments icon in a multi-row region, the View List link in a single-row region or the More... link that follows the inline attachment links in a single-row region, you launch the Attachments flow by displaying the Attachments page.

Figure 5. Attachments page



If the list of attachments is long, use the Hide/Show search region to list specific attachments by description, category, last updated

by and/or last updated date.

Select a **Title** link to view the content of an attachment in the View Attachment page. Select the **Update** icon to edit the attachment in the Update Attachment page or select the **Delete** icon to delete an attachment.

Select the **Add Attachments** button above the table to add attachments using the Add Attachment page. If you select the **Return to ...** link at the bottom of this page, you return to the product page from which you launched the Attachments flow.

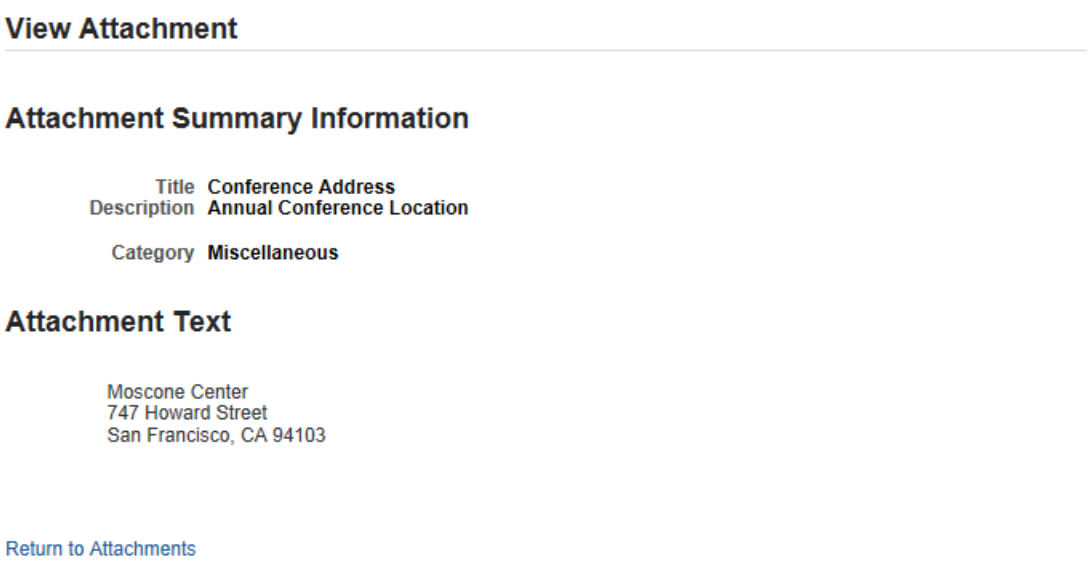
The Attachments page may also vary slightly depending on whether you enable certain features when you implement attachments in a region.

- If you enable the Document Catalog, and the function **FND_FNDATTCH_PUBTOCAT** is granted to the current user, a **Publish to Catalog** column appears in the Attachments page. You can select an enabled **Publish to Catalog** icon for a document to publish that document to the Document Catalog.
- As of Release 12.2, if you set the profile option FND Attachment AutoVue Server to enable printing using the AutoVue Document Print Service, a **Print Attachment** icon in the Attachments page is enabled for all file-type attachments.

View Attachment Page

To view the content of an attachment, select a File Name link in the first column of the table on the Attachments page. If the attachment is a web page, you navigate to the specified URL when you select the link. If the attachment is a file, a dialog box opens in your Browser when you select the link. You can either open the file and display the content or save the file. If you choose **Save**, a file with the content is created and saved to your client machine. If the attachment is short text or long text, a View Attachment page appears, as shown below, displaying the text content.

Figure 6. View Attachment page



When you select the **Return to Attachments** link on this page, you return to the Attachments page.

Adding Attachments

Add attachments to the current record using the Add Attachment page. The Add Attachment page varies depending on whether you enable the Document Catalog when you implement attachments in a page. If you:

- **Enable Document Catalog** - an Add poplist control appears at the top of the Add Attachment page. The Add poplist displays two choices:
  - **Desktop File, Short Text, Long Text, URL** - selecting this choice updates the Add Attachment page so you can add a file, text, or URL as an attachment.
  - **From Document Catalog** - selecting this choice updates the Add Attachment page so you can add an attachment from the Document Catalog.
- **Disable Document Catalog** - an Add poplist control does *not* appear at the top of the Add Attachment page, and you can only add a file, text, or URL as an attachment.

Add Attachment Page

When you add an attachment, you need to specify a category for the attachment. A category is a label that users apply to individual attachments and documents. When you set up a region to be attachment-enabled, you *must* assign a category for the documents

that can be attached. The Attachments page can query only those documents that are assigned to the category to which the calling page's entity is associated. A "Miscellaneous" category is seeded to provide easy visibility of a document across pages. If you do not assign a category to the documents that can be attached, the category defaults to Miscellaneous.

Adding a Desktop File, Text, or URL as an Attachment

In the Add Attachment page, select "Desktop File/ Text/ URL" from the Add poplist to add a new attachment. Specify a title to label the attachment, a description and a category, followed by an attachment type (File, URL, or Text) and attachment information (file name, URL, or the text itself) for the new attachment, as shown below.

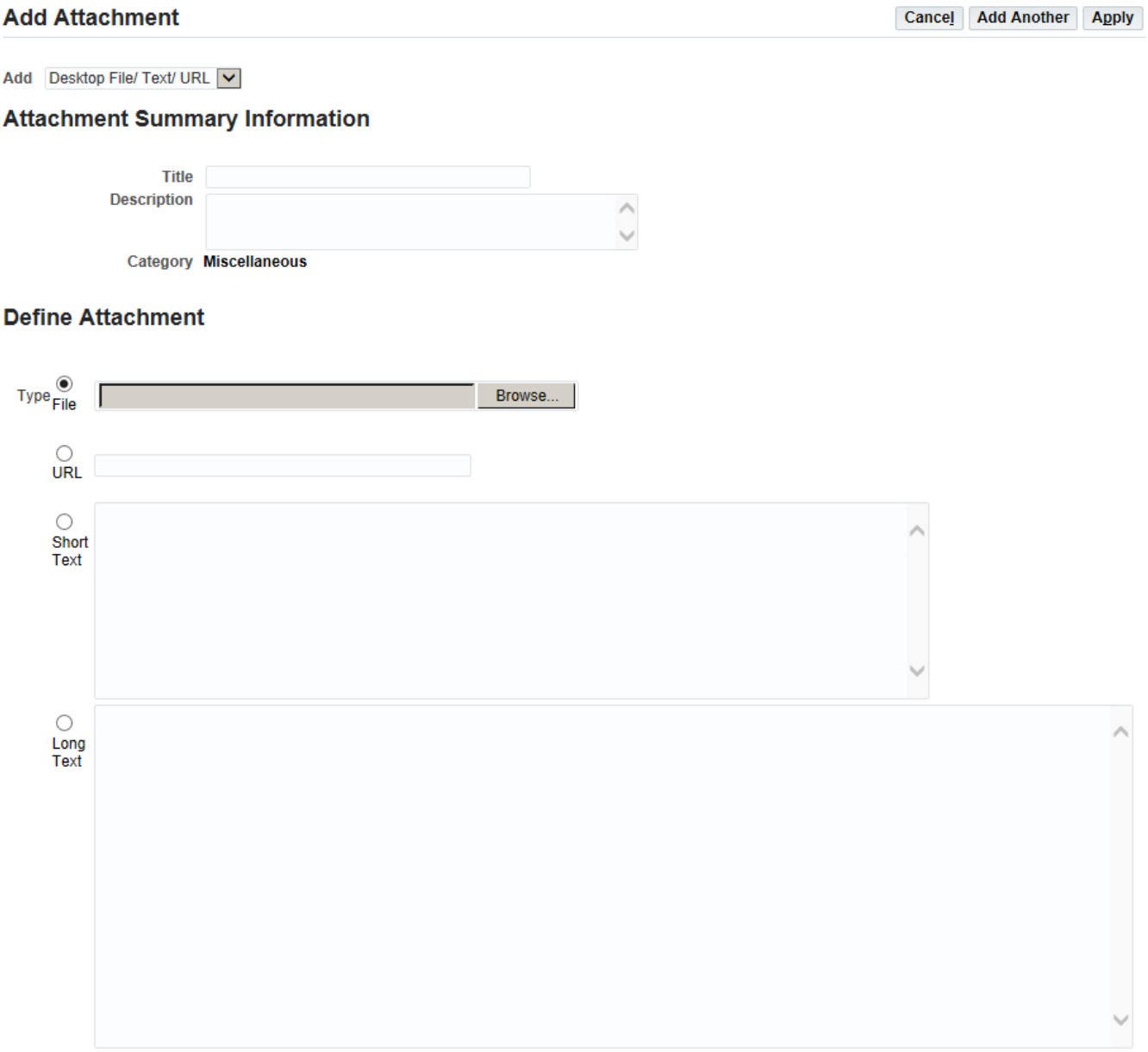
Note: Starting in Release 12.2.6, you can restrict the attachment types for an attachment component during design time or by using the personalization UI. Note that during design time, you are not allowed to restrict (disable) the attachment type that you previously set as the default attachment type. Any attempt to do so will result in a validation error. Similarly, if you attempt to set such a faulty configuration using the personalization UI, the default attachment type, which is considered enabled, will appear with a relevant developer mode exception.

Attention: Short Text type attachments have a maximum length of 4000 bytes. If you need to add an attachment that is longer than 4000 bytes, use Long Text type attachments.

Attention: An error message displays if you attempt to add a .txt, .doc or .docx file of 0 bytes. An error does not display if you attempt to add a document (.doc or .docx) with empty content that is not 0 bytes.

Note: If you specify a file-type attachment but do not specify a title for the attachment, the filename displays as the title instead. If the filename exceeds thirteen characters, then the first eleven characters display, followed by an ellipsis (...).

Figure 7. Add Attachment page



In the Add Attachment page, you can select **Cancel** to cancel and return to the previous Attachments page, select **Add Another** to add another attachment, or select **Apply** to insert the rows into the view object and return to the Attachments page.

Note: The new attachments are *not* committed until a submit is performed by the associated parent region. Clicking **Apply** on the Add Attachment page does *not* commit the changes by default. If you want a commit to occur when clicking **Apply**, refer to the Runtime Control - Committing Changes section for additional information.

AntiSamy Support and Attaching HTML Files

Previously, OA Framework handled the contents of file-type attachments as a byte stream, and could attach any file regardless of its character encoding. Now with the introduction of AntiSamy HTML sanitizing support in Oracle E-Business Suite Critical Patch Updates (CPU) for October 2011, January 2012, April 2012 and July 2012, if you attach an HTML file you must do the following:

1. Ensure that a `<meta>` tag for "Content-Type" exists in that HTML file with the correct character encoding name. For example:

   ```
   <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
   ```

2. If you cannot insert the `<meta>` tag into the HTML file, you must ensure the FND_NATIVE_CLIENT_ENCODING profile option value is set to the character encoding of the HTML file being attached.

AntiSamy is an API provided by the Open Web Application Security Project (OWASP) to remove malicious pieces of code from

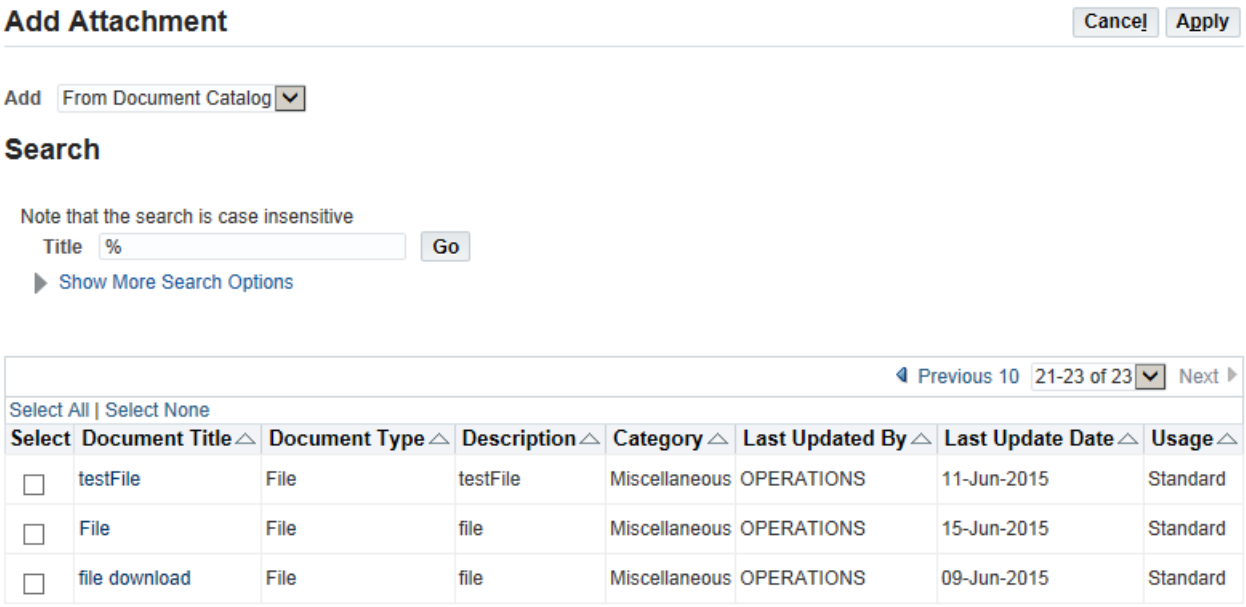
HTML files. OA Framework automatically applies this API if a file being attached has a "Content-Type" of "text/html". As mentioned earlier, OA Framework typically handles file-type attachments as a byte stream, but the AntiSamy API requires the HTML file to be scanned as a string. As a result, OA Framework must convert the contents of the file to a string. To ensure proper conversion, OA Framework must know the character encoding information of the file being attached to avoid corrupting non-ascii characters in the HTML. For more information about attachments security, refer to *Security Configuration Mechanism in the Attachments Feature in Oracle E-Business Suite My Oracle Support* (formerly Oracle *MetaLink*) Knowledge Document 1357849.1.

Adding an Attachment from the Document Catalog

Oracle E-Business Suite keeps a catalog of documents that have been attached to applications data records. You can use this catalog to attach an existing document to another data record, that is, copy an attachment from another record. To add an attachment from the Document Catalog, select "From Document Catalog" in the Add poplist of the Add Attachment page.

Use the search region to specify search criteria to locate one or more specific attachments. The results of your search are displayed in a table. The figure below displays all attachments currently stored in the document catalog:

Figure 8. Add Attachment page, displaying all attachments currently stored in the document catalog.



The table lists the name of the document, its type, description, category, who it was last updated by, when it was last updated, and its usage. Select the document name link to display the document in the View Attachment page.

The usage type indicates how the document may be used. There are three usage types:

- **Template** - the document is meant to be modified before use. When you select a Template usage document in the Document Catalog, a copy of the document is created and the document usage of the copy is set to One-Time. The copy is also updateable. If you delete an attachment that has a usage type of Template, the document content, as well as the association of the document to the record is deleted. Documents that appear in the Document Catalog with a usage type of Template, are created from the Documents Window screen in Oracle E-Business Suite. See the *Oracle E-Business Suite User's Guide* for additional information.
- **One-time** - the document is meant to be used only once. If you delete an attachment that has a usage type of One-time, the document content, as well as the association of the document to the record is deleted. All documents that you add using the Add Attachment page will have their usage type set to One-time.
- **Standard** - the document is a standard document that can only be referenced. When you select a Standard usage document in the Document Catalog, a copy of the document is not made. As a result, the document is not updateable. If you delete an attachment that has a usage type of Standard, the document content is not deleted, only the association of the document to the record is deleted. Documents that appear in the Document Catalog with a usage type of Standard, are created from the Documents Window screen in Oracle E-Business Suite. See the *Oracle E-Business Suite User's Guide* for additional information.

Note: You can only select **Standard** or **Template** usage types from the Document Catalog.

Once you locate the document(s) of interest, use the select checkbox to select the attachment(s). Choose **Apply** to attach the document(s) to your record and return to the Attachments page. Select **Cancel** to cancel and return to the Attachments page.

Editing an Attachment

In the Attachments page, you can choose the **Update** icon to update a particular attachment. The **Update** icon displays the Update Attachment page. The figure below shows the Update Attachment page for a Text attachment:

Figure 9. Update Attachment page for a text attachment.

Note: You can only edit the information as it pertains to the attachment. You cannot change the attachment type in this page. For example, if an attachment is a web page, you can update the URL that is associated with this attachment, but you cannot change the attachment type to a file.

In the Update Attachment page, selecting **Apply** updates the necessary rows in the view object and returns the user to the Attachments page.

Note: Changes to attachments are *not* committed until a submit is performed by the associated region. Clicking **Apply** on the Update Attachment page does *not* commit the changes by default. If you want a commit to occur when clicking **Apply**, refer to the Runtime Control - Committing Changes section for additional information.

Deleting an Attachment

Choose the **Delete** icon to disassociate an attachment from a record. If the usage type of the attachment is Template or One-time, a warning confirmation also appears as this action will delete the document content.

Important: You *must* delete the associated attachments before deleting the base entity, otherwise you will cause orphaned attachments. Therefore, the logic of deleting the base entity should also include deleting the associated attachments.

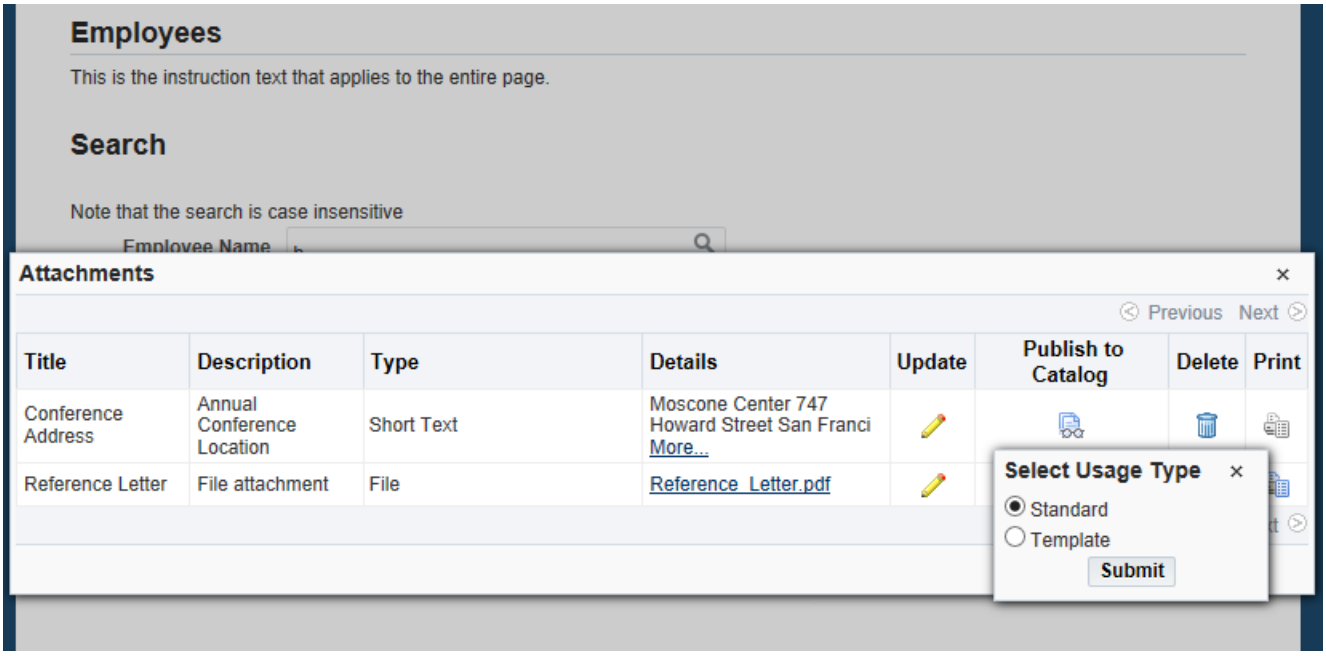
Publishing a Document to the Document Catalog

The **Publish to Catalog** icon is enabled in the Attachments page for any attachment that is a One-Time usage-type document. Choose the **Publish to Catalog** icon to publish the document to the document catalog so that it can be attached to other records. When you select the **Publish to Catalog** icon, the following behavior occurs depending on whether the document you select to publish is of type File, URL or Short Text or Long Text:

- **Short Text or Long Text**- a dialog page appears, asking you whether you want to publish the document to the Document Catalog as a Standard or Template usage-type document.
    - If you select **Standard** then choose **Continue**, the document is automatically reset as a Standard usage-type document and is published to the Document Catalog as such. A message appears above the Attachments page to confirm.
    - If you select **Template**, then choose **Continue**, a copy of the document is made and the copy is published to the Document Catalog as a Template usage-type document. A message appears above the Attachments page to confirm.
- **File** or **URL** - since File- or URL-type documents can only be published as a Standard usage-type, the document is automatically published to the Document Catalog as such and a message appears above the Attachments page to confirm this.

Note: A document is *not* actually published to the Document Catalog until you save (commit) all changes to your attachments in the originating region. In Release 12.2.5, the document is published to the Document Catalog when you select **Submit** from the **Select Usage Type** inline pop-up.

Figure 10. Publish to Document Catalog.



Once you publish a document to the document catalog, the **Publish to Catalog** icon is disabled for that document because it is no longer a One-Time usage document. You should be able to view that published document from the Document Catalog using the Add Attachment page.

Printing File-Type Attachments

As of Release 12.2, attachments are fully integrated with Oracle's AutoVue Document Print Service, allowing users to print their file-type attachments. This integration also allows an enterprise to support the printing of attachments of different file types from a server where the Auto Document Print Service is configured, without having to install or configure additional software to view and print files in client/server machines.

Note: Oracle E-Business Suite Release 12.2.6 integration with AutoVue Document Print Service is only certified with Oracle Autovue Release 20.2.3.

Important: If the Oracle E-Business Suite instance is running with the HTTPS protocol, then you must also import the SSL certificate of the Oracle E-Business Suite instance into the AutoVue server.

Note: For additional information about Oracle AutoVue, refer to the AutoVue Documentation Library.

To enable printing using the AutoVue Document Print Service, set the profile option FND Attachment AutoVue Server. Once this feature is enabled, the **Print Attachment** icon in the Attachments page will be enabled for all file-type attachments.

The Print Attachment icon:

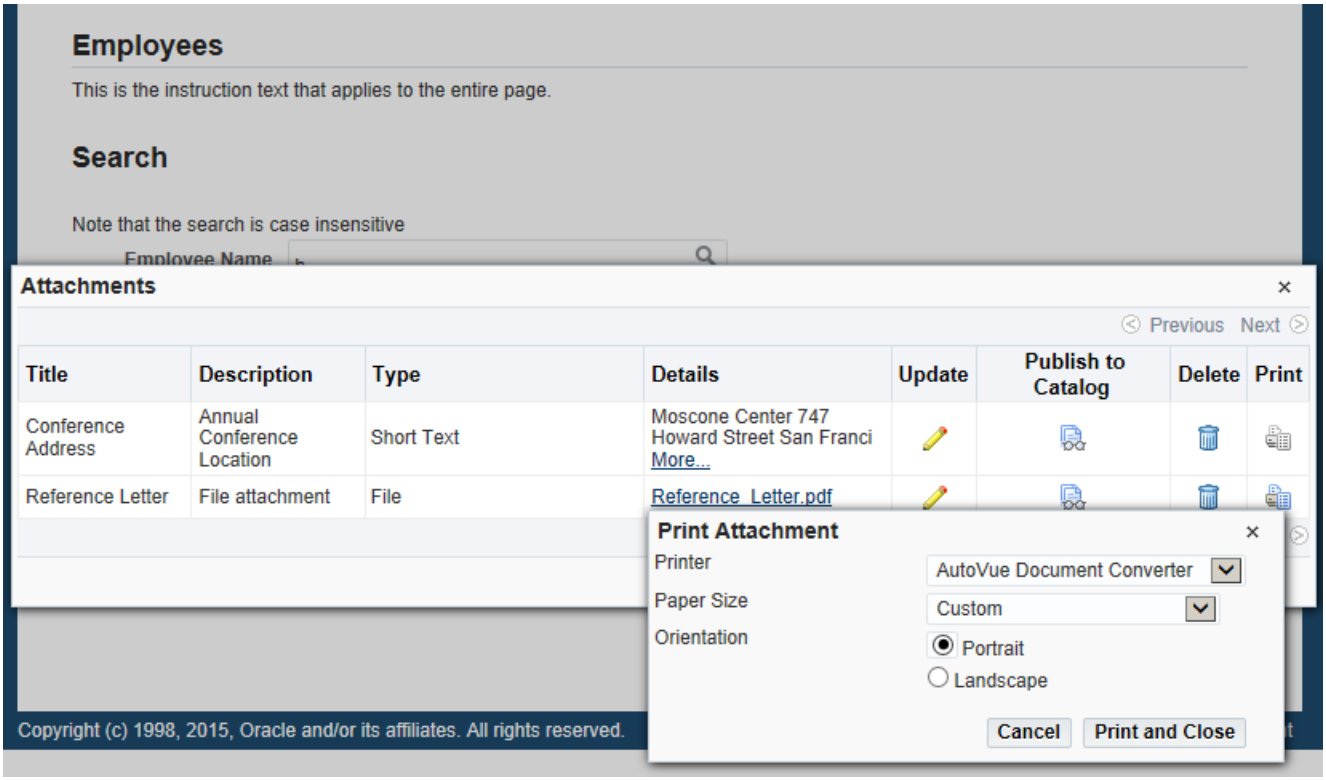


The icon remains disabled for all text- and URL-type attachments. Select the **Print Attachment** icon to display the **Print Setup** pop-up as shown below, where you can:

- Choose from a list of available printers
- Choose from a list of available paper sizes
- Specify portrait or landscape as the print orientation

  Note: An error message will appear if the AutoVue web service cannot be invoked to fetch the list of printers and paper sizes.

Figure 11. Print Setup pop-up.



Select **Print** to print the file with your selected settings or choose **Cancel** to close the **Print Setup** pop-up without printing. A confirmation message appears in the pop-up once the file is printed. If the file cannot print, an error message appears in the pop-up.

Note: You can only select the **Print Attachment** icon when the file-type attachment is committed to the database. If the Automatic Save property of an attachment region item is set to **False**, a user must explicitly commit the transaction of any newly-added file attachments before the user can print those attachments.

Inline Attachments

You can enable a feature called Inline Attachments that allows a user to add, copy, update or view attachments by selecting an icon, button, or link typically used to navigate to the Attachments user interface.

For example, by selecting the Add icon, an Inline Add Attachments pop-up window appears that allows you to add an attachment without navigating to the traditional Attachments User Interface.

The Add icon:



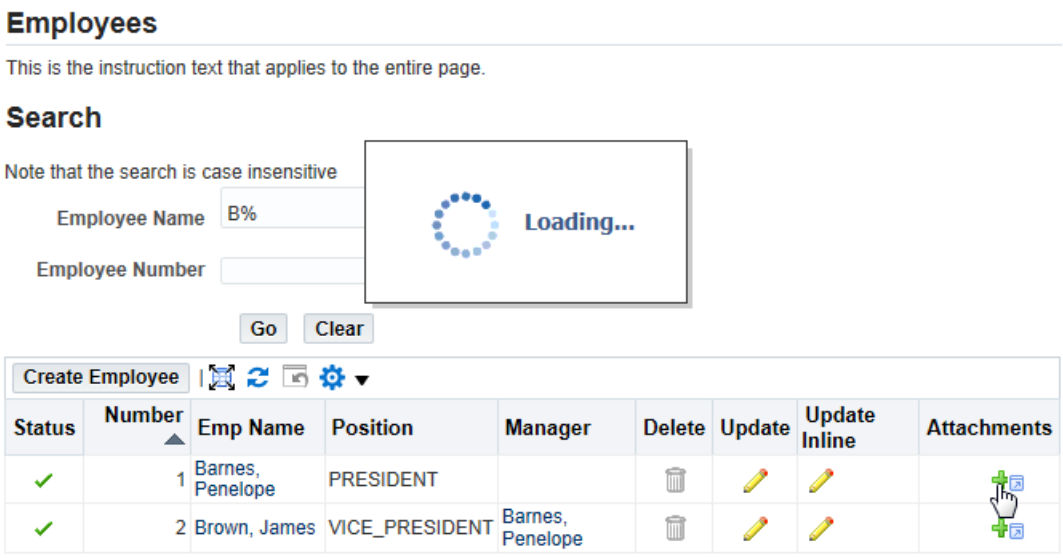
Similarly, by selecting the Attachments icon, an Inline View Attachments pop-up window appears that allows you to view, update or delete attachments inline.

The Attachments icon:



As of Release 12.2, the following loading indicator appears when you trigger the inline attachment pop-up, signifying background processing as the pop-up loads. This indicator also appears when you perform other actions such as Add, Update or Delete within an inline attachment pop-up, as processing occurs.

Figure 12. Loading indicator.



The inline pop-up window appears to the bottom and right of the item that triggers the pop-up, if there is enough browser window space. If there is insufficient space to the right, the pop-up window renders to the left of the item. If there is insufficient space at the bottom, the pop-up window renders to the top of the item.

Note: This feature is supported by Oracle E-Business Suite Release 12.1.2 and higher, and by a subset of the browsers certified with Release 12. For Microsoft Internet Explorer (IE), this feature is only supported in version 7.0 and higher. Refer to "Recommended Browsers for Oracle E-Business Suite Release 12", My Oracle Support (formerly Oracle *MetaLink*) Knowledge Document 389422.1 for additional information.

Enabling Inline Attachments

As of Release 12.2.5, Inline Attachments can appear in all modes set by the profile option **Self Service Accessibility Features**.

To enable Inline Attachments, refer to the Declarative Implementation section that describes how to enable attachments for an entity.

Inline Attachments Support

In Oracle E-Business Suite Release 12.1.2 and above, the Inline Attachment feature is supported on all four attachment styles and for all three attachment types (Text/ Url/ File). In Release 12.2.2 and above, the Inline Attachment feature is supported for Long Text attachment types as well.

Table 1. Summary of the support of the Inline Add Attachment pop-up window and the Inline View Attachments pop-up window for the different attachment styles.

Attachment Item Style	Inline Add Attachment	Inline View Attachments
Attachments Column in a Multi-Row Table Region (attachmentImage)	Supported	Supported
View List Link and Add Button in a Single Row Region (attachmentLink)	Supported	Supported
Inline Links of Attachments in a Single Row Region (messageInlineAttachment)	Supported	Supported
Attachments Table on the Page of a Single Row Region (attachmentTable)	Supported	Supported (for Text-type attachments only)

Table 2. List of events that trigger the Inline Add Attachment and Inline View Attachments pop-ups for the four attachment styles.

Attachment Item Style	Inline Add Attachment	Inline View Attachments
attachmentImage	Select **Add** (**Plus**) icon	Select **Attachment** (**Paper Clip**) Icon
attachmentLink	Select **Add** button	Select **Attachments List** link
messageInlineAttachment	Select **Add Attachment** link in the inline **View Attachment** pop-up	Select **More...** link

attachmentTable	Select **Add Attachment** button	Select **Title** link of Short Text or Long Text-type attachments only.

Inline Add Attachment Pop-up Window

When a user selects an icon, button or link that activates the Inline Add Attachment feature, as described above, an Inline Add Attachment pop-up window appears, displaying the following:

- **Attachment Type** poplist
- **Title** text input field
- **Category** poplist

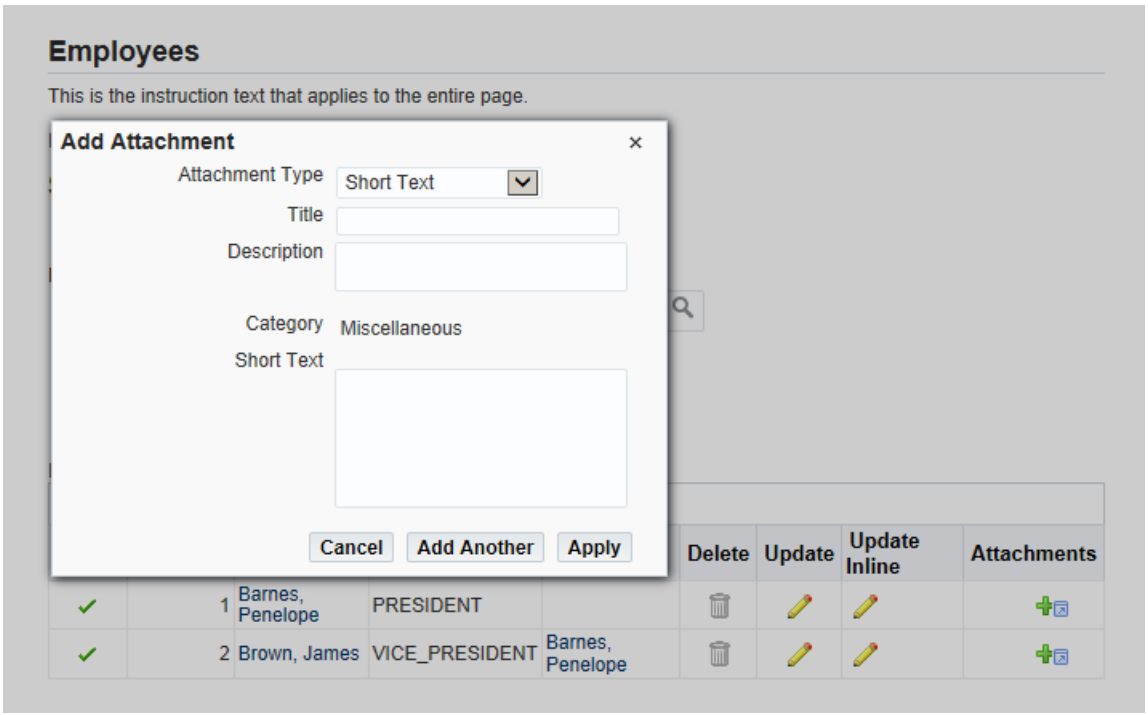
Based on the **Attachment Type** that a user selects, one of the following also renders:

- Text area
- Url text input field
- File-upload item

Note: If you specify a file-type attachment but do not specify a title for the attachment, the filename displays as the title instead. If the filename exceeds thirteen characters, then the first eleven characters followed by an ellipsis (...) display, unless the category of the attachment is an attachment table, then the full filename displays.

The default **Attachment Type** is Short Text. The figure below shows an example of an Inline Add Attachment pop-up window with the **Attachment Type** set to **Short Text**.

Figure 13. Add Attachment pop-up.



The bottom of the Inline Add Attachment pop-up window also displays **Save**, **Add Another** and **Cancel** buttons. Select the **Cancel** button or the **Close** icon to dismiss the Inline Add Attachment pop-up window. Select the **Save** button to post the entered data to the middle-tier and close the Inline Add Attachment pop-up window. Select the **Add Another** button to post the data to the middle tier and refresh the pop-up window for a new entry. The refreshed Inline Add Attachment pop-up window retains the **Attachment Type** last selected.

Note: Refer to Caveats on the iPad Platform in the Mobile Applications topic for more information about inline attachments behavior on the Apple iPad platform.

Note: Refer to AntiSamy Support and Attaching HTML Files for additional requirements when attaching an HTML file.

Attachment Types

As of Release 12.2.6, for the inline Add Attachment UI, you can set the default attachment type to **Long Text**, **Short Text, URL** or **File** during design time, programmatically at runtime, or by using the personalization user interface (UI). If you do not set a default attachment type using any of these methods, then the default attachment type automatically sets to **File**.

Starting in Release 12.2.6, you can also restrict the attachment types for an attachment component during design time, programmatically at runtime, or by using the personalization UI.

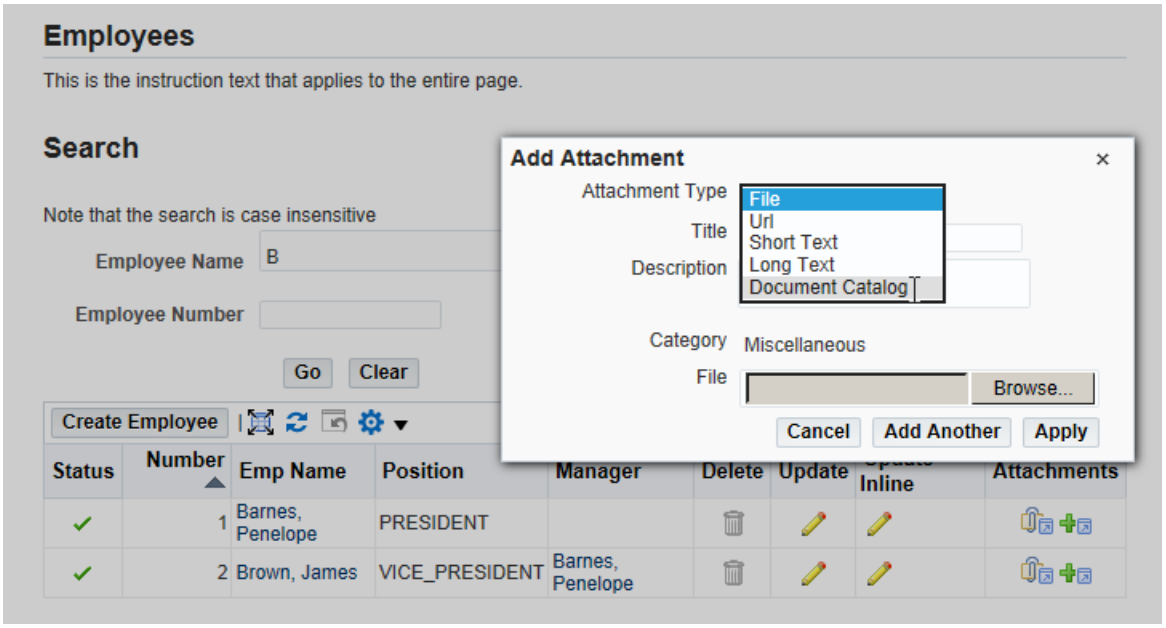
Warning: During design time, you are not allowed to restrict (disable) the attachment type that you previously set as the default attachment type. Any attempt to do so will result in a validation error. Similarly, if you attempt to set such a faulty configuration programmatically or by using the personalization UI, the default attachment type, which is considered enabled, will appear with a relevant developer mode exception that looks as follows:

```
"Developer Mode Error : The type '&ATTACH_TYPE' selected as Default Attachment Type for
attachment bean &BEAN_NAME is disabled. However, to show the Default Attachment Type, it has
now been set as enabled."
```

Adding an Attachment From the Document Catalog Using the Inline Add Attachments Pop-up Window

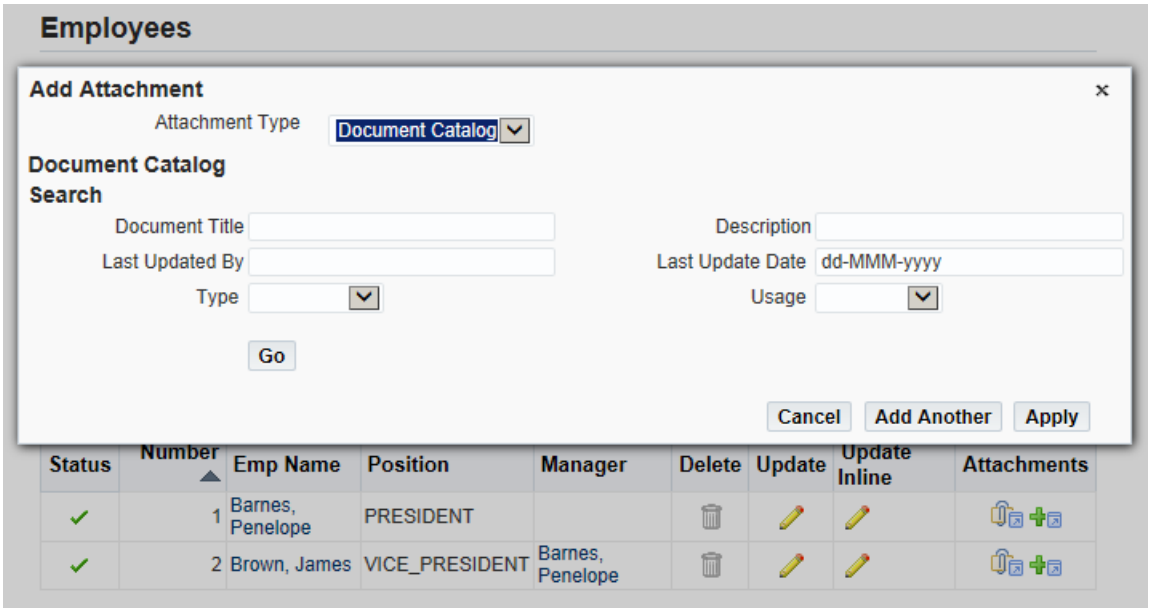
To add an attachment from the Document Catalog, select **Document Catalog** from the **Attachment Type** field.

Figure 14. Adding an attachment from the Document Catalog.



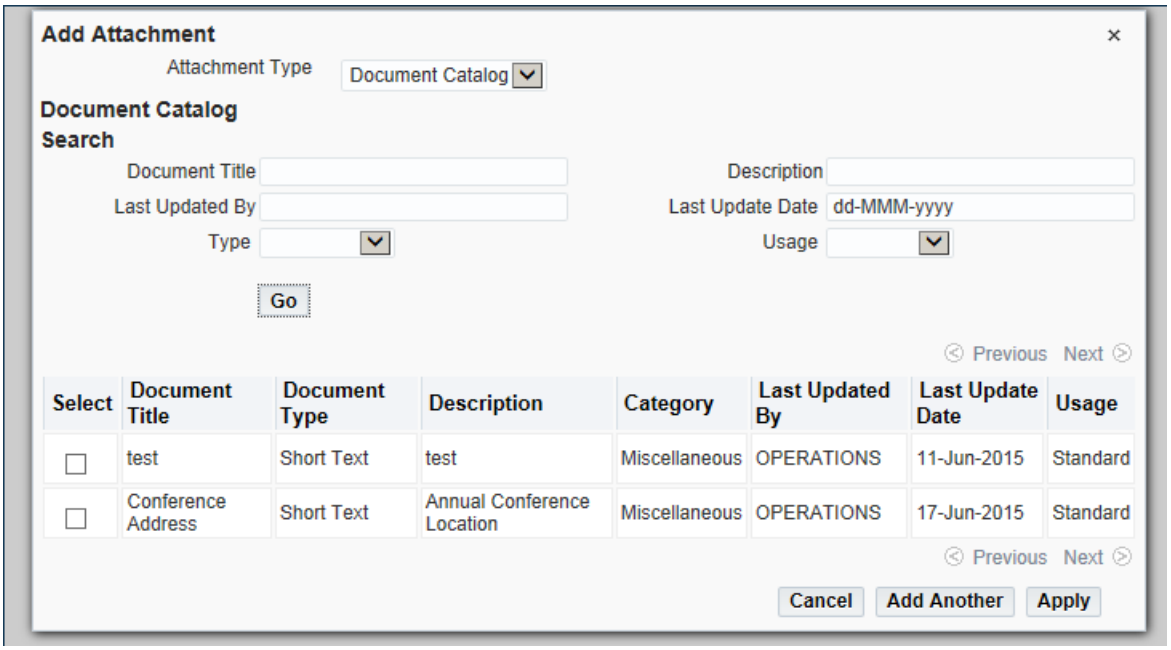
Specify search criteria to identify the attachment you wish to add from the document catalog.

Figure 15. Search criteria for the Document Catalog.



Select the desired attachment to add from the Document Catalog search results.

Figure 16. Document Catalog search results.



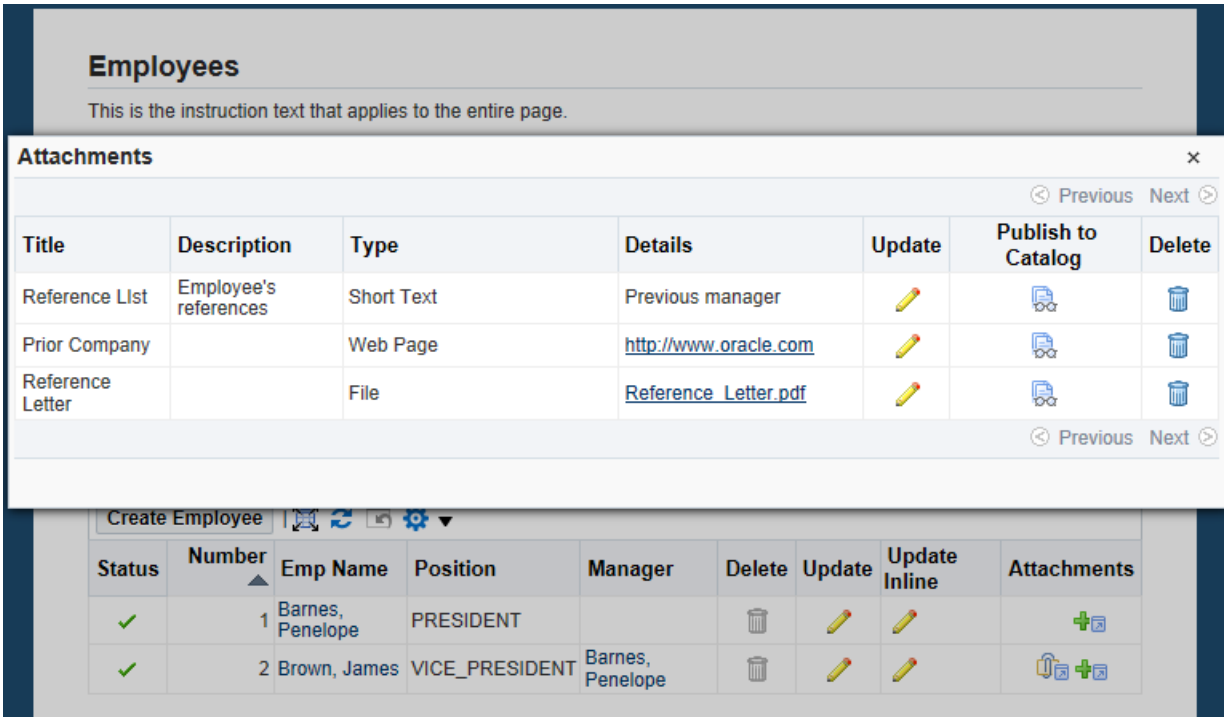
Inline View Attachments Pop-up Window

When a user selects an icon or link that activates the Inline View Attachments feature, as described above, an Inline View Attachments pop-up window appears with a table displaying the current record's attachments. If there are more than 99 attachments, the attachment count is shown as 99+.

The table displays the following columns:

- Title
- Attachment Type
- Details
- Update (icon)
- Delete (icon)

Figure 17. Example of an Inline View Attachments pop-up window for multiple attachments.



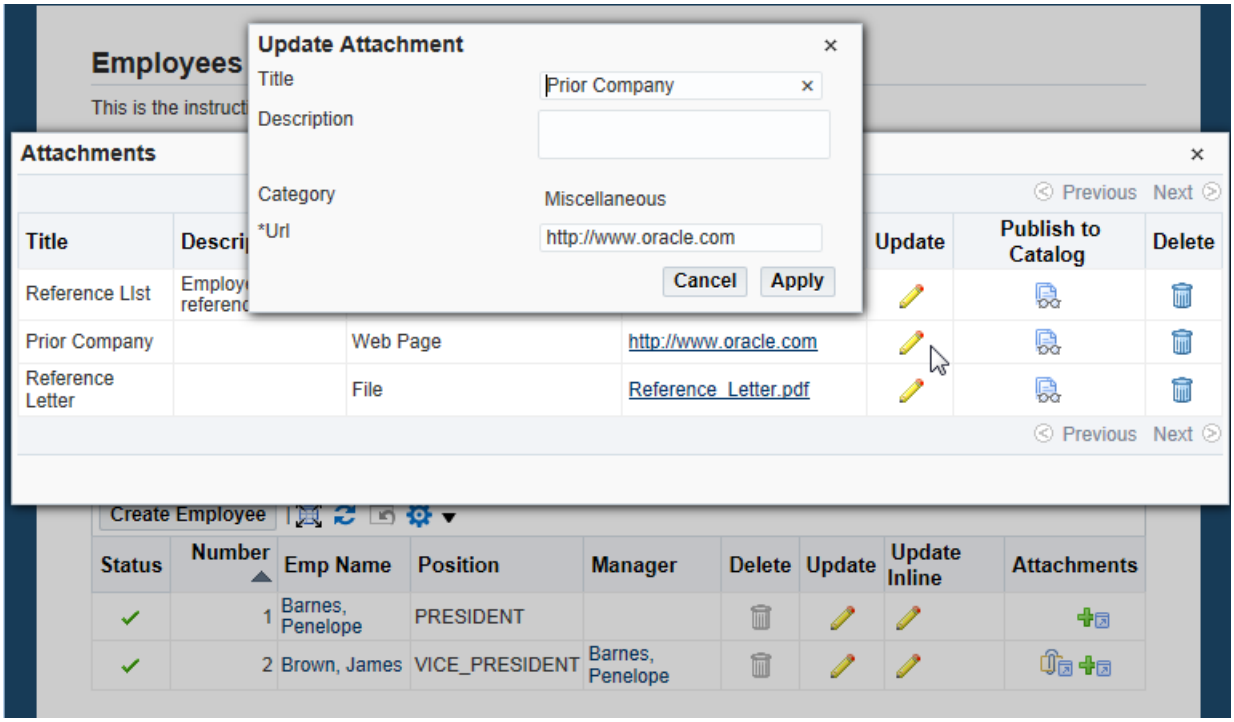
For Text-type attachments, the **Details** column displays a fixed number of characters. If more characters exist, an ellipsis (…) and a **More** link follows the displayed text. Select the **More** link to display the complete text in a separate pop-up window.

For URL-type attachments, select the URL to render the contents of the URL in a new window.

If the attachment item is update-enabled, an enabled **Update** (pen) icon appears. Select the **Update** icon to launch the Inline Update Attachment pop-up window. The pop-up window displays the editable title and details of the selected attachment. Note that the attachment type cannot be modified in this update window. Select the **Cancel** button or the **Close** icon to dismiss the Inline Update Attachment pop-up window. Select the **Update and Close** button to post the data to the middle-tier and close the Inline Update Attachment pop-up window.

Note: If you specify a file-type attachment but do not specify a title for the attachment, the filename displays as the title instead. If the filename exceeds thirteen characters, then the first eleven characters followed by an ellipsis (...) display, unless the category of the attachment is an attachment table, then the full filename displays.

Figure 18. Example of the Update Attachment pop-up window.



If the attachment item is delete-enabled, an enabled **Delete** (trash can) icon appears. Select the **Delete** icon to delete an attachment. A dialog box appears to confirm the delete action. Select **OK** on the confirmation dialog to delete the attachment and re-render the Inline View Attachments pop-up window without that attachment. If you delete the only remaining attachment in the pop-up window, the Inline View Attachments pop-up window no longer appears.

The Inline View Attachments pop-up window can render five attachments at a time. Use the **Next** and **Previous** links to navigate to other attachments.

For the Message Inline Attachment Item Style

For inline links of attachments in a single row region (the messageInlineAttachment item style), when you select the **More...** link, the Inline View Attachments pop-up window that appears also displays an **Add Attachment** link as shown in the figure below. Selecting the **Add Attachment** link displays the Inline Add Attachments pop-up window.

Figure 19. Example of a View Attachments pop-up from the More... link in a single row region containing inline links of attachments.

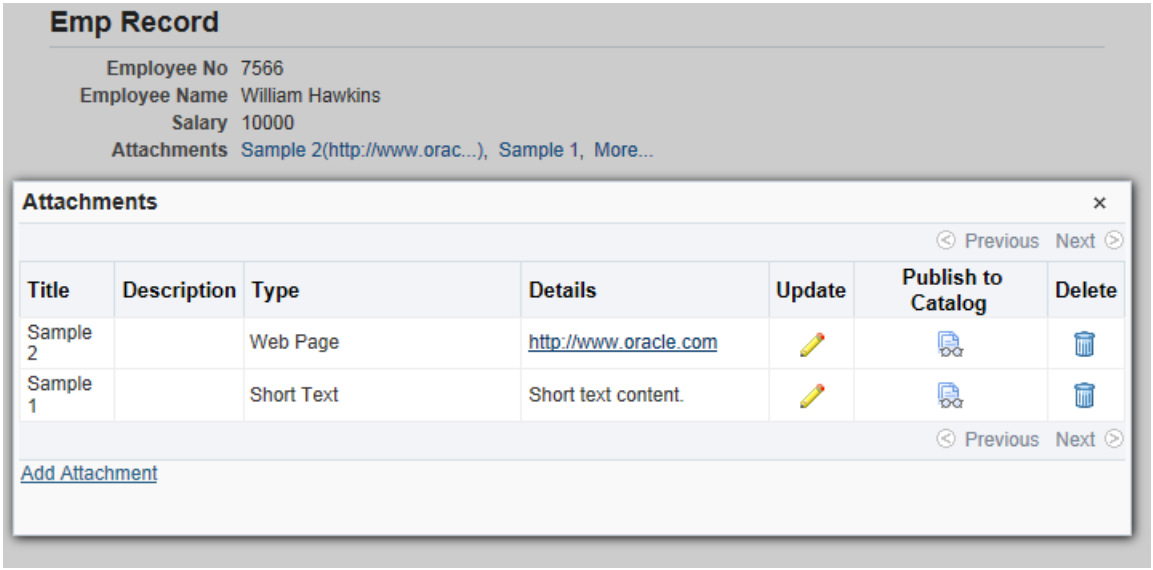
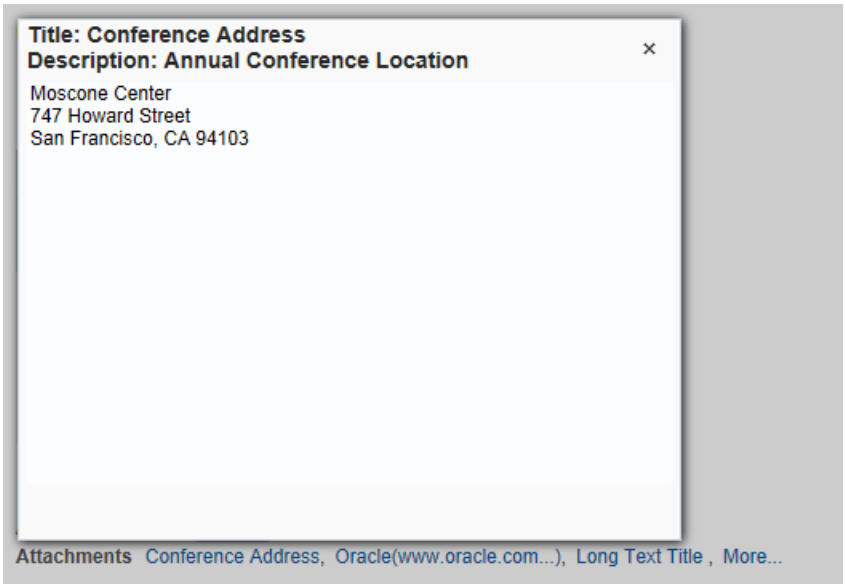


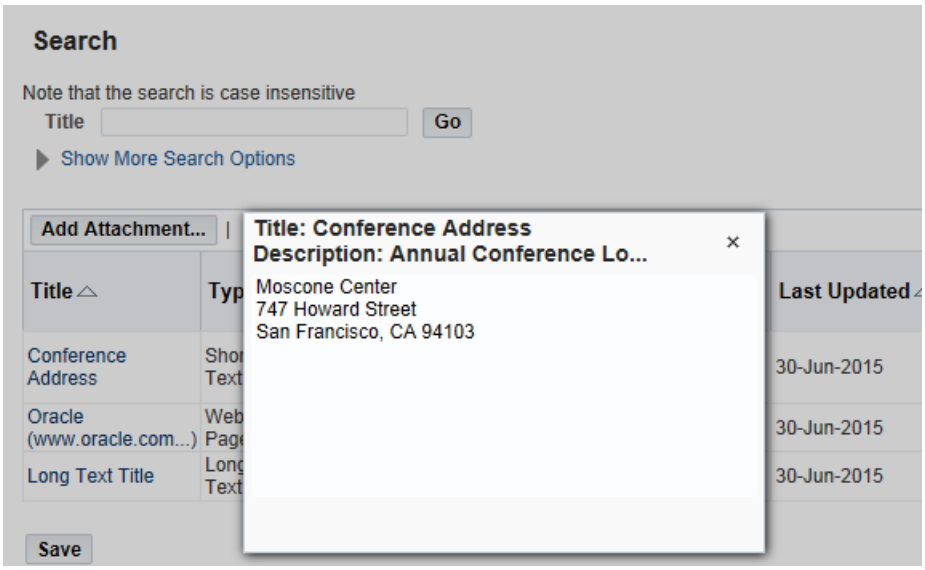
Figure 20. Example of a View Attachments pop-up from an inline link of an attachment in a single row region.



For the Attachments Table Item Style

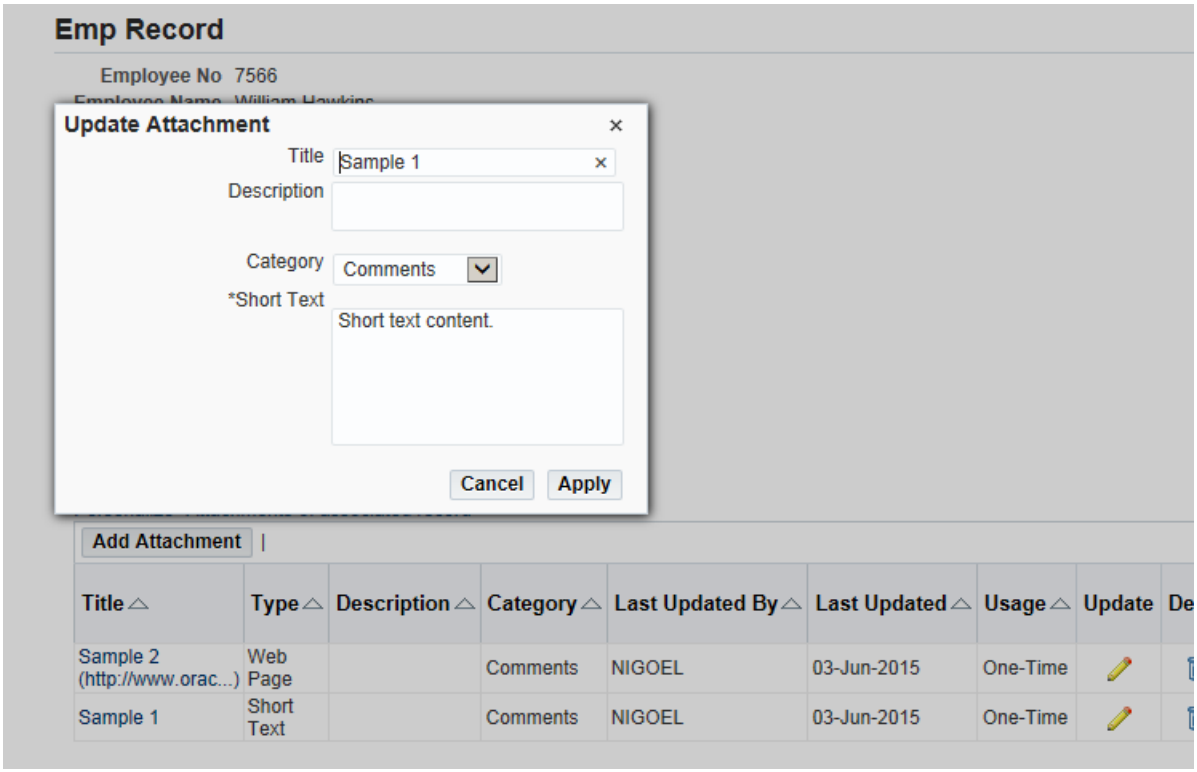
For an Attachments table in a single row region (the attachmentTable item style), when you select a **Title** link for a Text-type attachment record, a pop-up window with the content of that attachment displays.

Figure 21. Example of an View Attachments pop-up when you select a **Title** link for a Text-type attachment record from an Attachments table in a single row region.



When you select the **Update** icon for an attachment record in the table, an Inline Update Attachment pop-up displays as shown in the figure below, allowing you to update the attachment, regardless of type.

Figure 22. Update Attachment pop-up.



When you select the **Delete** icon for an attachment record, a dialog box displays asking you to confirm whether to delete the attachment.

Note: Refer to Caveats on the Apple iPad in the Mobile Applications topic for more information about inline attachments behavior on the Apple iPad platform.

Inline Attachments Accessibility Considerations

As of Release 12.2.5, Inline Attachments can appear in all modes set by the profile option **Self Service Accessibility Features**.

All enabled Inline Attachment pop-ups are focusable and are accessible by the [Tab] key on the keyboard. They may also be closed by pressing the [Esc] key. See the Pop-up topic for more general information about pop-up accessibility considerations.

Enabling the Attachments Feature for an Entity

Declarative Implementation

The Attachments page is rendered using standard OA Framework components. There is no programming required to enable attachments. You simply need to define your attachment region item and the relationship between the item and the entity via OA Extension. Be sure to follow the OA Framework naming standards as you define your OA Extension components.

Step 1: Define the view object for your application.

Step 2: Using OA Extension, define a region that is to be attachments-enabled.

Note: Make sure your region is enclosed in a form and has a **Submit** button. Changes to attachments data can only be committed with a form submit.

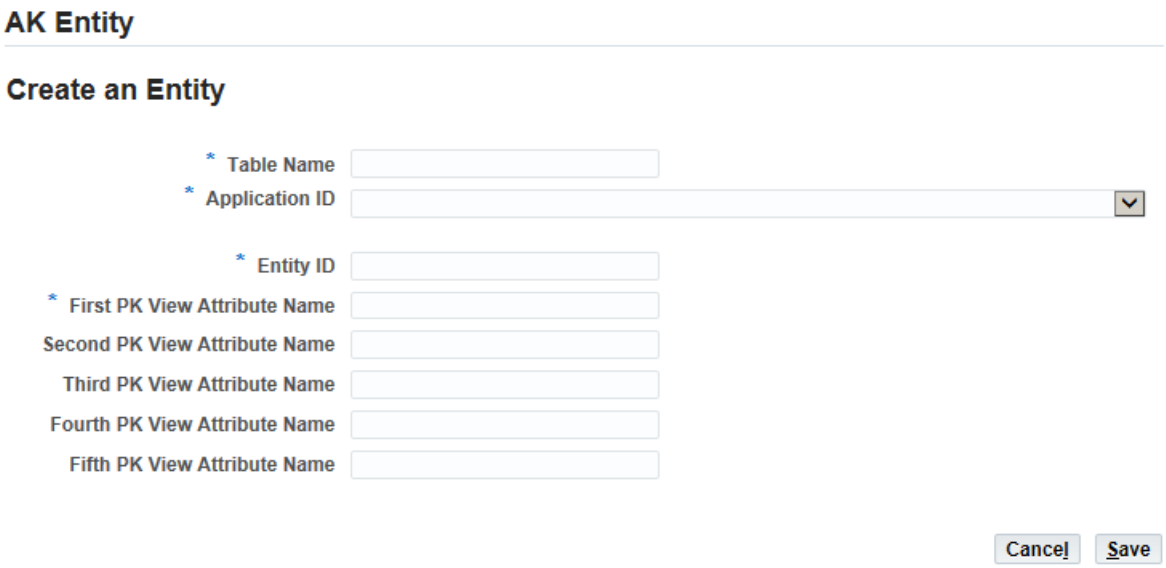
Step 3: Define your region items and for each region item, enter the appropriate view attribute name defined by your view object.

Step 4: If you are enabling attachments only for the OA Framework interface, skip to Step 5.

If you are an Oracle E-Business Suite Division developer and you want to enable attachments for both the Oracle Forms and the OA Framework interfaces, you need to specify your entity information in the `FND_DOCUMENT_ENTITIES` table so that the entity can be shared by both interfaces.

To define your Entity information in the `FND_DOCUMENT_ENTITIES` table, use the **Create an Entity** region in the **AK Entity** page under the **AK HTML Forms** responsibility in E-Business Suite:

Figure 21. AK Entity page.



a. Enter the main table name and Application ID that your view object references. The table name is just informative for users. OA Framework does not actually use this piece of information.

b. Enter the Entity ID. This is used to identify your entity and you need to reference this when you define your region item in OA Extension.

c. From Step 3, enter the view attribute names for the region items that you defined in OA Extension that can be used as primary keys. Note that the view attribute names that you enter into the Entity definition need to be able to uniquely identify a record of the specified entity.

Note: The E-Business Suite interface for defining entities in the **AK Entities** form is at: `OA.jsp?akRegionCode=AK_ENTITY_VIEW_PAGE&akRegionApplicationId=601`

Note: If you want to download the entities you define in `FND_DOCUMENT_ENTITIES`, you need to use `FNDLOAD`. The usage of `FNDLOAD` is as follows:

```
FNDLOAD <username> 0 Y mode configfile datafile [ entity [ param ...] ]
ORACLE password:
```

where

- `username` - APPS username
- `mode` - UPLOAD or DOWNLOAD
- `configfile` - configuration file
- `datafile` - data file
- `entity` - an entity name or - to specify all values in an upload
- `param` - a NAME=VALUE string used for parameter substitution

Step 5: If your region is a single row region, create a nested item with one of the following item styles in OA Extension:

- **attachmentTable** - to render an Attachments table below the current region.
- **attachmentLink** - to render the **View List** link and **Add** button below the current region. Users need to select the **View List** link to display the Attachments page.
- **messageInlineAttachment** – to render a list of inline attachment links. Users can select the **More…** link to navigate to the Attachments user interface to perform further operations on attachments.

  Note: The **messageInlineAttachment** item style should only be added to layout style containers such as messageComponentLayout, header, and so on. It should not be added to any table family members, such as table, advancedTable, HGrid or gantt.

  Note: Refer to the Page Layout topic for guidance on how to place an **attachmentLink** or **messageInlineAttachment** item style in a messageComponentLayout region so that its prompt aligns properly with the other elements of the layout region.

If you have a multi-row table region, define a region item with the item style **attachmentImage** nested in the table or in the column of an advanced table. This creates an attachment column in the table. In the attachment column, you select the **Attachment** icon to display the Attachments page or **Add** icon to display the Add Attachment page.

Step 6: Specify the following properties for the attachment region item you just created:

- `ID` - a unique ID for the attachment region item.
- `View Instance` - the name of the view instance for the entity you are attachment-enabling.
- `Rendered` - set to **True**.
- `Default Attachment Type` - beginning in Release 12.2.6, set the default attachment type that appears in the inline Add Attachment pop-up window to **shortText**, **longText** or **url**. The default is **file**.
- `Enable Document Catalog` - set to **True** to enable adding attachments from the document catalog or **False** to disable access to the document catalog. The default is **True**. Refer to the Attachments Page and Adding Attachments sections for further details.
- `Enable File Attachment` - beginning in Release 12.2.6, set to True to allow file attachments.
- `Enable Text Attachment` - beginning in Release 12.2.6, set to True to allow text attachments.
- `Enable URL Attachment` - beginning in Release 12.2.6, set to True to allow URL attachments.
- `Render Search Region` - set to **True** to render the Search region for the Attachments page or region. The Search region allows you to query for a specific set of attachments.

  Warning: During design time, you are not allowed to restrict (disable) the attachment type that you previously set as the default attachment type. Any attempt to do so will result in a validation error.

If the item style is **attachmentLink**, set these additional properties:

- `Prompt` - a label for the attachment region item that appears before the attachment link. No default prompt appears if this property is null.
- `Link Text` - text for the **Attachment** link, if you select **attachmentLink** as the item style. If this property is null, the link text defaults to "View".

If the item style is **attachmentTable**, set these additional properties:

- `Text` - a display name for the Attachments table.
- `Icon URI` - a gif file containing the image to display next to the Attachments table display name.
- `Copy Attachment Enabled` - optionally set the property to **true** to display a **Copy** column in the Attachments Table. When a user selects the **Copy** icon, an event fires for the corresponding attachment row, allowing you to programmatically get the values related to the attachment and copy them to a new entity. Refer to Copying an Attachment in the Runtime Control section for further details.

If the item style is **messageInlineAttachment**, set these additional properties:

- `Prompt` - a label for the attachment region item that appears before the inline attachment links. No default prompt appears if this property is null.
- `Links Displayed` - the maximum number of inline attachment links to display. The default is 5.

If the item style is **attachmentImage**, set this additional property:

- `Prompt` - a label for the attachment column. If this property is null, the prompt defaults to "Attachments".

Step 7: By default, changes made to the Attachments table or page are saved (committed) only when a user selects the **Apply** button on the base page from which Attachments is launched. See the Runtime Control section.

If you want changes to the Attachments table or page to commit automatically, without requiring a user to select **Apply** in the base page, you can turn on "auto-commit" by setting the `Automatic Save` property to **True** for the attachmentLink, attachmentTable, or attachmentImage item.

With "auto-commit" turned on, each action ("Add", "Update", "Detach") performed by the user in the Attachments table or page is automatically committed.

Step 8: In the Structure pane, select the default entityMap that is created under the attachment region item you just created in the previous step. Select the entityMap in the Structure pane. In the OA Extension Property Inspector, enter a unique ID for the entity map and one of the following values for the `Entity` property:

- If you performed Step 4 and defined an entity in the **AK Entity** form to share between the Oracle Forms and OA Framework interfaces, specify the Entity ID you defined in the **AK Entity** form.
- If you skipped Step 4 because you are enabling attachments only to be viewed from the OA Framework interface, enter a unique arbitrary value for Entity. Select the Entity Map named child in the Structure pane. Under **New** in the context menu, select **primaryKeys** to create a primary key named child for the entity map. Select the primaryKey named child in the Structure pane. Enter a view attribute name for the primary key in the OA Extension Property Inspector.

Note Many regions can share an entity; in fact, if you want the same attachment(s) to be viewable from different regions, those regions *must* share the same entity. In addition, the attachment region items of those different regions *must* have the same view attribute name and reference the same view instance.

Step 9: If you want to make the Attachments table or page read-only, so that the Update and Delete icons are disabled in their respective columns, select the Entity Map named child for the attachment region item in the OA Extension Structure pane. Set the

`Insert Allowed`, `Update Allowed`, and `Delete Allowed` properties in the Property Inspector to **False**. Set any combination of these properties to **False** to better control the level of change you want users to have in the Attachments table or page.

Note: The OA Framework `oracle.apps.fnd.framework.webui.beans.layout.OAAttachmentTableBean setUpdateable(boolean updateable)` method (which is specified by `setUpdateable` in interface `oracle.apps.fnd.framework.webui.beans.OAWebBeanAttachment`), to make an Attachments table read-only, has been deprecated. Use the `oracle.apps.fnd.framework.webui.beans.OAAttachmentImageBean setEntityMappings(Dictionary[] entityMaps)` method instead if you need to maintain backwards compatibility.

Step 10: Create a category map to specify the category of documents that can be associated with an attachment. Document categories provide security by restricting the documents that can be viewed, added or deleted as an attachment. In the Structure pane, select the entity map you just defined. Select **New > categoryMap** from the context menu to create a Category Map named child.

Step 11: Select the new categoryMap entity in the Structure pane. In the Property Inspector, specify values for the following properties:

- `ID` - specify a unique ID for the category map.
- `Category` - specify the document category that can be associated with the attachments. Note that the value of `Category` must be a predefined Name (internal name, rather than display name) from the **FND_DOCUMENT_CATEGORIES** table, such as **MISC**.
- `Secure` - specify **True** to secure the category using Application Security. By securing a category, the documents of that category will only be accessible to users based on the security rules defined in Oracle E-Business Suite. To secure a category, refer to the Securing Attachment Categories section for additional information. By default, `Secure` is set to **False** for backwards compatibility.

Once you define a categoryMap and if the `showAll` property on the **entityMap** is set to false, then only the attachments that are assigned with the same category as specified in your categoryMap(s) can be queried or added from the Attachments page.

Note: A property called `setDefaultMiscCategoryEnabled` creates a default category map and sets the category to MISC, if one is not already defined. As a result, if the `showAll` property on the entityMap is set to **False** and you do *not* define a category map for the entity, OA Framework creates a default category map for the entity and sets the category to MISC. If the `showAll` property is set to **True**, however, it disregards the category maps, as all categories are available.

If you change the `setDefaultMiscCategoryEnabled` property to **False**, and you do *not* create a category map before running your attachment-enabled page or the category map has only one category defined, the Category poplist does *not* display in the Search region of the Attachments table or in the Add Attachment page.

This property is set by default to **True**. To change the value of this property to **False**, you *must* use the following API:

```
attachBean.setDefaultMiscCategoryEnabled(false);
```

Step 12: To enable the Inline Attachments feature, set the profile option FND: Disable Inline Attachments to **false** at the Site or Application level. The default value at the Site level is **false**.

You can also set the `Inline Attachment Enabled` property of your attachment region item to enable or disable the Inline Attachments feature for that particular attachment item. Valid values for the `Inline Attachment Enabled` property are **true**, **false**, or **default**, where **default** means to default to the value set in the FND: Disable Inline Attachments profile option. Users may also personalize the `Inline Attachment Enabled` property at runtime.

Table 3. Illustrates Inline Attachments behavior for the different combination of values set for the profile option and the attachment item property.

Value of `Inline Attachment Enabled` property	Value of 'FND: Disable Inline Attachments' profile	Behavior prior to Release 12.2.5	Behavior in Release 12.2.5 and above
default	false	1. Mouse hover / tap - Inline Attachment pop-up appears. 2. Mouse click / double-tap / focus + [Enter] - Attachment Page appears.	Mouse click / tap / focus + [Enter] - Inline Attachment pop-up appears. No other interaction is supported.
default	true	Mouse click / double-tap / focus + [Enter] - Attachment Page appears.	Mouse click / tap / focus + [Enter] - Attachment Page appears. No other interaction is supported.

true	false	1. Mouse hover / tap - Inline Attachment pop-up appears. 2. Mouse click / double-tap / focus + [Enter] - Attachment Page appears.	Mouse click / tap / focus + [Enter] - Inline Attachment pop-up appears. No other interaction is supported.
true	true	1. Mouse hover / tap - Inline Attachment pop-up appears. 2. Mouse click / double-tap / focus + [Enter] - Attachment Page appears.	Mouse click / tap / focus + [Enter] key - Inline Attachment pop-up appears. No other interaction is supported.
false	false	Mouse click / double-tap / focus + [Enter] - Attachment Page appears.	Mouse click / tap / focus + [Enter] - Attachment Page appears.
false	true	Mouse click / double-tap / focus + [Enter] - Attachment Page appears.	Mouse click / tap / focus + [Enter] - Attachment Page appears.

You may also set these attachment region item properties to control the behavior of the Inline Attachments feature:

- `Update Allowed` - set to **true** to enable the **Update** icon in the Inline View Attachments pop-up window.
- `Delete Allowed` - set to **true** to enable the **Delete** icon in the Inline View Attachments pop-up window.

Securing Attachment Categories

As mentioned above, you can now set the `secured` property on a category map to **True** to ensure that the category is secured by Application Security. Oracle E-Business Suite provides a seeded 'category' object, along with permissions and permission sets that define the actions (functions) that can be performed on the seeded object. You can then use the seeded object and permission sets to define the security rules for the secured category.

The following object is seeded by Oracle E-Business Suite to enforce data security on attachment categories:

```
OBJ_NAME = FND_DOCUMENT_CATEGORIES
DATABASE_OBJECT_NAME = FND_DOCUMENT_CATEGORIES
PK1_COLUMN_NAME = NAME
```

The following permissions (functions) are seeded by Oracle E-Business Suite to define the securable actions on attachment categories:

```
FND_FORM_FUNCTIONS
  FUNCTION_NAME = "FND_ATTACHMENT_VIEW"
  OBJECT_ID = "FND_DOCUMENT_CATEGORIES"
  TYPE = "SUBFUNCTION"

  FUNCTION_NAME = "FND_ATTACHMENT_CREATE"
  OBJECT_ID = "FND_DOCUMENT_CATEGORIES"
  TYPE = "SUBFUNCTION"

  FUNCTION_NAME = "FND_ATTACHMENT_UPDATE"
  OBJECT_ID = "FND_DOCUMENT_CATEGORIES"
  TYPE = "SUBFUNCTION"

  FUNCTION_NAME = "FND_ATTACHMENT_DELETE"
  OBJECT_ID = "FND_DOCUMENT_CATEGORIES"
  TYPE = "SUBFUNCTION"
```

The following three permission sets are seeded by Oracle E-Business Suite to specify the combination of permissions (functions) necessary to perform a particular role on an object instance. You may add additional permission sets if other combinations of functions are required.

```
MENU_NAME = 'FND_ATTACHMENT_FULL_ACCESS'
  FUNCTION_NAME = 'FND_ATTACHMENT_VIEW'
  FUNCTION_NAME = 'FND_ATTACHMENT_CREATE'
  FUNCTION_NAME = 'FND_ATTACHMENT_UPDATE'
  FUNCTION_NAME = 'FND_ATTACHMENT_DELETE'

MENU_NAME = 'FND_ATTACHMENT_VIEW'
  FUNCTION_NAME = 'FND_ATTACHMENT_VIEW'
```

```
MENU_NAME = 'FND_ATTACHMENT_NO_DELETE'
  FUNCTION_NAME = 'FND_ATTACHMENT_VIEW'
  FUNCTION_NAME = 'FND_ATTACHMENT_CREATE'
  FUNCTION_NAME = 'FND_ATTACHMENT_UPDATE'
```

Using the seeded object and permission sets, define one or more grants to define the rules to secure your attachment category. Refer to the online help for the **Grants** page in the **Functional Administrator** responsibility, as well as the *Oracle E-Business Suite Security Guide* for information on how to define a grant. The section on Application Security in Chapter 3: Menus and Page Security also discusses grants and permissions in more detail.

The following example illustrates two grants that you can create to access the secured attachment category called "To Approver":

```
FND_GRANTS
  GRANTEE_TYPE = GROUP
  GRANTEE_KEY = MANAGER_ROLE
  MENU_NAME = FND_ATTACHMENT_FULL_ACCESS
  OBJECT_ID = FND_DOCUMENT_CATEGORIES
  INSTANCE_TYPE = INSTANCE
  INSTANCE_SET_ID = NULL
  INSTANCE_PK1_VALUE = TO_APPROVER
  CTX_SECGRP_ID = -1
  CTX_RESP_ID = -1
  CTX_RESP_APPL_ID = -1
  CTX_ORG_ID = -1


FND_GRANTS
  GRANTEE_TYPE = GROUP
  GRANTEE_KEY = EMPLOYEE_ROLE
  MENU_NAME = FND_ATTACHMENT_VIEW
  OBJECT_ID = FND_DOCUMENT_CATEGORIES
  INSTANCE_TYPE = INSTANCE
  INSTANCE_SET_ID = NULL
  INSTANCE_PK1_VALUE = TO_APPROVER
  CTX_SECGRP_ID = -1
  CTX_RESP_ID = -1
  CTX_RESP_APPL_ID = -1
  CTX_ORG_ID = -1
```

This example shows that users who sign on as MANAGER_ROLE can view, create, update and delete attachments of category "To Approver", whereas users who sign on as EMPLOYEE_ROLE can only view attachments of category "To Approver".

Note: You can use the `setCategoryMapsForAddAndUpdate` method to override the behavior of this particular example.

Now suppose you have a Requisition Header (ID 100) and it has three attachments as shown in the table below. In OA Extension, the meta data for this Requisition Header attachment item contains two categoryMaps, called "MISC" and "To Approver" and categoryMap "To Approver" is secured.

Table 4. Three attachments associated with the Requisition Header.

Attachment ID	Category ID	Category Name
11	1	MISC
12	33	To Approver
13	33	To Approver

If User A signs on as MANAGER_ROLE, User A would see and have full access to the three attachments indicated in the table above. If User B signs on as EMPLOYEE_ROLE, User B would also see the three attachments above, but would only be able to view and not create, update or delete attachments associated with the "To Approver" category.

If no grant is defined for EMPLOYEE_ROLE, then if User B signs on as EMPLOYEE_ROLE, User B would only see the one attachment associated with the MISC category.

Granting Access to Custom Categories

If you define your own custom categories, you may want to grant access for MANAGER_ROLE to all these custom categories instead of creating a separate grant for each newly defined category. The most convenient way to achieve this by having your System Administrator create an object instance set definition.

Since only a seeded user would have a User ID less than or equal to 2, seeded categories in the FND_DOCUMENT_CATEGORIES table would also have a value in the CREATED_BY column that is less than or equal to 2. Therefore, all custom categories have a CREATED_BY value larger than 2:

```
FND_OBJECT_INSTANCE_SETS_VL
  INSTANCE_SET_NAME = "OA_DOC_CATEGORY_CUSTOM"
  OBJECT_ID = "FND_DOCUMENT_CATEGORIES"
  PREDICATE = "&TABLE_ALIAS.CREATED_BY > 2"

FND_GRANTS
  GRANTEE_TYPE = GROUP
  GRANTEE_KEY = MANAGER_ROLE
  MENU_NAME = FND_ATTACHMENT_FULL_ACCESS
  OBJECT_ID = FND_DOCUMENT_CATEGORIES
  INSTANCE_TYPE = SET
  INSTANCE_SET_ID = OA_DOC_CATEGORY_CUSTOM
  INSTANCE_PK1_VALUE = NULL
  CTX_SECGRP_ID = -1
  CTX_RESP_ID = -1
  CTX_RESP_APPL_ID = -1
  CTX_ORG_ID = -1
```

With this definition, MANAGER_ROLE would have full access to attachments of all custom categories.

Runtime Control

In general, there are no runtime control steps necessary to enable attachments for an entity. However, you can refer to the following sections for instructions on additional programmatic changes you can make to attachments:

- Securing Attachment Documents to a Particular Context
- Securing Categories
- Getting a Handle to an Attachment Event
- Controlling When to Commit Changes
- Creating or Updating an Attachment
- Copying an Attachment
- Hiding Columns in an Attachment Table
- Enabling Inline Attachments
- Displaying the Attachments Table on the Page of a Single Row Region
- Setting the Default Attachment Type
- Restricting Attachment Types

Securing Documents to a Particular Context

To secure your attachment documents with a particular context, such as organization, set of books, or business unit, you can do so using these two APIs in the `oracle.apps.fnd.framework.webui.beans.OAWebBeanAttachment` interface:

- `setSecurityType(int securityType)`
- `setSecurityId(int securityId)`

Typically, attachments in financial applications are secured by sets of books, attachments in manufacturing applications are secured by organization, and attachments in human resource applications are secured by business unit ID.

The following example illustrates how to use these APIs to secure documents by organization ABC. This should be included in the controller code for the page that launches the Attachments feature:

```
attachmentBean.setSecurityType(1);
// security type 1 = Organization, 2 = Sets of Books,
// 3 = Business Unit, and -1 = No security type

attachmentBean.setSecurityId(123);
// org id: 123 = company "ABC"
```

As a result of the above code, when a user creates an attachment for the application page, the document is automatically secured to organization ABC. The Attachments UI displays documents that have this security context, as well as documents that are not secured.

If you do *not* set a security context when you enable the Attachments feature for a page, the attachment documents that get created will have a security type of "None" and can be shared across different security context.

Securing Categories

If you need to secure an attachment category programmatically, you can use the `setCategorySecured` method on `OAAttachmentImageBean`, `OAAttachmentTableBean`, `OAMessageAttachmentLinkBean`, or `OAMessageInlineAttachmentBean`.

Getting a Handle to an Attachment Event

If you want to get a handle on an attachment event so that you can add custom logic before leaving the base page, you can do so by checking for the return of any of the following `OAWebBeanConstants` from `UIConstants.EVENT_PARAM`:

- `OA_ADD_ATTACHMENT` - returned when the **Add Attachment** button is pressed from attachment table or when **Add** icon/button is pressed from attachmentImage/attachmentLink style.
- `OA_UPDATE_ATTACHMENT` - returned when the **Update** attachment icon is pressed from attachment table.
- `OA_DELETE_ATTACHMENT` - returned when the **Delete** attachment icon is pressed from attachment table
- `OA_VIEW_ATTACHMENT` - returned when the link for viewing attachment is pressed from attachment table
- `OA_GOTO_ATTACHMENTS` - returned when navigating to Attachments page from the base page. Valid for attachmentImage, attachmentLink, and messageInlineAttachment styles only.

For example, the following code in the `processFormRequest` method of the controller, checks if the **Add Attachment** button has been selected:

```
String eventName = pageContext.getParameter(EVENT_NAME);
// Check if Add attachment button is clicked
if ( OA_ADD_ATTACHMENT.equals(eventName) )
{
// product team's special handling code
}
```

Committing Changes

By default, changes made to the Attachments table or page are saved (committed) only when a user selects the **Apply** button on the product page from which the Attachments flow is launched. The Automatic Save property on the attachment web bean is set to **False** in this case. This means that in the `processFormRequest` method of the controller object, you need to capture the event of this submit button and call `commit()` explicitly.

If you want changes to the Attachments table or page to commit automatically, without requiring a user to select **Apply** in the base product page, you can turn on "auto-commit" by setting the `Automatic Save` property on the attachment web bean to **True**.

With "auto-commit" turned on, each action ("Add", "Update", "Delete") performed by the user in the Attachments table or page is automatically committed.

Creating or Updating an Attachment

As of Release 12.1.3, you can programmatically create or update an attachment using these `oracle.apps.fnd.framework.server.OAAttachmentServerUtils` APIs:

- ```
  createAttachment(String input, String[] pkValues, String categoryName,
      String entityId,String contentType, BlobDomain fileContent, String title,
      String description, OAApplicationModule am, String automaticallyAddedFlag,
      String attachmentType)
  ```

- ```
  updateAttachment(String input, String[] pkValues,String categoryName,
      String entityId, String contentType,BlobDomain fileContent, String title,
      String description, OAApplicationModule am, String automaticallyAddedFlag,
      String attachmentType, long attachmentDocumentId)
  ```

Copying an Attachment

As of Release 12.2, you can enable the ability to copy an attachment from one entity to another from the Attachment Table in a single row region. After you enable the **Copy** column in the Attachment Table, a user may select the **Copy** icon to fire an event that passes the following parameters for the corresponding attachment row:

- `Mode`
- `EntityID`
- `Pk1Value`
- `Pk2Value`
- `Pk3Value`
- `Pk4Value`
- `Pk5Value`
- `AttachedDocumentId`
- `MediaId`
- `DatatypeId`
- `DocumentId`

Programmatically get the values of these parameters for the attachment row by using `OAPageContext` in the

`processFormRequest` method of the controller object. For example, the following code in the `processFormRequest` method gets the value of the `Mode` parameter (where `pageContext` is an object of the `OAPageContext` interface):

```
String mode = pageContext.getParameter(Mode);
```

Handle the captured parameter values accordingly by using the following FND PL/SQL APIs to copy the attachment to another entity:

```
fnd_attached_documents2_pkg.copy_attachments (AFAKATDB.pls)
fnd_documents_pkg.insert_row (AFAKADCB.pls)
fnd_attached_documents_pkg.insert_row (AFAKAADB.pls)
```

Call the `fnd_attached_documents2_pkg.copy_attachments` with a new argument, `x_orig_attach_doc_id`. By passing the argument `x_orig_attach_doc_id`, you limit the copying to a single specified attachment and document. If you do not specify a value for this argument, the API copies ALL associated attachments and documents for the entity. This API stores the originating attached_document_id and document_id (`orig_attach_doc_id` and `orig_doc_id`, respectively) in their corresponding `fnd_attached_documents` and `fnd_documents` tables.

Refer to the "FNDSQF Routine APIs" chapter of the *Oracle E-Business Suite Developer's Guide* for more information about these APIs.

Hiding Columns in an Attachment Table

As of Release 12.2.4, you may programmatically hide the attachment table columns listed below, in a web bean of style attachmentTable or in an Attachments page that includes an attachmentImage, attachmentLink or messageInlineAttachment item:

- Category
- Last Updated By
- Last Updated
- Usage
- Update
- Delete
- Print Attachment

  Note: By setting the profile option FND Attachment AutoVue Server to disable the Print Attachment column, you disable the Printing icon for all the user's attachment items. The programmatic approached described below allows you to disable printing for a particular item.

To hide any of these columns, set the options listed below by editing the `processRequest(pageContext, webBean)` of the attachment web bean controller (for `OAMessageInlineAttachmentBean`, `OAAttachmentTableBean`, `OAAttachmentImageBean`, or `OAMessageAttachmentLinkBean`):

```
attachmentBean =
(OAMessageInlineAttachmentBean)webbean.findChildRecursive("attachment_item_ID");

attachmentBean.put(ATTACH_DELETE_COLUMN_RENDERED, Boolean.FALSE);
//to hide Delete column
attachmentBean.put(ATTACH_UPDATE_COLUMN_RENDERED, Boolean.FALSE);
//to hide Update column
attachmentBean.put(ATTACH_PRINT_COLUMN_RENDERED, Boolean.FALSE);
//to hide Print Attachment column
attachmentBean.put(ATTACH_USAGE_COLUMN_RENDERED, Boolean.FALSE);
//to hide Usage column
attachmentBean.put(ATTACH_LAST_UPDATED_COLUMN_RENDERED, Boolean.FALSE);
//to hide Last Updated column
attachmentBean.put(ATTACH_LAST_UPDATED_BY_COLUMN_RENDERED, Boolean.FALSE);
//to hide Last Updated By column
attachmentBean.put(ATTACH_CATEGORY_COLUMN_RENDERED, Boolean.FALSE);
//to hide Category column
```

Note: This feature only hides a column and does not disable the column as in the case of the Delete and Update icons.

Note: OA Framework does not support this feature for inline attachments.

Enabling Inline Attachments

You can set the value of the `Inline Attachment Enabled` property on an attachment region item programmatically by calling the following API, exposed on all attachment component web beans. Passing true to the method enables the Inline Attachment pop-up UI.

```
public void setInlineAttachmentEnabled(boolean enabled)
```

Displaying the Attachments Table on the Page of a Single Row Region

If you want to display the Attachments table on the page, but do not set the `Text` property on the attachmentTable item, then from the controller code of the page where this web bean renders, you should call the `setHeaderDisabled` method on the web bean. This removes the extra white space that gets generated for the header place holder.

Setting the Default Attachment Type

Beginning in Release 12.2.6, you can use the following API to set the default attachment type for the inline Add Attachments pop-up window:

```
public void setDefaultInlineAttachmentType(String attachmentType)
```

Note: The value of the default attachment type set programmatically takes precedence over the value of the default attachment type set declaratively or using the personalization UI.

Restricting Attachment Types

Beginning in Release 12.2.6, you can use the following API to enable or disable a specific attachment type:

```
final public void setDisplayedAttachmentDataTypes(boolean fileType, boolean  shortTextType,
  boolean longTextType, boolean urlType)
```

Note: The value set programmatically takes precedence over the value of the property set declaratively or using the personalization UI.

Note: If you modify an existing attachment web bean to restrict a particular attachment type (by redesigning the page programmatically or through personalization), then any previously added attachments of that newly restricted type will no longer appear in the View Attachments pop-up window.

Warning: You are not allowed to restrict (disable) the attachment type that you previously set as the default attachment type. The default attachment type, which is considered enabled, will appear with a relevant developer mode exception that looks as follows:

```
"Developer Mode Error : The type '&ATTACH_TYPE' selected as Default Attachment Type for
attachment bean &BEAN_NAME is disabled. However, to show the Default Attachment Type, it has
now been set as enabled."
```

Enabling Virus Scanning

As of Release 12.1, you can enable virus scanning on a file type attachment before uploading. To enable virus scanning, set the **`Virus Scan Enabled`** property in OA Extension to **true** for the **attachmentLink**, **attachmentTable**, or **attachmentImage** item.

As of Release 12.2, you can also use the profile option FND: Disable Virus Scan to enable virus scanning. The combination of the **`Virus Scan Enabled`** property and the "FND: Disable Virus Scan" profile option control virus scanning as shown in the table below:

Table 5. Virus scanning as controlled by the FND: Disable Virus Scan profile and the Virus Scan Enabled property.

"FND: Disable Virus Scan" Profile Value	`Virus Scan Enabled` Property Value	Scan Done?
No	Yes	Yes
No	No	Yes
Yes	Yes	Yes
Yes	No	No

You can also control virus scanning at runtime. The `processRequest` of the controller class should include the following code: `attachBean.setVirusScanEnabled(true);` where attachBean is the web bean corresponding to the attachmentLink, attachmentTable, or attachmentImage item.

Note: In order for virus scanning to work properly, the following profiles must also be set appropriately:

- FND Attachment Antivirus Software Type
- FND Attachment Antivirus Server

The anti-virus software repairs the file if a virus is detected before uploading the repaired file. If the file cannot be repaired, an error message displays about the infected file and the file is not uploaded.

Personalization Considerations

See a summary of Attachments personalization considerations in the Oracle Application Framework Personalization Guide.

Services Exposed

Bulk Upload - Overview

The OAAttachmentsService is developed using the 12.1 Business Service Object infrastructure. This allows users to upload attachments in bulk. This service can be searched for from the Integration Repository , which contains WSDL documents for all EBS services. Refer to the "Introduction to SOA and Services" section for an overview of the SOA concepts.

Service Description

Name – OAAttachmentsService.
Note: This is the name to be used to search for the service in the Integration Repository.

This service exposes the processDataSource service API for adding attachments in bulk. To learn more about this, refer to the **Invoking Bulk Upload Operation** section in this document.

Methods - processDataSource.

Service Data Objects - FndAttachBulkUploadVO

Methods

processDataSource

This method performs the Bulk Upload operation and attaches uploaded data with specified business entities. User needs to invoke this method by passing appropriate parameters as mentioned in the table below.

Table 6. Parameters

Parameter Name	DataType	Description
dataSourceName	java.lang.String	The name of the data source to process. In this case, the value for this should be "**FndAttachBulkUploadVO**".
operation	java.lang.String	The operation to perform. Can be one of the following: “Create” or “Merge”.
oaattachmentsprocessControl	/oracle/svc/ProcessControl	control the post control object
ServiceRoot	/oracle/svc/DataObject	The root data object containing a list of data objects to be processed.

Return Type

oracle.svc.DataObject

Note: This is an object of type oracle.svc.DataObject whose qualified name is OAAttachmentServiceRoot which has APIs to get the list of the service data objects returned by the service method.

This operation will return oracle.svc.DataObject instance which contains the list of service data objects of type 'FndAttachBulkUploadVO'.
The bulk upload status can be verified by getting the value of the service data object present in the returned DataObject. For successful attachments it lists the attachment id and for invalid records it lists appropriate error messages.

Service Data Objects

Following Service Data Objects are available in this service:

FndAttachBulkUploadVO

This is the Service Data Object that contains the attributes that need to be input by the user to upload the attachments corresponding to the business entity. The fileContent from this data source will be attached to the specified business entities.

Table 7. Attributes

Attribute Name	Data Type	Description
fileName	java.lang.String	The name of the file to be uploaded. **Note that this is a mandatory field.**
contentType	java.lang.String	The type of file content viz. text/html, image, icon etc This is MIME type and can contain any valid MIME entry.
fileContent	oracle.jbo.domain.BlobDomain	The content of the file.
pk1_Value	java.lang.String	Primary key1 value.
Pk2_Value	java.lang.String	Optional primary key2 value.
Pk3_Value	java.lang.String	Optional primary key3 value.
Pk4_Value	java.lang.String	Optional primary key4 value.
Pk5_Value	java.lang.String	Optional primary key5 value.
categoryName	java.lang.String	The name of the category. This should be a valid value of Category ID from the FND_DOCUMENT_CATEGORIES table.
entityId	java.lang.String	The id of the entity to which the document will be attached.
description	java.lang.String	Optional description of the attachment.
status	java.lang.String	The status of the upload operation for the current object. For successful attachments it lists the attachment id and for invalid records it lists appropriate error messages. This is used only for the output operation and users need to ignore this attribute while invoking the service

Related Documents

- Search services in Integration Repository
- Service Interface in Integration repository
- Invoking service from BPEL
- Web Services and Function Security
- Business Service Objects (BSO)

Known Issues

- See a summary of key Attachment issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline
  - Attachments Templates
  - Attachments Flows
- Javadoc File
  - `oracle.apps.fnd.framework.webui.beans.layout.OAAttachmentTableBean`
  - `oracle.apps.fnd.framework.webui.beans.OAAttachmentImageBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageAttachmentLinkBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageInlineAttachmentBean`
  - `oracle.apps.fnd.framework.webui.beans.OAWebBeanAttachment`
  - `oracle.apps.fnd.framework.webui.OAWebBeanConstants`
  - `oracle.apps.fnd.framework.server.OAAttachmentServerUtils`
- Lesson
- Sample Code

.

Auto-Repeating Layout (ARL)

Overview

You can auto-repeat the layout of children web beans in a container, based on the rows in a view object by using a process called Child View Usage. The process attaches a data source (view object) to a container web bean, which then replicates its children web beans based on the number of data objects or rows in the source. Each replica of the container's children reflects a different row in the data source, similar to what you see when you render a table or advanced table web bean.

Contents

- Auto-Repeating Layout Variations
- Usage Restrictions
- Declarative Implementation
  - Creating a One-Level Auto-Repeating Layout
  - Creating a Two-Level Auto-Repeating Layout
- Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

Auto-Repeating Layout Variations

You can implement an auto-repeating layout in one or two levels:

- **One Level** - you create a container web bean and attach a view object to it, which results in the replication of the container children based on the rows in the view object.
- **Two Levels** - you create a master-detail relationship between two lists and create a view link to join them. The outer list replicates its container children based on the master view object and the inner list replicates its container children based on the detail view object.form

You can also implement an auto-repeating layout as a display-only list, or as a list with form elements:

- **Read Only** - a read-only auto-repeating layout can be based on a single view object or a single view link. In the case of a single view object, a one level list of children are replicated. In the case of a single view link, a composite list results, where a different inner list is created for each current master record.
- **Form Element** - an auto-repeating layout with form elements can be based only on a single view object. A layout with form elements allows you to update any row and submit your changes to update the underlying view object.

Usage Restrictions

- You cannot implement an auto-repeating layout in the following container regions:
  - Table
  - Advanced Table
  - Hide/Show
  - Switcher
  - HGrid
  - SubTabLayout
- You cannot replicate the above container regions indirectly. For example, you cannot create a header region, add a table region beneath it, and expect to replicate the header region (and the underlying Table region). This is not supported.
- You cannot implement a two-level auto-repeating layout with a list containing form elements.
- OA Framework does not support view objects (for the Child View Usage) with composite primary keys. The view object should have a single-column primary key.
- LOVs are not supported in auto-repeating layouts.
- The items under an auto-repeating layout should not be made hidden/rendered on a Partial Page Refresh (PPR) event.
  - If such a scenario exists on a PPR event, the page should be re-rendered by calling `pageContext.setForwardURL()`. This will ensure the items under an auto-repeating layout are rendered with the correct bean IDs.

Declarative Implementation

This section describes how to declaratively implement an auto-repeating layout for your page.

Creating a One-Level Auto-Repeating Layout

Step 1: In your project, create an **application module** for your auto-repeating layout, if you don't yet have one for your page.

Step 2: Create a **view object** for your auto-repeating layout and add it to your **application module**. For example, you can create the following BC4J setup:

- Create an application module.
- Create a view object based on the Dept table, called `DeptVO`.
- `DeptVO` has the primary key '`Deptno`'.

Step 3: Define a page in OA Extension, and create a pageLayout region beneath it. Set the `AM Definition` property on the pageLayout region to the application module you created in Step 1.

Step 4: Create a container region and the children that you want as your auto-repeating layout. Select, in the Structure pane, the container region. For example, suppose you create a header region, with a rowLayout region that contains a messageTextInput item and a submitButton item beneath it and you want the messageTextInput and submitButton web beans to auto-repeat for all the rows returned from an view object. In this case, you would select the rowLayout region.

Step 5: For the selected container region, set the `Child View Instance` property to the name of the view object you created in Step 2 and the `Child View Attribute` property to an attribute in the view object that uniquely identifies a row (the primary key). Based on the BC4J example in Step 2, you would set the `Child View Instance` property to DeptVO and the `Child View Attribute` property to Deptno.

Note In case a view object does not have a primary key, you can use RowId as the `Child View Attribute`.

Creating a Two-Level Auto-Repeating Layout

Step 1: In your project, create an **application module** for your auto-repeating layout, if you don't yet have one for your page.

Step 2: Create a **view object** for your master list, a **view object** for your detail list and a **view link** between these two view objects. Add these view objects and view link to your application module. For example, you can create the following BC4J setup:

- Create an application module.
- Create a view object based on the Dept table, called `DeptVO`.
- `DeptVO` has the primary key '`Deptno`'.
- Create a view object based on the Emp table, called `EmpVO`.
- Create a view link between `DeptVO` and `EmpVO`, called `DeptEmpVL`.
- Add the above view objects and view link to the application module.

Step 3: Define a page in OA Extension, and create a pageLayout region beneath it. Set the `AM Definition` property on the pageLayout region to the application module you created in Step 1.

Step 4: Create the outer container region and its children, and the inner container region and its children, that you want to use for your two-level auto-repeating layout.

Step 5: Select, in the OA Extension Structure pane, the outer container region and set the following properties:

- `Child View Instance` - set to the name of the view object you created in Step 2 for the outer list.
- `Child View Attribute` - set to an attribute in the outer view object that uniquely identifies a row (the primary key).
- `View Link Instance` - set to the view link defined between the inner and outer view objects.

Based on the example in Step 2, you would set the `Child View Instance` to DeptVO, the `Child View Attribute` to Deptno, and the `View Link Instance` to DeptEmpVL.

Note In case a view object does not have a primary key, you can use RowId as the `Child View Attribute`.

Step 6: Select, in the OA Extension Structure pane, the inner container region and set the following properties:

- `Child View Instance` - set to the name of the view object you created in Step 2 for the inner list.
- `Child View Attribute` - set to an attribute in the inner view object that uniquely identifies a row (the primary key).
- `View Link Instance` - set to the view link defined between the inner and outer view objects.

Based on the example in Step 2, you would set the `Child View Instance` to EmpVO, the `Child View Attribute` to Deptno, and the `View Link Instance` to DeptEmpVL.

Note In case a view object does not have a primary key, you can use RowId as the `Child View Attribute`.

Runtime Control

There are no programmatic steps required for implementing an auto-repeating layout region.

The child view usage set on the container web beans is automatically queried when the page renders. If you wish to turn off auto-execution of the query, set the `oracle.apps.fnd.framework.webui.OAWebBeanConstants` attribute, `CHILD_VIEW_AUTO_QUERY_ATTR`, to `Boolean.FALSE`, as illustrated in the following code example:

```
containerBean.setAttributeValue(
  OAWebBeanConstants.CHILD_VIEW_AUTO_QUERY_ATTR, Boolean.FALSE)
```

Personalization Considerations

See a summary of Auto-Repeating Layout personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guidelines
- Javadoc File
  - `oracle.apps.fnd.framework.webui.OAWebBeanConstants`
- Lesson
- Sample Code

Bound Values

Overview

Data binding allows you to map data from UIX web beans to BC4J components and back, bridging the gap between the two so you can create HTML pages that generate dynamic content for each user. As mentioned in the Anatomy of an OA Framework Page, OA Framework web bean attributes are implemented as data bound values, so that the underlying data source (BC4J) is not resolved to the component (by the UIX framework) until rendering time. OA Framework uses UIX's `oracle.cabo.ui.data.BoundValue`, `oracle.cabo.ui.data.DataObject` and `oracle.cabo.ui.data.DataObjectList` interfaces to perform data binding at runtime.

A table content Switcher, on the other hand, is a region with two or more display alternatives. The display alternatives are predefined items of which only one is selectively rendered at any given time.

Bound Values versus Table Content Switchers

You should limit your use of Switchers to within tables, particularly when you want to switch between different kinds of web beans, such as a poplist or a checkbox. When you have only one type of web bean, but the value of an attribute on that web bean varies at runtime, then you should implement that attribute as a bound value rather than as a Switcher.

There are exceptions to this, however, as demonstrated in the ToolBox Tutorial Delete Lab. The tutorial example creates an employee table that contains a Delete column. The Delete column allows you to delete employees from the table, depending on the status of the employee - if the employee is active, the Delete icon is enabled, otherwise it is disabled. However, to meet the standards of 508, alternate text (alt text) is associated with the enabled icon, as well as the disabled icon. At runtime, to be able to display the **enabled** Delete icon, with its alt text, or the **disabled** Delete icon with its appropriate alt text, the tutorial uses the convenience of a table content Switcher to switch between the two distinct sets of attribute values for the same web bean type.

If you were to use bound values instead of a Switcher in this case, you would bind the image source of the Delete icon to a view object attribute to get the image file name, and bind the alt text to another view object attribute to get the alt text for the image.

Note Although you can use a table content switcher outside a table, UIX discourages this. Instead you should bind the **Rendered** property of the indexed child when possible.

Contents

- Bound Values
  - Why Use Bound Values?
  - Data Binding in OA Framework
  - Data Binding for Fields Outside a Table
  - Data Binding for Fields in a Table
- Declarative Implementation
- Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

Bound Values

Why Use Bound Values?

Since it is possible to write custom code to populate the contents of an HTML page, as well as derive UI content from certain metadata specified in the OA Extension Property Inspector, why bother to use bound values? There are several reasons, the main reason being that bound values simplify the code needed to fetch values from a data source at rendering time.

In the case of interdependent attributes, where the value of one attribute depends on the value of another, the use of bound values eliminates the need for a lot of duplicate code. Consider the example, "If data type is NUMBER, right align". Without data binding, you would have to write code such as:

```
public void setDataType(String type)
{
  if (“NUMBER”.equals(type))
  {
    setAttributeValue(HALIGN_ATTR, HALIGN_RIGHT);
    setAttributeValue(DATATYPE_ATTR, type);
  }

}
```

```
public void setHAlign(String hAlign)
{
  if ("NUMBER"Equals(getAttributeValue(DATATYPE_ATTR)))
  {
    setAttributeValue(HALIGN_ATTR, HALIGN_RIGHT);
  }
  else
  {
    setAttributeValue(HALIGN_ATTR, hAlign);
  }
}

public String getDataType()
{
  return getAttributeValue(DATATYPE_ATTR, type);
}

public String getHAlign()
{
  if ("NUMBER"Equals(getAttributeValue(DATATYPE_ATTR)))
  {
    return HALIGN_RIGHT;
    return getAttributeValue(HALIGN_ATTR, hAlign);
  }
}
```

By implementing bound values, you can reduce this code to:

```
bean.setAttributeValue(HALIGN_ATTR, new HAlignBoundValue(bean));
```

with HAlignBoundValue defined as:

```
public class HAlignBoundValue implements BoundValue
{
  private OAWebBean mOAWebBean;
  HAlignBoundValue(OAWebBean bean)
  {
    mOAWebBean = bean;
  }
  public Object getValue(RenderingContext context)
  {
    if ("NUMBER"Equals(bean.getDataType()))
    {
      return HAlignRight;
    }
  }
}
```

Note that bound values are especially relevant when the value of the "depended on" attribute can change at any point up until rendering time. See the Runtime Control section for another example.

Another occasion when bound values are desirable is in the case of tables, where you have a handful of columns, but an unknown number of rows. Rather than consider a bean for each row and column, which does not scale, or reuse the bean hierarchies, you can bind the data in each column to a data source, such as a view object.

Data Binding in OA Framework

To understand how bound values work, let us take a look at the structure of a typical web bean. It can consist of indexed children, named children, and attributes. Attributes derive their values from a data dictionary of name/value pairs. Some name/value pairs are static values, such **prompt** /**"Employee"** or **style**/**"OraTipText"**, and are specified at design time in the OA Extension Property Inspector. Others are implemented as data bound values, such as the text attribute (**`TEXT_ATTR`**) on message beans, where the value is retrieved from the data source at rendering time.

For attributes such as `TEXT_ATTR`, UIX calls the `OABoundValue` class, an OA implementation of the UIX `BoundValue` interface. The class contains a single method:

```
public Object getValue(RenderingContext context);
```

Before examining how a value is fetched from a `BoundValue` object, note that OA Framework creates a named `DataObject` for each data source and then caches the `DataObject` on `oracle.cabo.ui.RenderingContext`. A data source is identified by the **viewName** set on the web bean. OA Framework then gets the value from the DataObject using the **viewAttr** attribute on the bean as a lookup key. Oracle Application implementations of the UIX DataObject interface are: `OADictionaryDataViewObject`,

OADictionaryDataRow, and OADictionaryDataString.

At rendering time, when UIX uses the BoundValue object to retrieve the value for TEXT_ATTR, the getValue method in the BoundValue object constructs a key using **viewName** and **viewAttr** and gets the named DataObject that is cached on RenderingContext. The logic in the BoundValue object then invokes selectValue, the only method in the UIX DataObject interface, which in turn calls the getAttribute method on the underlying view object to retrieve the value. That value is put on the BoundValue.getValue method, and is the value returned for the attribute at rendering time.

Attention To guarantee thread-safety in a multi-threaded enviroment, OA Framework uses data bound values (OABoundValue) for web bean attribute values. Classes that implement OAWebBean use the data bound values wherever necessary. If these web bean classes provide multiple versions of a method such as setText(String text) and setText(OAPageContext pageContext, String text), then to use data bound values, be sure to select the one that has the OAPageContext parameter. The method without OAPageContext is usually inherited from the super class. For example, use OAMessageTextInputBean.setText(OAPageContext page context, String text) instead of OAMessageTextInputBean.setText(String text).

Avoid calls like setAttributeValue(TEXT_ATTR, "<hardcoded string>"), as the attribute value is not a data bound value in this case. If you intend to set your custom attribute value through the setAttributeValue method, you should set the attribute value to your own custom data bound value object.

Attention When you want to get an attribute value from a bean, use the getAttributeValue(RenderingContext, AttributeKey) signature whenever possible to ensure that the attribute value is resolved correctly. Simply calling the attribute accessor or the getAttributeValue(AttributeKey) method without the RenderingContext parameter, may return incorrect results if the attribute was set as a bound value. For example, use:

getAttributeValue(pageContext.getRenderingContext(), RENDERED_ATTR);

The public Oracle E-Business Suite bound values that you can use are as follows and are discussed in more detail in the Runtime Control section:

OADataBoundValueAppModule
OADataBoundValueViewObject
OAFunctionSecurityBoundValue

Data Binding for Fields Outside a Table

When OA Framework implements bound values for fields outside of a table, the named dictionary of DataObjects is maintained with the current context. The data object name for each bean is defined by:

- Application module name and view object instance name - for view object data
- NON_VIEW_OBJECT_DATA (static variable in oracle.apps.fnd.framework.webui.OAWebBeanConstants interface) - for non-view object data

The data attribute name for each bean is defined by:

- View attribute name - for view object data
- Region code, item name, etc. - for non-view object data

Data Binding for Fields in a Table

When you implement bound values for fields in a table, OA Framework generates a DataObjectList instead of a single DataObject. A DataObjectList is simply a list of DataObjects. UIX iterates through the list, setting the "current DataObject" for each iteration. As a result, the bound value on the text attribute (TEXT_ATTR) uses the "current DataObject" to fetch its value, as described above.

Declarative Implementation

Certain attributes (the Required, Rendered, Disabled, and Read Only properties in the Property Inspector) can be bound to values declaratively using SPEL binding.

Runtime Control

OA Framework provides three implementations of bound values that you can use:
oracle.apps.fnd.framework.webui.OADataBoundValueViewObject,
oracle.apps.fnd.framework.webui.OADataBoundValueAppModule and
oracle.apps.fnd.framework.webui.OAFunctionSecurityBoundValue.

OADataBoundValueViewObject

You can use the OADataBoundValueViewObject class to bind a web bean's property with a view object attribute. To do this, set the attribute, representing the web bean's property, to:

```
    new OADataBoundValueViewObject(webBean, viewAttrName)
```

At rendering time, the attribute value becomes the value of the `viewAttrName` of the view instance associated with the web bean (which defaults to the OA Extension View Instance property). If you wish to override this default view instance, you can use the alternate constructor that allows specification of the view usage name:

```
    new OADataBoundValueViewObject(webBean, viewAttrName, viewUsageName)
```

You can also indicate whether or not to format the output of a bound value.

Specify `OADataBoundValueViewObject` as follows, to bind attributes that need to be formatted and rendered on the page (such as `TITLE_ATTR`, `TEXT_ATTR`, `PROMPT_ATTR`, ...). The output that results is always a string, formatted according to the data type of the web bean or view attribute:

```
OADataBoundValueViewObject(OAWebBean webBean,
                                                    String lookupName,
                                                    String viewUsageName)

OADataBoundValueViewObject(OAWebBean webBean,
                                                    String lookupName)
```

Specify `OADataBoundValueViewObject` as follows, to bind attributes that do not need to be formatted and rendered on the page (such as `RENDERED_ATTR`, `READONLY_ATTR`, `DISABLED_ATTR`, ...). Pass `formatToString` as '`false`'.

```
OADataBoundValueViewObject(OAWebBean webBean,
                                                    String lookupName,
                                                    String viewUsageName,
                                                    boolean formatToString)


OADataBoundValueViewObject(OAWebBean webBean,
                                                    String lookupName,
                                                    boolean formatToString)
```

Example

The following code shows how to bind the currency code attribute of the Total bean to display a formatted total according to the currency code attribute value of each row in a table:

```
// Now format the order total value based on the PO's currency code.
// "CurrencyCode" is the name of the attribute in the POSimpleSummaryVO
// that this table is referencing.

OAMessageStyledTextBean orderTotal =
  (OAMessageStyledTextBean)webBean.findIndexedChildRecursive("OrderTotal");
if (orderTotal !== null)
{
  orderTotal.setAttributeValue(CURRENCY_CODE,
    new OADataBoundValueViewObject(orderTotal, "CurrencyCode"));
}
```

OADataBoundValueAppModule

You can use the `OADataBoundValueAppModule` class to bind a web bean's property with a value returned from an application module. To do this, override `initializeWebValues(Hashtable paramValues)` in the application module to put values into the `Hashtable` keyed by a `lookupName`. For example, to have a specific web bean's prompt derived from one of the values returned by the application module, put `(lookupName, "Value of the prompt")`. Then set the attribute, representing the property of the web bean to:

```
    new OADataBoundValueAppModule(webBean, lookupName)
```

At rendering time, the attribute value is fetched from the web values `Hashtable` based on `lookupName`.

OAFunctionSecurityBoundValue

`OAFunctionSecurityBoundValue` is a bound value that returns Boolean.TRUE or Boolean.FALSE based on whether the current session user is authorized to access a specific function. Use this class to check whether a certain function is granted based on specific data context. For example, you can bind the RENDERED_ATTR attribute to a function so that the bean is hidden if the function is not granted

```
//Hides the customer name bean if function 'ShowCustomerNameFunction' is not granted.
...
```

```
OAFunctionSecurityBoundValue fSBoundValue =
  OAFunctionSecurityBoundValue("ShowCustomerNameFunction");
custNameBean.setAttribute(pageContext.getRenderingContext(),
  RENDERED_ATTR, fSBoundValue);
...
```

Other UIX Bound Values

UIX provides many other useful bound values. The following table lists the bound values that support some basic operations.

Operations	UIX Bound Values
Arithmetic	`AddBoundValue`, `ConcatBoundValue`
Comparison	`ComparisonBoundValue`, `AndBoundValue`, `OrBoundValue`, `NotBoundValue`
Type Conversion	`ToBooleanBoundValue`, `ToDateBoundValue`, `ToCharacterBoundValue`, `ToIntegerBoundValue`, `ToStringBoundValue`
Matching Attribute Values Among Beans	`NodeAttributeBoundValue`

For a complete list of UIX bound values, refer to the `oracle.cabo.ui.data.bind` package.

Example 1

The following code shows how to bind a property based on the value of another property. In this case the total bean is not rendered if the salary bean is not rendered:

```
OAMessageStyledTextBean totalBean =
  (OAMessageStyledTextBean)webBean.findIndexedChildRecursive("totalBean");
OAMessageStyledTextBean salaryBean =
  (OAMessageStyledTextBean)webBean.findIndexedChildRecursive("salaryBean");
if(totalBean!=null && salaryBean!=null)
{
  totalBean.setAttribute(RENDERED_ATTR,
    new NodeAttributeBoundValue(salaryBean, RENDERED_ATTR));
}
```

Example 2

The following code shows how to use UIX standard bound values to achieve more complex or compound binding. In this example, the name attribute is concatenated with the description attribute at rendering time:

```
OAHeaderBean headerBean =
  (OAHeaderBean)webBean.findIndexedChildRecursive("headerBean");
if (nameBean!==null && descBean!=null && headerBean!=null)
{
  headerBean.setAttributeValue(TEXT_ATTR, new ConcatBoundValue(new BoundValue[]
    {new OADataBoundValueViewObject(concatBean,"Name", viewUsage),
    new OADataBoundValueViewObject(concatBean, "Description", viewUsage)});
}
```

Form Submit Bound Values

See Submitting the Form for additional information about bound values that can be used in specific cases where you need to force a form submit when a component is activated.

Personalization Considerations

See a summary of Bound Values personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guideline(s)

    - None
- Javadoc
    - `oracle.cabo.ui.data.BoundValue`
    - `oracle.cabo.ui.data.DataObject`
    - `oracle.cabo.ui.data.DataObjectList`
    - `oracle.cabo.ui.RenderingContext`
    - `oracle.apps.fnd.framework.webui.OAWebBeanConstants`
    - `oracle.apps.fnd.framework.webui.OADataBoundValueViewObject`
    - `oracle.apps.fnd.framework.webui.OADataBoundValueAppModule`
    - `oracle.apps.fnd.framework.webui.OAFunctionSecurityBoundValue`
- OA ToolBox Tutorial / Sample Library
    - Delete Lab.

Branding

Overview

As described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Branding, every OA Framework page reserves the upper left-hand corner for either:

- Basic (Non-Contextual) Branding - includes corporate ("Oracle") and product brand names
- In-Context Branding - includes user-selected contextual information in addition to the corporate and product brand names

All OA Framework pages must provide basic branding support as described below. In-Context branding may be used in select cases (see Oracle Branding Options in the guideline for specific use case recommendations).

Basic (Non-Contextual) Branding

Basic branding supports a regular layout style as shown in Figure 1, where "E-Business Suite" is the product branding text that appears after the "Oracle" corporate brand name.

Figure 1: Basic Branding (corresponds to BLAF "Regular" Layout)



Regular Layout

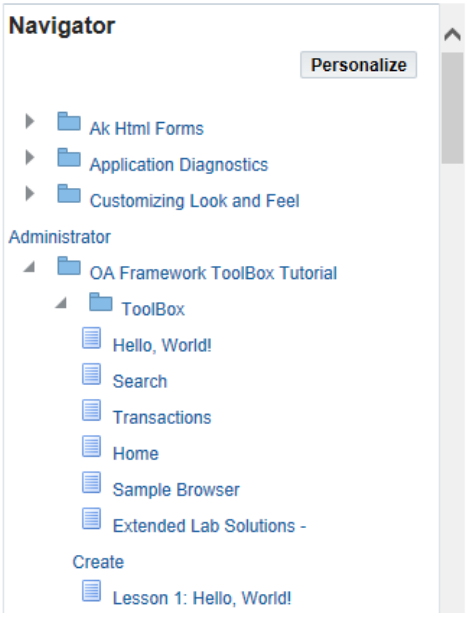
Pages Launched from the Oracle E-Business Suite Home Page

If your OA Framework page is launched from the Oracle E-Business Suite Home page, the OA Framework automatically sets the branding text for you based on the current selected responsibility and page link. This ensures consistency for the user between the options presented in the responsibility main menu, and the branding text displayed on the application page.

For example, as shown in the Navigator in Figure 2, the user has selected the responsibility **Workflow Administrator Web Applications**. The top-level menu associated with this responsibility has several submenus with prompts: **Administrator Workflow**, **Transaction Monitor** and so on. If the user selects a link beneath one of these submenus (the corresponding function must be of type JSP or INTEROPJSP), the OA Framework automatically sets the branding text to the parent submenu's prompt.

- If the user selects the **Business Events** link in the example below, the OA Framework sets the branding text to **Administrator Workflow**.
- If the user selects the **Transaction Monitor** link in the example below, the OA Framework sets the branding text to **Transaction Monitor** (if the **Transaction Monitor** link label were to be changed to "My Monitor," the branding text would still be set to **Transaction Monitor** since the link prompt has no effect on this).

Figure 2: Oracle E-Business Suite Configurable Home page Navigator tree



If you attach a function directly to the responsibility main menu instead of a grouping submenu, the branding text defaults to the responsibility name. For example, assume you have the following structure based on the Workflow use case shown above. Note that the new "Worklist" function is attached directly to the main menu. When this page renders, the branding text will be **Workflow Administrator Web Applications**.

```
Workflow main menu // root menu associated with the Workflow Administrator Web Applications responsibility
| -- Worklist (function)
```

```
| -- Administrator Workflow (grouping menu)
     | -- Home (function)
     | -- Developer Studio (function)
     | -- Business Events (function)
     | -- Status Monitor (function)
     | -- Notifications (function)
| -- Transaction Monitor (grouping menu)
     | -- Transaction Monitor (function)
```

In this case, you can either add a grouping menu to your main menu and move the function accordingly, or you can use the manual override instructions provided below.

Branding Override

If the default behavior is unacceptable, you can override it by following these instructions:

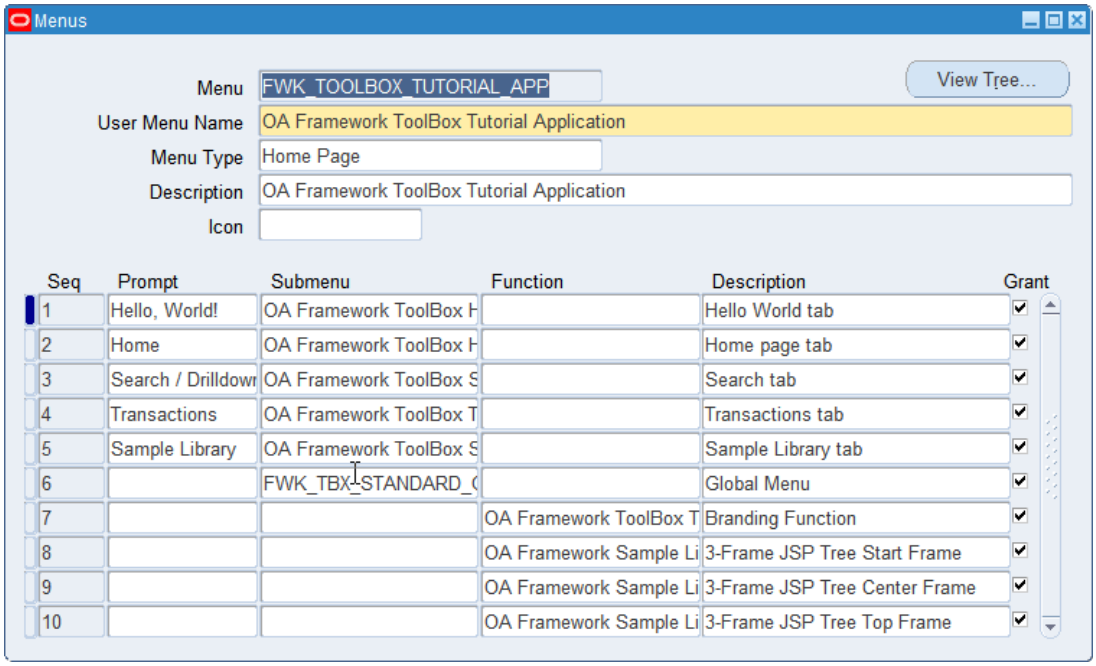
Step 1: Define a form function to represent your branding text (see Tabs / Navigation if you need information on creating application functions and menus). Note that the **User Function Name** value must be set to the branding text that you want to display in the upper left-hand corner of your application's pages. For example, in the OA Framework ToolBox Tutorial, we defined a function with the following properties. The value "OA Framework ToolBox Tutorial" displays in the branding area.

- Function: FWK_TOOLBOX_BRAND
- User Function Name: OA Framework ToolBox Tutorial

Step 2: Associate this function with your application's "Home Page" navigation menu.

Note: It's important that you do not specify a prompt when you associate your branding function with the menu. Figure 3 shows the FWK_TOOLBOX_BRAND function properly associated with the ToolBox Tutorial's "Home Page" navigation menu.

Figure 3: OA Framework ToolBox Tutorial "Home Page" menu with branding function



Step 3: Set the request parameter **OAPB** to the name of the function that you created in Step 1. You should specify this wherever you set the menu context **OAHP** request parameter (see Tabs / Navigation if you need information about this).

For example, if you access your application from a test JSP, the page link should include the OAPB parameter in addition to the OAHP and the OASF menu parameters. This is illustrated in the URL below from the OA Framework ToolBox Tutorial **test_fwktutorial.jsp**:

```
<a href="<%=URLMgr.processOutgoingURL("OA.jsp?OAFunc=FWK_TOOLBOX_HELLO
&OAPB=FWK_TOOLBOX_BRAND&OAHP=FWK_TOOLBOX_TUTORIAL_APP
&OASF=FWK_TOOLBOX_HELLO&transactionid=" + transactionid +
"&dbc=" + dbcName, macKey) %>">Hello,World!</a><br>
```

Alternatively, you can specify the product branding on the page itself:

Step 1: For each page in your application, select the **pageLayout** region in the JDeveloper structure pane, right-click and select **New > productBranding**. Set the corresponding item's `Style` to **formattedText**. Assign the **formattedText** item a standards-compliant `ID`, and set the `Text` property to product name that you want to display in the branding area.

Note: The branding context that you set declaratively on a page is not retained across the application. If you want to retain the branding context across an application, you should set the OAPB parameter, as it was introduced specifically for that purpose.

The following matrix describes the precedence order for the different sources of the branding text value. The OAPB parameter, if specified, always prevails. The declarative page setting also overrides the automatic setting of the grouping menu value. The responsibility name displays only if no other options are available.

OAPB Parameter	Declarative Product Branding	Grouping Submenu	Result: Branding Source
Not specified	Not specified	Specified	Grouping Submenu
Specified	Not specified	Specified	OAPB
Not specified	Specified	Specified	Declarative Product Branding
Specified	Specified	Specified	OAPB
Not specified	Not specified	None	Responsibility

Once the branding context is set with the OAPB parameter value, it remains unchanged until you explicitly reset the **OAPB** parameter value. Options for resetting this value include:

- Defining in in a URL associated with a item
- Defining it in a navigation function's Web HTML call
- Passing it as a parameter in a JSP forward or client redirect method call

Tip: If you support switching application contexts at runtime, remember to change your brand!

Pages Launched from Forms

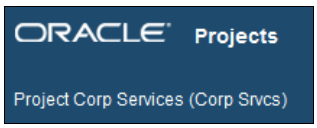
For OA Framework pages launched from a form, the branding text should render as the display name of the *current form function*. To do this, call `fnd_function.user_function_name` in your form and set the OAPB branding parameter to this value when you open the OAF page. For additional information about opening OA Framework pages from Forms, see the Forms / OA Framework Integration document.

For OA Framework pages launched from a link in the Forms Navigator, the branding text should render as the display name of the associated form function. So, when you define this form function's Web HTML Call, set the OAPB parameter value to the current function name.

In-Context Branding

The in-context branding includes the corporate and product brand images. Additionally, contextual information renders below the corporate and product information as shown in Figure 4.

Figure 4: Example of in-context branding



Note that this style of branding is intended to be used only for cases where the user makes a contextual selection when starting work that remains unchanged for the life of the application or task.

Declarative Implementation

Step 1: For each page in your application, select the **pageLayout** region in the JDeveloper structure pane, right-click and select **New > productBranding**. Set the corresponding item's `Style` to **formattedText** instead of "image" as you did above. Assign the **formattedText** item a standards-compliant `ID`, and set the `Text` property to product name that you want to display in the branding area.

Step 2: Select the **pageLayout** region again, right-click and select **New > inContextBranding**. Set the corresponding **formattedText** item's `ID` and set the `Text` property to the context you wish to display (for example, this would be "Project Corp Services (Corp Srvcs)" in Figure 4 above).

Note: Creating an **inContextBranding** component without a **productBranding** component is invalid. The OA Framework throws a Developer Test Mode error for this condition if you work with this test mode enabled.

Runtime Control

If you need to change the contextual information programmatically, you can do the following:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{

 super.processRequest(pageContext, webBean);

 OAFormattedTextBean inContextTextBean =
   (OAFormattedTextBean)createWebBean(pageContext, FORMATTED_TEXT_BEAN);

 // Remember that you must pass a translated value obtained from message dictionary,
 // not a static String as shown here.

 inContextTextBean.setText("Text for In-Context Branding");
```

```
 // Ensures the correct CSS style is applied.
 inContextTextBean.setStyleUsage(IN_CONTEXT_BRANDING_STYLE);

 OAPageLayoutBean page = pageContext.getPageLayoutBean();
 page.setInContextBranding(inContextTextBean);

}
```

Personalization Considerations

- See a summary of Branding personalization considerations in the Oracle Application Framework Personalization Guide.
- You can also refer to the section called Personalizing Your System: Branding in the Oracle Application Framework Personalization Guide for additional information.

Known Issues

- None

Related Information

- BLAF UI Guidelines
  - Branding
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`

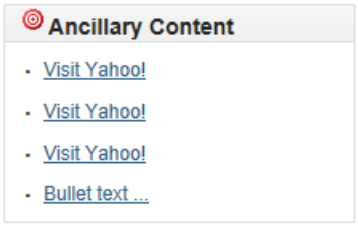
.

Bulleted List

Overview

Simple, HTML bulleted lists appear in numerous BLAF UI Guideline specifications. For example, they are routinely used in content containers as shown below:

Figure 1: Example of a content container with a bulleted list.



As implemented in the OA Framework, the bulleted list is a container that can hold any kind of children (although, for all practical purposes, most bulleted lists are simply comprised of plain text or links). Each region item is rendered with a bullet.

The bulleted list can be split into columns by specifying the maximum number of rows (bullet items) that should render in a column before starting another until the 3 column maximum is reached. Once the total number of rows would exceed 3 columns using the specified multiplier, all rows are allocated as evenly as possible to 3 columns.

Declarative Implementation

To add a bulleted list to your page, follow these steps. The OA Framework will create an `oracle.apps.fnd.framework.webui.beans.layout.OABulletedListBean`.

Step 1: Create a region item set its style to **bulletedList**.

Step 2: Set the region's `ID` property in accordance the OA Framework FIle Naming Standards.

Step 3: Add one or more items to the **bulletedList**. They can be of any item style.

Step 4: (optional) Set the `Height` property to determine how many items should render before a new column is created.

Step 5: Save your work.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

Instantiate

See the `oracle.apps.fnd.framework.webui.OAControllerImpl` Javadoc for other `createWebBean*()` signatures.

Note: Always choose a signature that lets you specify the web bean's internal name.

```
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.layout.OABulletedListBean;
...
processRequest(OAPageContext pageContext, OAWebBean webBean)
{
...


 // Create the list bean container
 OABulletedListBean bullets =
   (OABulletedListBean)createWebBean(pageContext,
                                     OAWebBeanConstants.BULLETED_LIST_BEAN,
                                     null, "bulletList");

 // Create and add a link, plain text and an image to the bulleted list
```

```
 OALinkBean link = (OALinkBean)createWebBean(pageContext,
                                             OAWebBeanConstants.LINK_BEAN, null,
                                             "linkEx");
 OAStaticStyledTextBean text =
   (OAStaticStyledTextBean)createWebBean(pageContext,
                                         OAWebBeanConstants.STATIC_STYLED_TEXT_BEAN,
                                         null, "textEx");

 OAImageBean image = (OAImageBean_createWebBean(pageContext,
                                                OAWebBeanConstants.IMAGE_BEAN,
                                                null, "imageEx");

 bullets.addIndexedChild(link);
 bullets.addIndexedChild(text);
 bullets.addIndexedChild(image);
...
```

Control Visual Properties

To set the multiple after which the rows are split into columns, get a handle to the bulleted list bean and call `setRows(int)`. The default split size is 10. You can also pass the value `Integer.MAX_VALUE` as the parameter if you want all your bullet items to render in a single column, regardless of their number.

Personalization Considerations

- None

Known Issues

- None

Related Information

- BLAF UI Guideline
  - Content Containers in a Page
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OABulletedListBean`
- ToolBox Tutorial / Sample Library
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.SampleBrowserPG.xml`

Buttons (Action/Navigation)

Overview

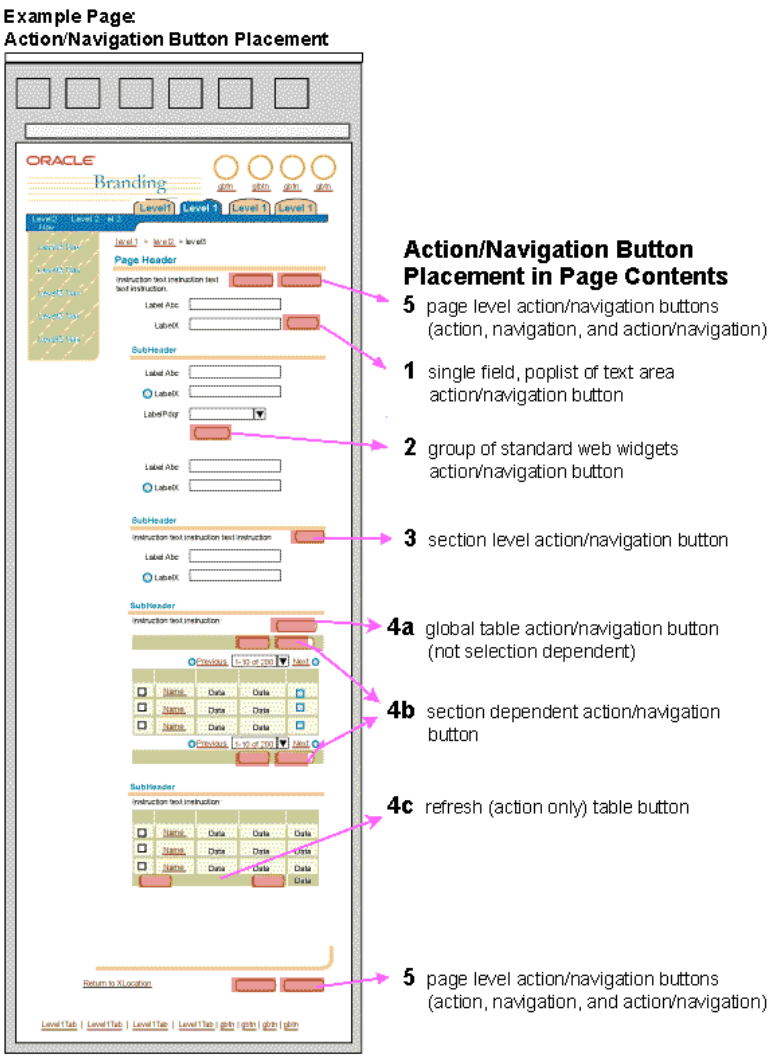
As described in the BLAF UI Guideline: Buttons (Action/Navigation) specification, action/navigation buttons can be used to:

- Perform actions without navigating the user off the page (the page redraws with evidence that an action has been performed)
- Navigate the user to another page *without* performing any actions
- Perform an action and navigate the user to another page

Action/navigation buttons can be placed as follows within a page.

- In relation to a single field, poplist, or text area
- In relation to a group of standard widgets
- In relation to a table
- In relation to the entire page (as individual action buttons, or within a multistep navigation control)

Figure 1: BLAF UI Guidline illustration of all possible action/navigation button usages within a page



This document describes how to implement each of the following:

- Action (Submit) Buttons
- Navigation (Link) Buttons

It also describes how to position buttons in different page locations (for example, page-level buttons).

For information on implementing buttons for navigating multistep transaction flows, see the Locator Element: Page/Record Navigation document.

Action (Submit) Buttons

Action buttons submit the page form when selected by the user (they perform an `HTTP POST`).

Declarative Implementation

To add a submit button to your page (regardless of its location) follow these steps. The OA Framework will instantiate an `oracle.apps.fnd.framework.webui.beans.form.OASubmitButtonBean` with the name you assign the region item in Step 1.

Step 1: Create a region item and set its style to **submitButton**

Step 2: Set the region item's `ID` property in accordance the OA Framework File naming standards.

Step 3: (optional): If you are adding a common button such as the "Apply," "Go," or "Cancel" buttons, or a standard button for your product, specify an attribute set. For example, the standard OA Framework "Go" button attribute set is:

/oracle/apps/fnd/attributesets/Buttons/Go

See Implementing the View for additional information about using attribute sets.

Step 4: Specify the button label by setting the Prompt property, and making it unique. There should be no buttons with duplicate labels, unless they have the same functionality. For example, if there are 'Go' buttons at page level, table level and search level, they should all be labeled differently since they will have different functionality. The same principle also applies to the Prompt property provided via attribute set.

Step 5: If the button label is meaningful on its own, do not specify the Additional Text property. Only Specify Additional Text if additional information is required. This value is displayed as tooltip text when the mouse hovers over the button. Remember that the additional text is only available to mouse users and not to mobile or tablet users, so consider changing the button label to something more meaningful and leaving the Additional Text property null.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically-created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

Instantiate

To instantiate an `OASubmitButtonBean`, call the appropriate `createWebBean()` factory method in the `oracle.apps.fnd.framework.webui.OAControllerImpl` class. If you select a signature that requires a constant to determine what kind of bean to create, use `oracle.apps.fnd.framework.webui.OAWebBeanConstants.BUTTON_SUBMIT_BEAN`.

Control Visual Properties

The only visual property of a button that you might change at runtime is its text. To do this, get a handle on the `OASubmitButtonBean` and call its `setText(pageContext, String)` method.

Remember when setting String values displayed in the user interface to always obtain the value from Applications Message Dictionary first. Never set a hard-coded value.

Control Behavior and Data

In rare cases, the BLAF UI Guidelines allow for buttons to be disabled. To disable a button, get a handle on `OASubmitButtonBean` and call its `setDisabled(boolean)` method.

You might also want to turn off Javascript `onSubmit` validation when the button is pressed (for example, you have a submit button that might be pressed before a page's data is fully entered by the user, and you don't want to annoy him with premature validation errors). In your controller's `processRequest()` method, get a handle to the `OASubmitButtonBean` and call its `setUnvalidated(Boolean)` method.

Finally, you can also turn off server-side validation when a submit button is pressed. In this case, all page data will be pushed to the underlying view object(s) (and corresponding entity objects) where server-side validation *will* be performed, however, any exceptions will be ignored so code that you write in `processFormRequest()` will proceed as if there had been no validation performed in the `processFormData()` phase. To implement this in your controller's `processRequest()` method, get a handle to the `OASubmitButtonBean` and call its `setServerUnvalidated(Boolean)` method.

For additional information about the submit processing phases, see Implementing the View. For information about bypassing validation, see Implementing the Controller.

Tip: If you define a button and you don't want it to perform either client or server-side validation (a transaction "Cancel" button is a common example if you don't need to implement special processing when it's pressed), consider making it a plain navigation button instead.

Handle Button Press Events

When the user selects a submit button, the browser performs an `HTTP POST` while adding the button's name to the request (note that this name is set to the button's `ID` property, or the name value specified when creating the web bean programmatically). To ascertain whether a particular submit button has been pressed , add the following code to a controller associated with a region above the button in the page's bean hierarchy.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
 ...

 // Check to see if a submit button named "Go" has been pressed.
 if (pageContext.getParameter("Go") != null)
 {

 ...
```

If the action button that you're implementing should also navigate the user to another page after performing an action (or you need to forward back to the current page so you can edit the web bean hierarchy in `processRequest()`), use the `setForward*()` methods in the `oracle.apps.fnd.framework.webui.OAPageContext`.

Blocking on Submit

Whenever a submit action takes place on a page, subsequent submits can be blocked. When using a blocking on submit technique, when the submit action takes place, the cursor becomes busy and prevents any other submit action until the current submit event has been handled. The block on submit behavior is not enabled by default on a page. However, For Partial Page Refresh (PPR) events alone, it is enabled by default.

To implement the block on submit behavior on a specfic page, add the following code to the processRequest() of that page:

```
import oracle.apps.fnd.framework.webui.beans.OABodyBean;
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
     {
super.processRequest(pageContext, webBean);
OAWebBean body = pageContext.getRootWebBean();
if (body instanceof OABodyBean)
     {
((OABodyBean)body).setBlockOnEverySubmit(true);
     }
...
...
     }
.
```

Navigation (Link) Buttons

Navigation buttons navigate the user to a new destination without performing any actions. In other words, they are simply fancy links that perform an `HTTP GET` when selected by the user.

Note: It is perfectly appropriate for a navigation button to perform an `HTTP POST` instead of a `GET` if necessary for your page design. In this case, simply define the button with a **submitButton** item style. When selected, the button will submit the page form as described above.

Declarative Implementation

To add a plain, link-like button to your page (regardless of its location) follow these steps. The OA Framework will instantiate an `oracle.apps.fnd.framework.webui.beans.nav.OAButtonBean` with the name you assign the region item in Step 1.

Step 1: Create a region item and set its style to **button**.

Step 2: Set the region item's `ID` property in accordance the OA Framework File Standards.

Step 3: (optional): If you're adding a standard button for your product, specify an attribute set.

See Implementing the View for additional information about using attribute sets.

Step 4: (assuming no attribute set): Specify the button label by setting the `Prompt` property

Step 5: (assuming no attribute set): Specify the ALT text by setting the `Short Description` property. This value is required for assistive technologies, and is also displayed as tooltip text when a mouse moves over the button.

Step 6: Specify the `Destination URI` property as shown in the following examples:

OA.jsp?OAFunc=FWK_TBX_EMPLOYEE&retainAM=Y
OA.jsp?page=/oracle/apps/fnd/framework/toolbox/tutorial/webui/PoDetailsPG

See Implementing the View for additional information about specifying URL parameters.

Step 7: Beginning in Release 12.2.5, the button item supports showing an additional icon along with the button title. Use the `Icon URI` property to specify the URI for the icon file. Use the `Icon Position` property to specify the position of the icon relative to the button text. Valid choices include **start**, **end**, **top** or **bottom**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically-created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

Instantiate

To instantiate an `OAButtonBean`, call the appropriate createWebBean factory method in the `OAControllerImpl` class. If you select a signature that requires a constant to determine what kind of bean to create, use `OAWebBeanConstants.BUTTON_BEAN`.

Control Visual Properties

The visual property of a button that you're likely to change is whether it's displayed or not. To do this, get a handle to the `OAButtonBean` and call its `setRendered(boolean)` method. You can also change its prompt by calling its `setText(pageContext, String)` method, and its ALT text by calling `setShortDescription(String)`.

Remember when setting String values displayed in the user interface to always obtain the value from Applications Message Dictionary first. Never set a hard-

coded value.

As of Release 12.2.5, you can display an icon along with the button title. You can programmatically add an icon using code similiar to the example below:

```
button =
   (ButtonBean)webBean.findIndexedChildRecursive("buttonrighticon");
   button.setIcon("cabo/images/error.png");
   button.setIconPosition(button.BUTTONICON_POSITION_START);
```

Control Behavior and Data

In rare cases, the BLAF UI Guidelines allow for buttons to be disabled. To disable a button, get a handle the `OAButtonBean` and call its `setDisabled(Boolean)` method.

If you need to set the button's destination programmatically, always specify the URL in relation to the document root node. For example:

Handle Button Press Events

When the user selects a plain navigation button, the browser issues an `HTTP GET` request to display the new target page. There is no need to write any code to ascertain whether the button is pressed. If you must handle the button press before navigating to a new page, create an OASubmitButtonBean instead.

Location-Specific Button Implementations

This section describes how to add action/navigation buttons to specific locations within a page. In all cases, create either a submit button or a plain button as described above.

Page-Level Buttons (Page Button Bar)

As of Release 12.2.5, page-level action/navigation buttons render to the right of the page title, on the same line, as shown below.

Figure 2: Page-level action/navigation buttons on the Create Supplier page.



Page-level action/navigation buttons also render below the page content if the page-level buttons on the top right of the page are scrolled out of view.

Declarative Implementation

Note: You must specify a page title for your page if you want the page-level action/navigation buttons to appear at the top of the page. If you don't set this value, they will appear only beneath the "ski." See Headers and Subheaders for additional information about specifying a page title.

To add page-level action/navigation buttons:

Step 1: Select the **pageLayout** region and add a region beneath it with the style **pageButtonBar**. The OA Framework will an instantiate an `oracle.apps.fnd.framework.webui.beans.nav.OAPageButtonBarBean`.

Step 2: Add one or more buttons to the **pageButtonBar** region (follow the instructions above for adding specific button types). Add them in ascending sequence as you want them to appear from left to right. So, for example, if you have a "Cancel" and "Apply" button on your page, and you want them to render with "Apply" being the rightmost button as specified in the UI Guidelines, add the "Cancel" button first.

Note: The OA Framework automatically adds the correct amount of space between buttons when you add them to the **pageButtonBar**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

To instantiate the `OAPageButtonBarBean` for page-level action/navigation buttons, follow this example showing an `OASubmitButtonBean` "Apply" and an `OAButtonBean` "Cancel":

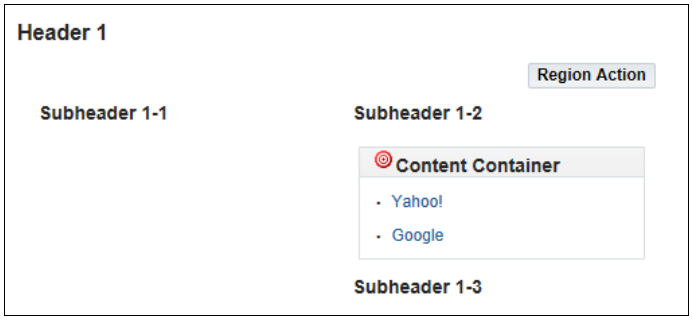
```
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;

import oracle.apps.fnd.framework.webui.form.OASubmitButtonBean;
import oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean;
import oracle.apps.fnd.framework.webui.beans.nav.OAButtonBean;
import oracle.apps.fnd.framework.webui.beans.nav.OAPageButtonBarBean;
...
processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   ...
```

```
// Assuming the controller is associated with the pageLayout region
OAPageLayoutBean page = (OAPageLayoutBean)webBean;

// Remember to use Message Dictionary for UI Strings; never hard-code Strings as shown here.

// Always use the createWebBean signatures that let you specify the component's internal name,
// unless you're creating a web bean defined in JDeveloper.
OAPageButtonBarBean buttons =
   (OAPageButtonBarBean)createWebBean(pageContext, OAWebBeanConstants.PAGE_BUTTON_BAR_BEAN, null, "pbBar");
OASubmitButtonBean applyButton =
    (OASubmitButtonBean)createWebBean(pageContext, OAWebBeanConstants.BUTTON_SUBMIT_BEAN, null, "applyButton");
applyButton.setText("Apply");
OAButtonBean cancelButton =
   (OAButtonBean)createWebBean(pageContext, OAWebBeanConstants.BUTTON_BEAN, null, "cancelButton");
cancelButton.setText("Cancel");

// Now add the buttons to the page button bar.  Remember to add them as you want them
// to display from left to right (in an American UI).
buttons.addIndexedChild(cancelButton);
buttons.addIndexedChild(applyButton);

// Finally, set the page button bar on the page layout. This example assumes the
// page title was set declaratively.
page.setPageButtons(buttons);

...
```

Region-Level Buttons

Buttons that relate to a region render right-justified immediately below the region header as shown below.

Figure 3: Example of a region-level button

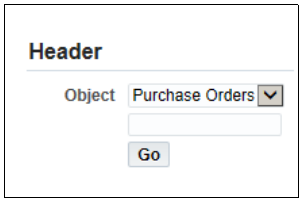


For a reference implementation, select the Headers and Subheaders option in the ToolBox Sample Library. Note that adding a **pageButtonBar** region to any region other than the **pageLayout** region results in its child(ren) button(s) rendering only once beneath the region's header.

Component Group Buttons

Buttons that relate to a group of components render immediately below the group.

Figure 4: Example of a button for a group of related widgets



For a reference implementation, select the Headers and Subheaders option in the ToolBox Sample Library.

Component Button

A button that relates to a single component (like a field or poplist) renders immediately to the right of the component as shown below. A "Go" button for a single search field is a common example.

Figure 5: example of a button in relation to a single widget

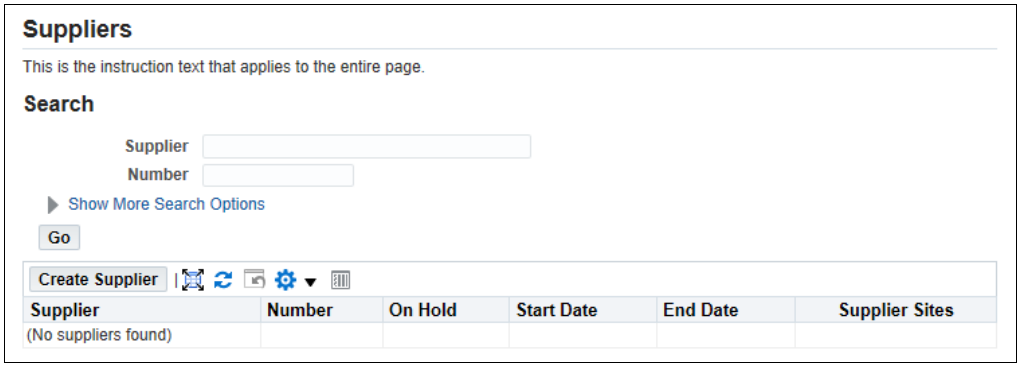


For a reference implementation, select the Workflow Worklist option in the ToolBox Sample Library.

Table Button

Buttons that relate to an entire table render flush left (or right aligned when translated to BiDi languages), immediately above the table as shown below. A button for creating objects displayed in the table is a common example.

Figure 6: Example of a button in relation to a table



For a reference implementation, select the Transactions option in the ToolBox Sample Library.

Buttons that perform actions on a selected table row are a special case. See the tables (classic or advanced) documentation for additional information about implementing control bar buttons and poplists.

Personalization Considerations

- See a summary of Button personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guideline(s)
  - Buttons (Action/Navigation)
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.form.OASubmitButtonBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OAButtonBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OAPageButtonBarBean`
- OA Framework ToolBox Tutorial / Sample Library

Buttons (Global)

Overview

As described in the BLAF UI Guideline: Buttons (Global) specification, global buttons provide access to tasks that apply to an entire application , and as such, are accessible from every page in the application. Global buttons render in the upper right-hand corner of a screen.

Accommodating Different Screen Sizes

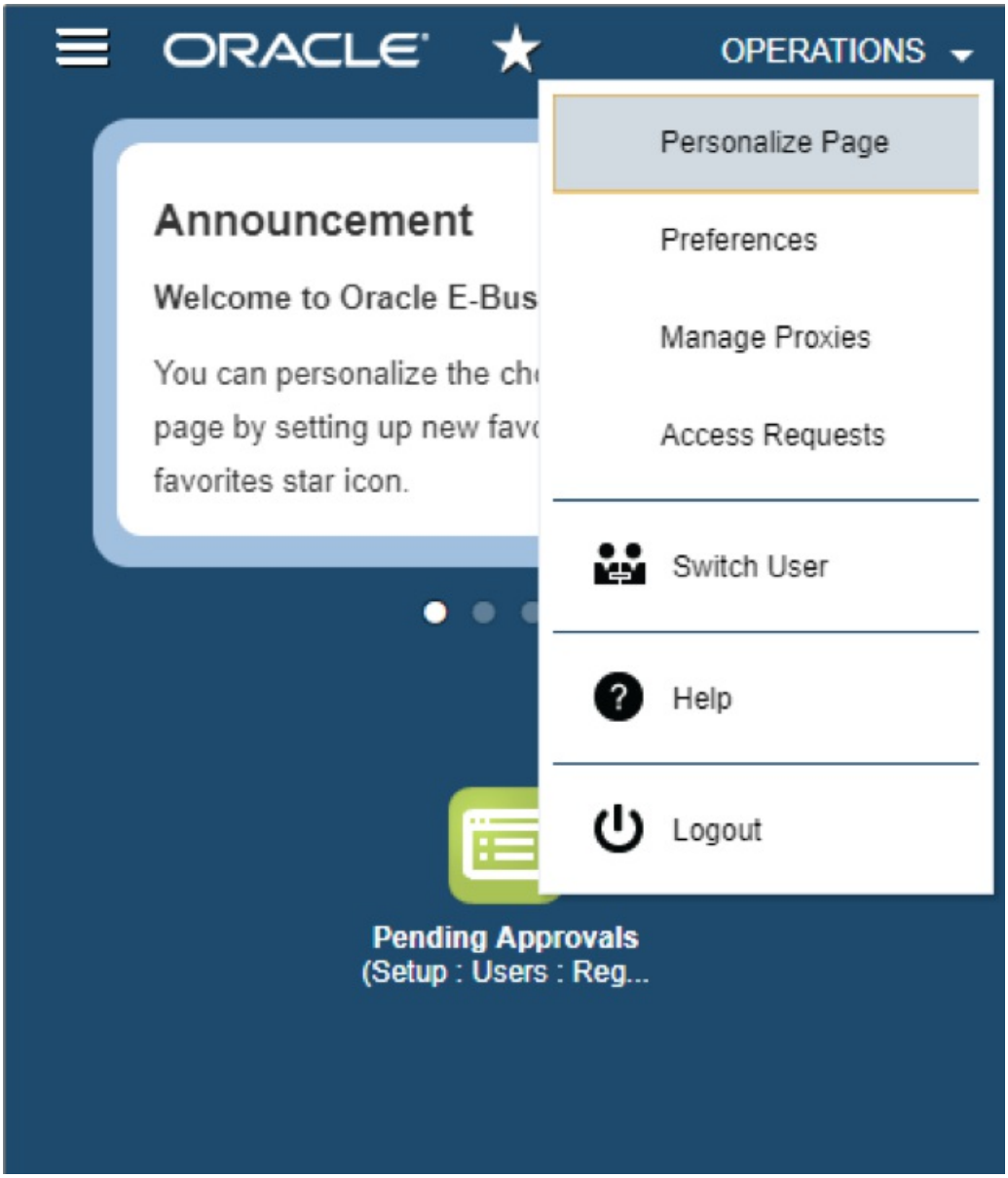
As of Release 12.2.9, the Global Header has been enhanced to accommodate users running Oracle E-Business Suite on devices with various sizes of screen, such as tablets and mobile phones, as well as the more traditional desktop and laptop computers.

Key design criteria here were:

- When there is not enough space to display the global header/footer completely, it is consolidated (has its elements rearranged) to make the best use of the available space.
- Various elements are also styled with flat corners as part of this strategy.

When an OA Framework page is accessed from a device that does not have enough space to display the complete global header, the header is displayed in a consolidated manner as shown in Figure 1 below.

Figure1: Consolidated display of header



The favorites and worklist are rendered as icons at the top, and the rest of the menu options are displayed as a consolidated list under the username popup menu.

In consolidated mode, to save space the favorites and worklist are rendered as icons only.

Home Page Menu Styling

To better utilize available space, the menus (Navigator, Favorites and Worklist) rendered in the Home Page are styled with flat corners and their sizes are adjusted.

Figure 2: Navigator menu rendered with flat corners

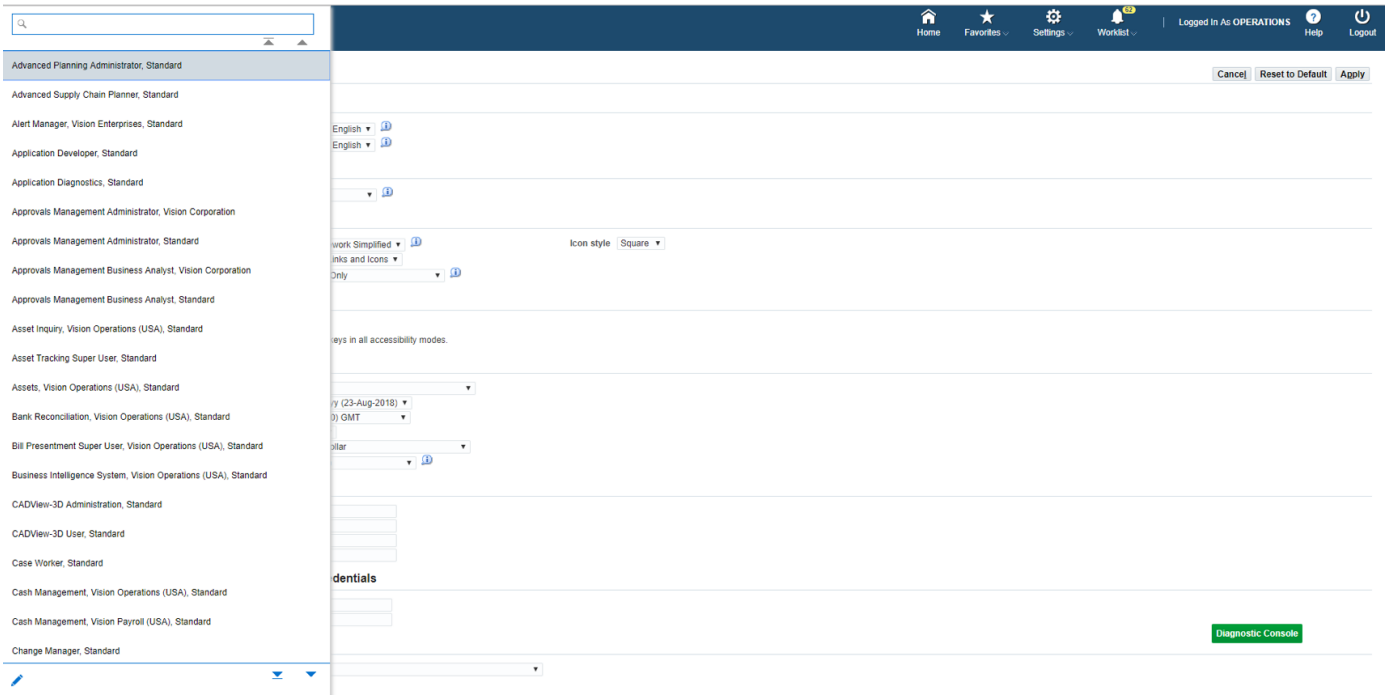


Figure 3: Favorites menu rendered with flat corners

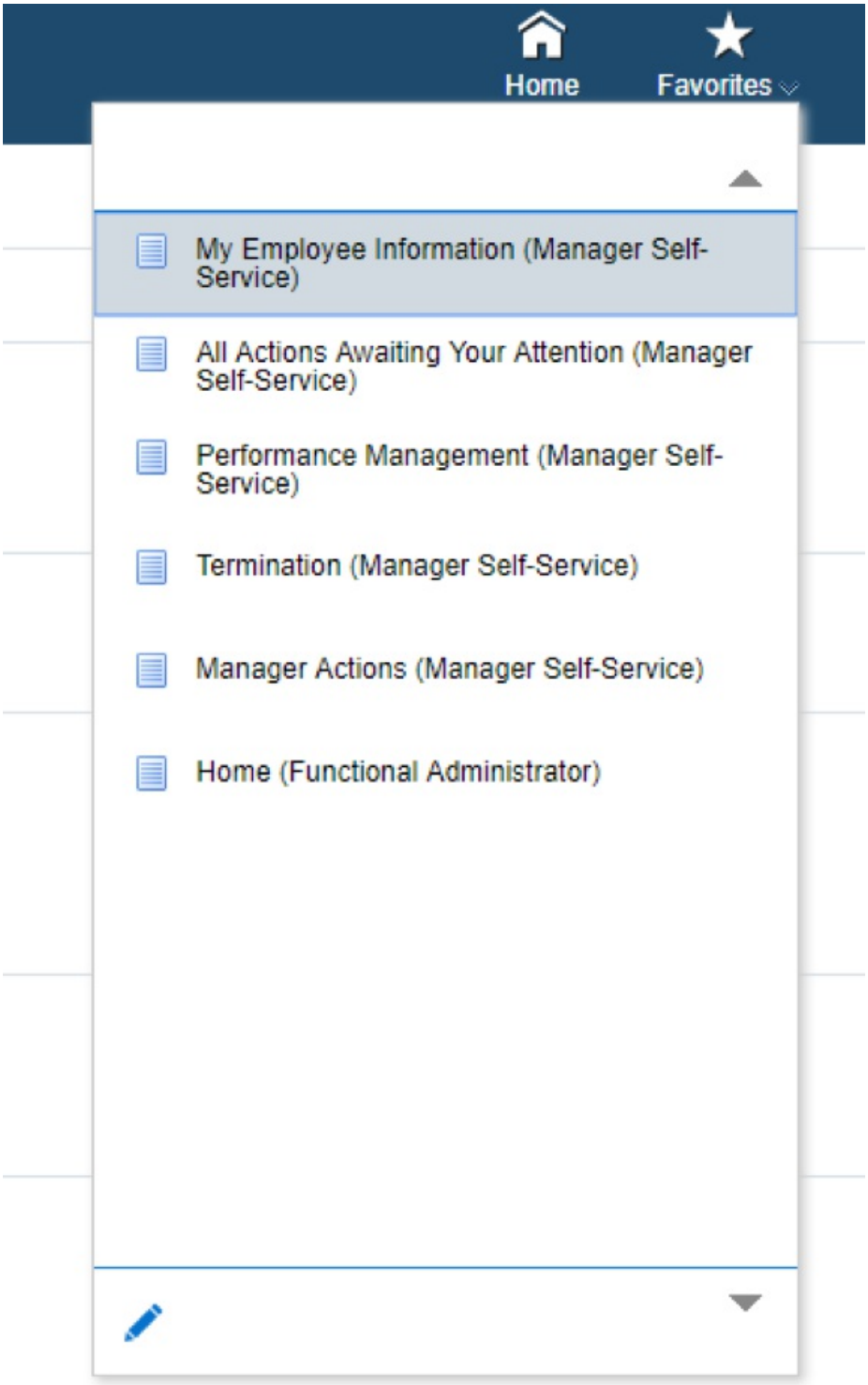
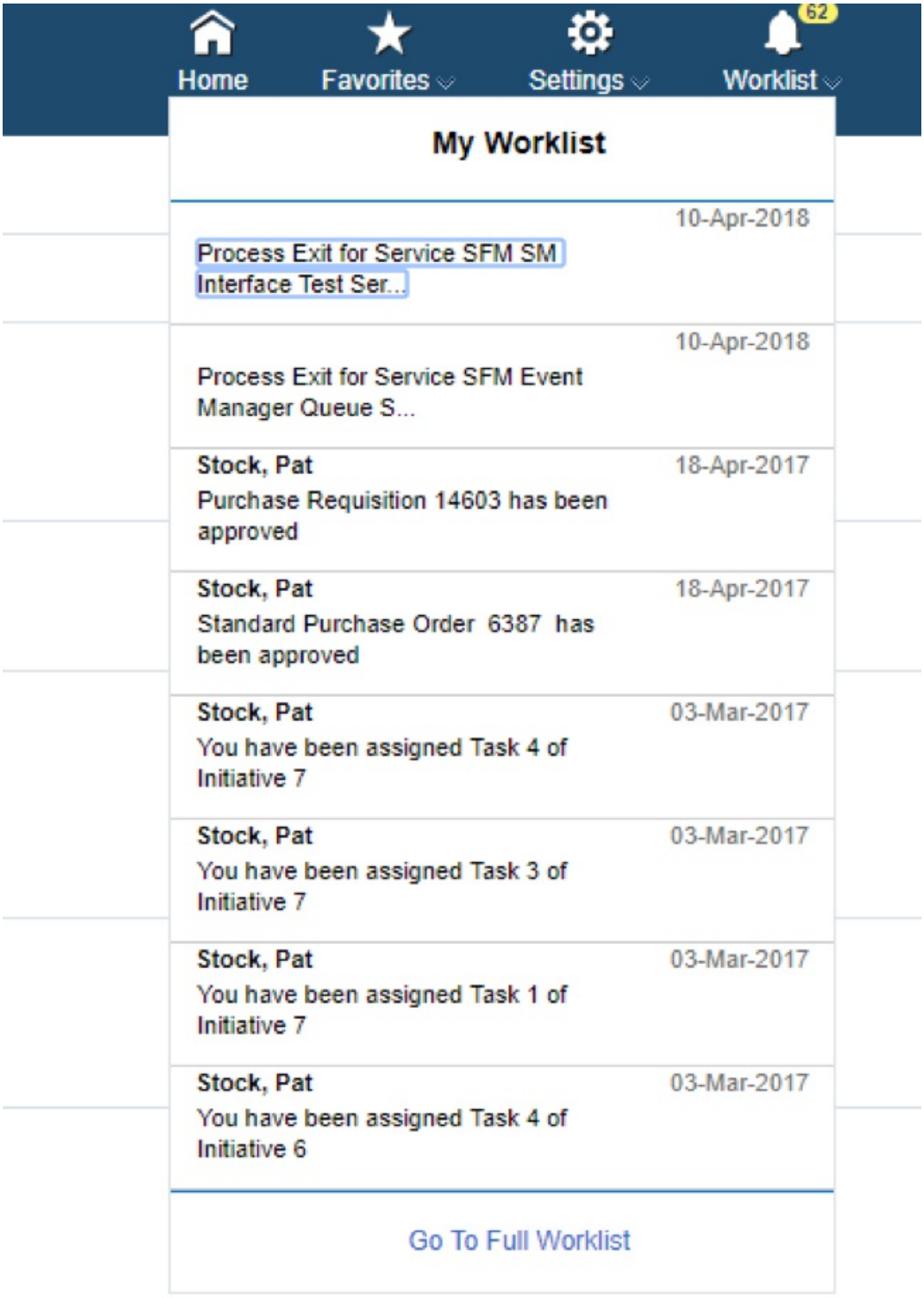
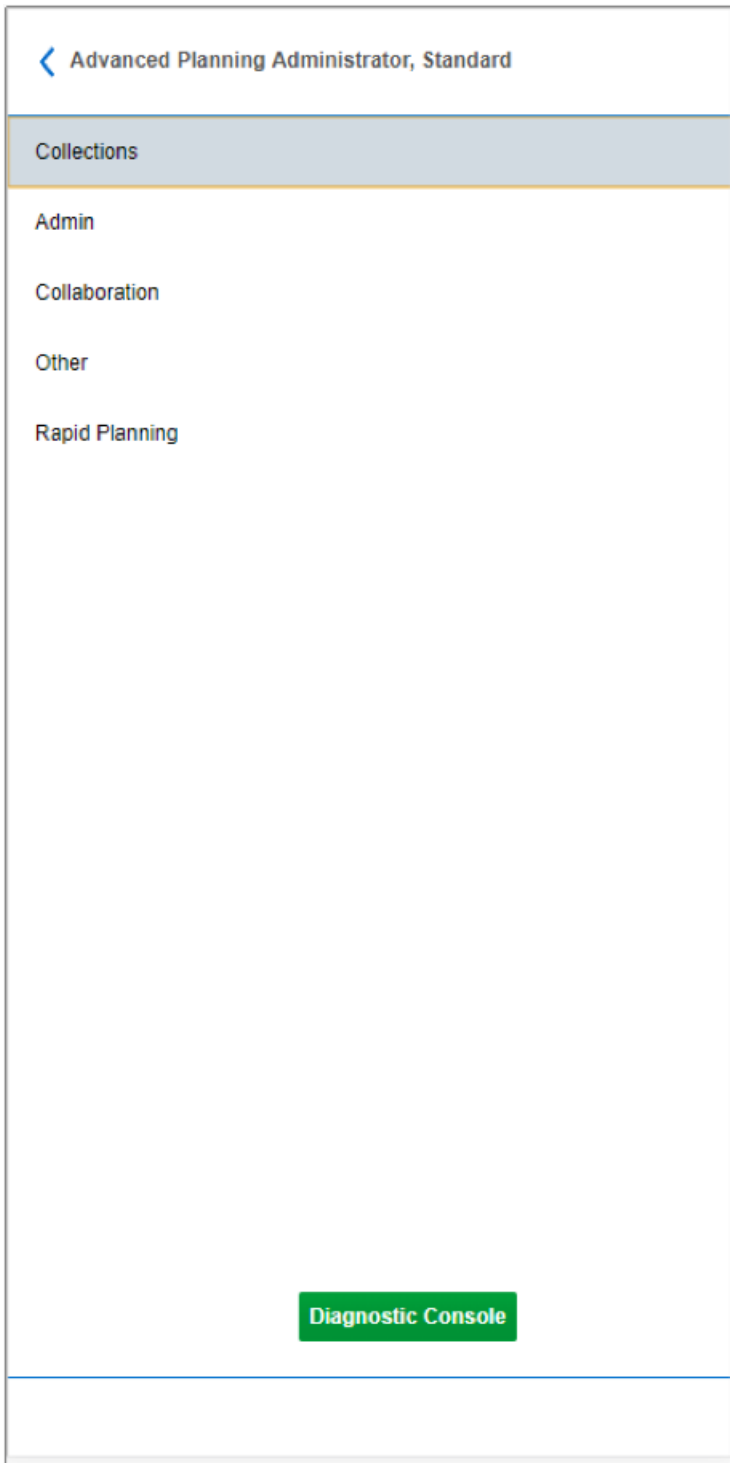


Figure 4: Worklist menu rendered with flat corners



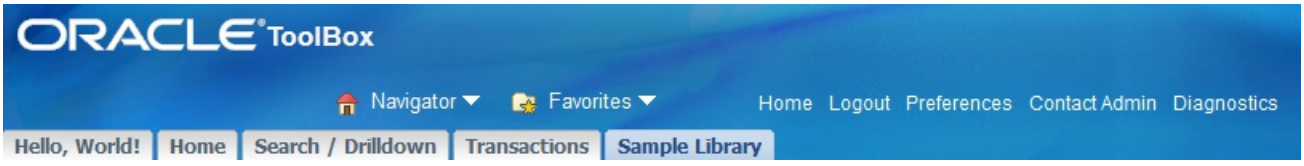
When the page is accessed from a mobile phone, the navigator menu occupies the entire available screen space.

Figure 5: Display on a mobile phone



The parent menu that is clicked to display the child menu is displayed at the top, to help the user understand the context of the displayed menu. Also, the back arrow allows the user to navigate back to previous menus.

Figure 6: Release 12.2.3 and earlier OA Framework ToolBox Tutorial Application global buttons with "Regular" branding



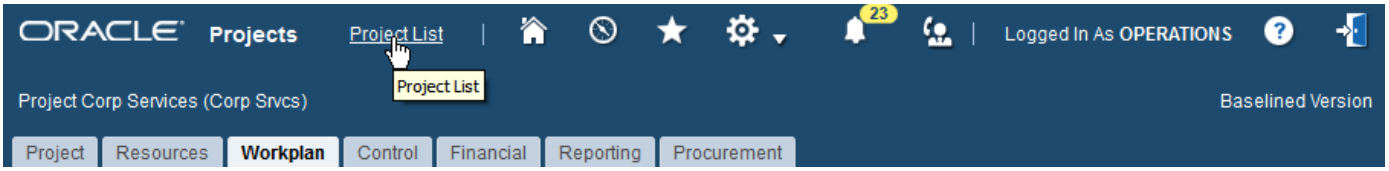
For information about Application Switchers (which appear with the global buttons) see Switchers (Application, Context, Table

Content).

Universal Global Header

A new Universal Global Header, shown in Figure 7, was introduced in Release 12.2.4 to provide a simple and uniform look-and-feel across all mobile, tablet, and desktop devices. The new global header appears in all pages of the applications, and merges all global buttons with standard header information into a single row to provide more display space for page content.

Figure 7: Release 12.2.4 Universal Global Header with iconized global buttons displaying corporate, product, and in-context branding and product-specific global button.



The Universal Global Header supports the following features (appearing in the order listed across the header, from left to right):

- **Corporate, product** and **in-context branding** - as shown in Figure 2, the Universal Global Header displays all branding to the left as in prior releases which includes Corporate branding (Oracle), followed by Product branding (Projects). In-context branding (Project Corp Services (Corp Srvcs)) appears in a separate row below Corporate branding. The organization and user information displays to the right of the window, below the standard global buttons.
- **Product-specific global buttons** - buttons seeded by product teams, if any, appear after the branding information, before the **Home** button. The button appears as text if no icon is defined. In Figure 2 above, **Project LIst** is an example of a product-specific global button with no icon defined.
- **Overflow Indicator** - An overflow indicator appears if in-context branding text exceeds the 60 character limit. Upon hovering over the indicator, the complete text displays.
- **Standard global buttons** - standard global buttons appear to the right of the header and may be set as icons, links or icons and links. The global buttons are (appearing in the order listed across the header, from left to right):
  - **Home** - Appears to the right of any product-specific buttons. Navigates back to the Home page and appears only if the current page is not the Home page.
  - **Navigator** - Displays a rich drop-down navigation menu of responsibilities and functions, with Search input.
  - **Favorites** - Displays a drop-down menu of Favorites.
  - **Settings** - Consolidates these global functions as items in a drop-down menu:
    - **Diagnostics**
    - **Personalize Page**
    - **Preferences**
    - **Manage Proxies**
    - **Access Requests**
  - **Worklist** / **Notification Count** - Displays a list of notifications and a link to navigate to the Full Worklist in a drop-down menu. A count of your notifications appears next to the **Worklist** button.
  - **Switch User** - Appears if a proxy user is set up.
  - **Return to Self** - Appears if a proxy user is set up.
  - **Contact Admin** - Displays the Contact Administrator page.
- **User context** - Displays as "Logged In As ..." after the global buttons.
- **Help** - Displays page-level context-sensitive help.
- **Logout** - Log out of Oracle E-Business Suite.

Enabling the Universal Global Header

The Universal Global Header displays by default as part of the Release 12.2.4 Look-and-Feel. You can control whether the global buttons appear as icons only, links only, or icons and links using the FND: Branding Size profile option. OA Framework displays the global buttons as icons by default. If an icon is not available for a button, then the button displays as link text.

Declarative Implementation

Global buttons are included in an application's menu definition: they are not defined as an OA Frameworkk component region in Oracle JDeveloper. Customers can disable global buttons for specific responsibilities and users by leveraging the Application Object Library function security features.

Tip: Menus are cached. If you change a menu for an application that you're running within JDeveloper, remember to terminate the OC4J server (**Run > Terminate**) and re-run your .jsp page to see your menu changes. If you change a menu for a deployed application (as opposed to one that you're running within Oracle JDeveloper), remember to stop and restart the web application server's listener and JVM to see your menu changes.

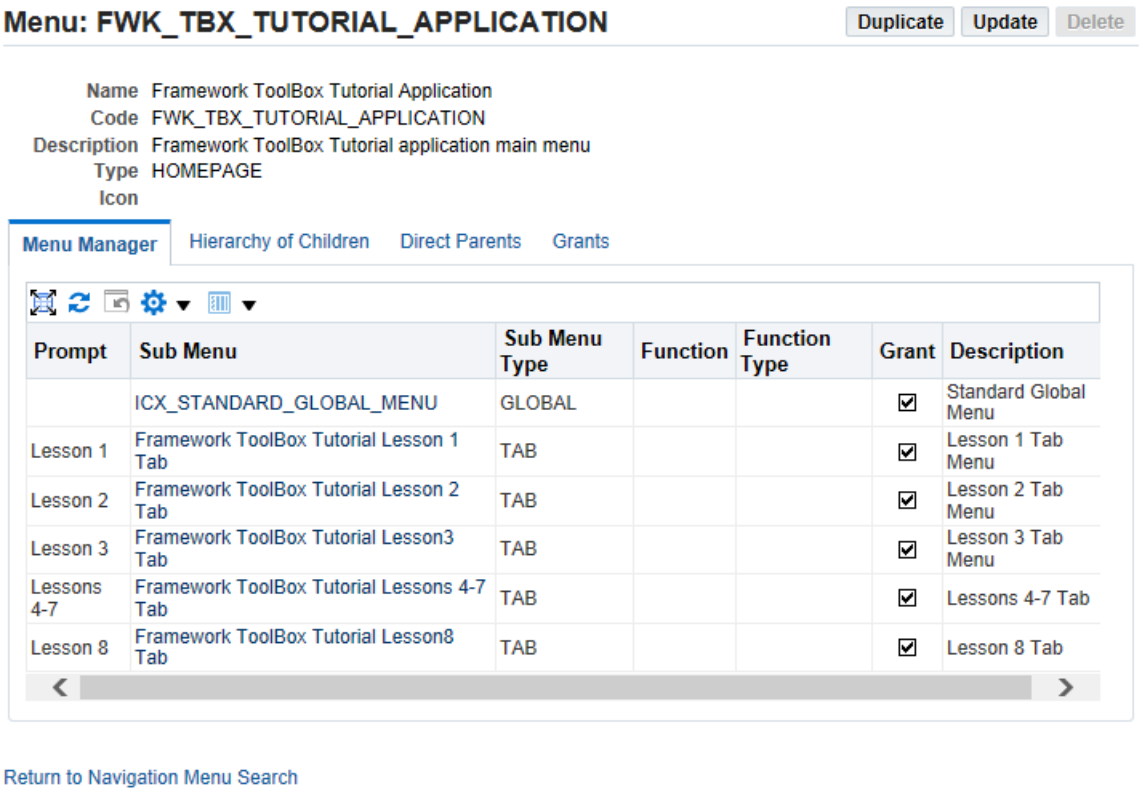
Standard Global Buttons

The standard global buttons that appear in all applications (Home, Logout, Preferences, and Help) are included in a seeded menu called **ICX_STANDARD_GLOBAL_MENU**. To incorporate these buttons in your application, simply add this predefined menu to your application's top-level "Home Page" menu as shown below for the OA Framework ToolBox Tutorial "Home Page" menu. Note that the global submenu does not have an associated prompt.

Note: The ICX_STANDARD_GLOBAL_MENU_G menu has never been supported.

Tip: This document assumes you are familiar with creating OA Framework application menus. If not, you might want to read Tabs / Navigation first.

Figure 3: FWK_TBX_TUTORIAL_APPLICATION "Home Page" menu definition in Oracle E-Business Suite Release 12.



Diagnostics and Personalize Global Buttons

There are two additional "standard" global buttons that render only if corresponding profile options are set:

- As shown in Figure 1 above, the Diagnostics button gives users the ability to view log messages for a page (customers generally use this feature under the guidance of an Oracle Support representative). To enable this global button, set the FND: Diagnostics profile option value to **Yes** at the appropriate level for your needs. See Logging for additional information.
- The "Personalize" button gives users the ability to personalize the current page. To enable this global button, set the Personalize Self-Service Defn profile option to **Yes** at the appropriate level for your needs. See the Personalization Guide for additional information.

Help Global Button

When a user selects the global Help button, OA Framework uses the Application Object Library online Help technology to present page-level context-sensitive help content.

To enable context-sensitive Help for a page, you must specify the `Help Target` property in JDeveloper for the **pageLayout** region. Note that any `Help Target` property set for a child of the pageLayout region is ignored.

- The `Help Target` value must be unique.
- The `Help Target` value must start with an alpha character, and may not be longer than 40 characters.
- The `Help Target` must comply with the following syntax: *`<appShortName>_<packageFunctionalComponent>_<pageName>`* (for example, **FND_tutorial_HelloWorldPG** and **FND_labsolutions_HelloWorldPG**). The *package functional component* in this context helps to clearly differentiate pages given that you could have the same page name in different packages owned by the same product. If necessary, you may abbreviate the functional component name. Never abbreviate the product short code; abbreviate the page name only as a last

resort.
- Remember to coordinate with your technical writer so she can incorporate the target in the page's documentation.

For detailed information about defining and deploying online Help content, see the *Oracle E-Business Suite Setup Guide*.

Product-Specific Global Buttons

You can also display product-specific global buttons (see the BLAF Global Buttons Guideline for limits on the number of buttons that you should add, and criteria that you should evaluate before designing a product-specific global button).

To do this, you must define a custom menu including the buttons you want to display.

1. Add the **ICX_STANDARD_GLOBAL_MENU** to your "Home Page" menu as described above.
2. Define functions for each of your product-specific global buttons as you would for any page that you include in a menu, with one small difference: you must also specify an icon (these images should be added to your `$APPL_TOP/MEDIA` directory).
3. Create a new menu of type **Global** and add your button functions. Remember to specify a prompt for each function (this value displays as the link text).
4. Add your custom global menu to your "Home Page" menu. Do not specify a prompt.

Product-Specific Preferences

When a user selects the global Preferences button, OA Framework displays both general and any product-specific preferences in a side navigation menu (if you add them to a PREFERENCES menu using the following instructions).

Additional Information: Refer to the Oracle Applications User's Guide for details about how to use the general Preferences page.

You can configure the General Preferences Show Flag profile option to hide the General Preferences menu entry if you wish.

For Oracle's in-house Oracle E-Business Suite developers: *If you think you have content to add to the General Preferences menu, please contact the OA Framework team.*

Note: If you do add pages to the General Preferences menu, you must create a global grant to the function's direct parent menu (or permission set). For Oracle's in-house Oracle E-Business Suite developers, you may package that grant to ship with the related page patch to enable the rendering of the page in the General Preferences menu. Note that any other page security requirements are fully honored, as the global grant simply ensures the menu gets rendered with the valid page patch present and will not override more specific security rules.

Warning: You must retain the application module for all preference page menus and transactions, so that when users return to their original page (prior to selecting the Preferences global button), the transactional context is not lost.

The following instructions assume you know how to create menus and functions. If not, read Tabs / Navigation first.

Step 1: Create a menu of type PREFERENCES ("App Pref Menu Container"), and add it to your responsibility root menu. Be sure to leave the prompt null. For example:

FWK_TOOLBOX_TUTORIAL_APP (responsibility root menu)

- ICX_STANDARD_GLOBAL_MENU
- FWK_TOOLBOX_PREFERENCES (Application-specific Preferences container, prompt = "")
- FWK_TOOLBOX_HELLO_TAB (Tab menu, prompt = "Hello, World")
- FWK_TOOLBOX_SEARCH_TAB (Tab menu, prompt = "Search")

Step 2: Create one or more submenus (of any type; OA Framework ignores whatever value you set here) and add them to the PREFERENCES menu *with prompts*. The submenu prompts render as peers to the General prompt. Add page-level functions to these submenus.

For example, to include an application-specific preferences menu entry called "ToolBox Preferences", as shown above in Figure 4, create the following responsibility root menu structure for the OA Framework ToolBox Tutorial Application. Note that this example creates two different preferences pages under the "ToolBox Preferences " menu entry.

FWK_TOOLBOX_TUTORIAL_APP (responsibility root menu)

- ICX_STANDARD_GLOBAL_MENU
- FWK_TOOLBOX_PREFS_CONTAINER (Application-specific Preferences container, prompt = "")
    - FWK_TOOLBOX_PREFERENCES (Peer to "General", prompt = "ToolBox Preferences ")
        - FWK_TOOLBOX_PREF_FUNC1 (prompt = "Some Page")
        - FWK_TOOLBOX_PREF_FUNC2 (prompt = "Another Page")
- FWK_TOOLBOX_HELLO_TAB (Tab menu, prompt = "Hello, World!")
- FWK_TOOLBOX_SEARCH_TAB (Tab menu, prompt = "Search")

Note: Although you can have nested submenus in your PREFERENCES menu, the UI Guidelines recommend that you create as flat a menu as possible. Furthermore, if you want to display only a single page, you can add a function for this page directly to the PREFERENCES menu. Specify a prompt in this case.

Note: All Application (product-specific) preferences (menu of Type PREFERENCES) in the current responsibility are added to the side navigation menu. If there are no product-specific PREFERENCES menu defined, and only one preference page exists, the side navigation menu is not displayed.

Defining a Global Function

When defining a global function, parameters can be specified to allow a popup on a global button. Follow the steps below.

1. In the function definition, choose type JSP and define an HTML call that points to a valid page. This is needed to redirect the page if the popup cannot be rendered, such as from the preferences page.

2. Define parameters as shown in this list:
   a. popup=Y (to mention that a popup needs to be rendered)
   b. regionCode=(code of the region like /oracle/apps/fnd/¿.)
   c. params=nameParam:{@FName}::valParam:{@LName} (: to separate parameter name and parameter value, :: to separate two parameters)
   d. viewName=name of the view to be added to the popup
   e. width=<width of the popup>
   f. height=<height of the popup>
   g. title=<title of the popup>

   (**Note**: Parameters a and b are required, while parameters c and g are optional.)

3. OAGlobalButtonBean implements the OAPopupMaker interface.

4. When constructing the global button, if only parameters a and b are specified:
   a. A popup bean with the specified region code is added to the page hierarchy.
   b. A popup is defined as an embedded popup.
   c. The global button's onclick is set as popup enabled.
   d. The HTML call specified in the function definition is ignored.

5. When constructing the global button, if parameters a, b, c and d are specified and the view instance is accessible under the page layout's root AM:
   a. A popup bean with the specified region code is added to the page hierarchy.
   b. A popup is defined as a parameterized popup.
   c. The global button's onclick is set as popup enabled.
   d. The HTML call specified in the function definition is ignored.

6. When constructing the global button, if parameters a, b, c and d are specified and the view instance is not accessible under the page layout's root:
   a. The global button's destination is set to the HTML call specified in the function definition.
   b. A popup bean is not created or accessible from that page.

Runtime Control

At runtime, OA Framework reads the global menu definition(s), reconciles the standard buttons with your buttons, and instantiates an `oracle.apps.fnd.framework.webui.beans.layout.OAGlobalButtonBarBean`, to which it adds indvidual `oracle.apps.fnd.framework.webui.beans.layout.OAGlobalButtonBeans`. OA Framework then sets the `OAGlobalButtonBarBean` on the `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean` by calling its `setGlobalButtons(UINode globalButtonsNode)` method.

Instantiation

You should not instantiate global buttons yourself: always create them declaratively.

Control Visual Properties

At runtime, there are rare occasions when you might need to do the following (we say "rare" because, by definition, global buttons are universally applicable, so it is fairly uncommon to disable or hide them if a user has function security access):

To hide or disable an individual global button, use the following `processRequest` code:

```
if (< condition >)
{
```

```
 OAPageLayoutBean page = pageContext.getPageLayoutBean();

 // You must call prepareForRendering() if the following code is
 // included in a pageLayout controller.  If you add it further down
 // in the layout hierarchy, the menus will be fully processed so you
 // don't need to do this.

 page.prepareForRendering(pageContext);

 OAGlobalButtonBarBean buttons = (OAGlobalButtonBarBean)page.getGlobalButtons();

 // Note OA Framework automatically assigns global buttons their corresponding
 // function name, so this is how you find them.

 OAGlobalButtonBean button =
  (OAGlobalButtonBarBean)buttons.findIndexedChildRecursive("<FUNCTION_NAME>");

 // Note that you wouldn't do both at the same time.

 button.setRendered(false); // Hide the global button, or
 button.setDisabled(true);  // Disable the global button so it renders, but is not selectable

}
```

To hide all the global buttons, use the following `processRequest` code:

Warning: Hiding all the global buttons is not recommended. You must get UI team approval before doing this, and you must verify that there isn't a declarative alternative: for example, creating multiple "Home Page" menus for your application and alternating between the one with and the one without global buttons.

```
if (< condition >)
{
 OAPageLayoutBean page = pageContext.getPageLayoutBean();
 page.prepareForRendering(pageContext);
 page.setGlobalButtons((UINode)null); // Must come after prepareForRendering()

 // And, if for some reason, you don't want to display any tabs,
 // do the following.
 page.setTabs(null);
}
```

Tip: If you want to create a page with no "chrome" (no tabs, global buttons and so forth -- typically required when you display content in a secondary browser window), it would be better to create the page without a **pageLayout** region. In this case, simply create the page starting with a **stackLayout**, a **header** with a **messageComponentLayout**, or whatever other layout you need for your content. If the page has form elements, you must remember to add a form to your page. For example, a typical layout might look like:

```
stackLayout // top region for the page
|-- form
    | -- everything else
```

Without a **pageLayout** region, OA Framework won't try to display any menu components, so you don't need to programmatically hide them.

Handle Standard Global Button Selection

In some cases, you might need to incorporate custom processing when a user selects a standard global button (the "Logout" button is a common case). Since there is no event point for handling this (for example, selecting the standard "Logout" button forwards to the `OALogout.jsp`), you should programmatically change the destination URL on the "Logout" global button so you can intercept the selection as shown in the following sample code.

Note: The following code is provided to assist in cases when it's absolutely essential that you intercept these standard button actions (and any UI deviations have been approved by the UI Design and Usability team). As a rule, this should be avoided.

Warning: You must preserve the original destination URL of the standard logout JSP, and your custom JSP should forward to this destination URL when it completes its processing. If you don't preserve the correct URL, or you forget to forward to `OALogout.jsp`, you may cause a memory leak (the application module will not be released) and a hung session (the session will not be invalidated).

```
processRequest(...)
{

  // Assume you create a JSP called "MyLogout.jsp" to implement your custom processing.
```

```
  String myLogoutDestination = "MyLogout.jsp&<someParam>=...&<someOtherParam>=...";
  myLogoutDestination = new OAUrl(myLogoutDestination).createURL(pageContext);

  String oldLogoutDestination = null;

  // Find global logout button
  OAGlobalButtonBean logoutButton = findGlobalButton("ICX_LOGOUT");

  // Save old destination url
  oldLogoutDestination = logoutButton.getDestination();
  saveLogoutDestination(oldLogoutDestination);

  //Set the new destination url
  logoutButton.setDestination(myLogoutDestination);

...
...

}
```

In your custom MyLogout.jsp:

```
...


// do some custom processing

// Always redirect to OALogout.jsp when you're done
String finalDestination = getOldLogoutDestination();
<jsp:forward page="<%= finalDestination %>" />
```

Personalization Considerations

- See a summary of Buttons (Global) personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guideline(s):
  - Buttons (Global)
- Developer's Guide
  - Branding
  - Tabs / Navigation
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAGlobalButtonBarBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAGlobalButtonBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`
- OA Framework ToolBox Tutorial / Sample Library.

Buttons (Links)

Overview

This document describes the different kinds of links that you can create for your pages.

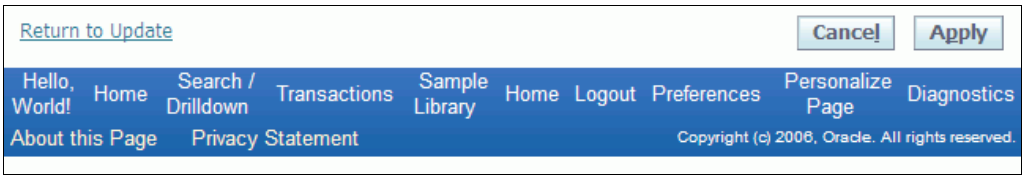
- "Return to" Link
- Display-Only Table Column Link
- Display-Only Text Field Link
- Plain Link
- "Mailto" Action Link
- Form Submit Links

Note: For information about buttons that behave as links, see Buttons (Action / Navigation). For information about embedding links in instruction text, see the Instruction Text document.

"Return to" Link

"Return to..." links render below the page contents as shown:

Figure 1: "Return to..." link and page-level action/navigation buttons below the page contents.



Declarative Implementation

A "Return to... link" is a special named component of the page layout region. To create one:

Step 1: Select your pageLayout region in the JDeveloper Structure pane, right-click and select **New > ReturnNavigation**. JDeveloper creates a **link** item for you.

Step 2: Name the link in accordance with the OA Framework File Standards, set the `Text` to display, and set the `Destination URI` to the target page. For example: `OA.jsp?page=/oracle/apps/dem/employee/webui/EmpSearchPG&retainAM=Y`.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

Instantiate

To add a return link programmatically:

```
processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

 OALinkBean returnLink =
   (OALinkBean)createWebBean(pageContext, OAWebBeanConstants.LINK_BEAN, null, "returnLink");
 returnLink.setDestination("OA.jsp?page=/oracle/apps/dem/employee/webui/EmpSearchPG&retainAM=Y");

 // Retrieve and set the translated link text.
 String linkText = pageContext.getMessage("AK", "FWK_TBX_T_RETURN_TO_POS", null);
 returnLink.setText(linkText);

 // Add the return link to the page.  This example assumes the controller is

 // associated with the pageLayout region.
 ((OAPageLayoutBean)webBean).setReturnNavigation(returnLink);

}
```

Display-Only Table Column Link

Typically, display-only table columns are **messageStyledText** items. To enable the link, simply set the `Destination URI` property and set the `CSS Class` to **OraLinkText**. Then, set the `View Instance` and `View Attribute` properties as you normally would to obtain the link text.

Remember that you can use tokens in your `Destination URI` to obtain parameter values for the request when the link is selected. For example, the following `Destination URI` value obtains the `empNum` primary key from the `EmployeeNumber` attribute in the current row of the associated view object:

```
OA.jsp?page=/oracle/apps/dem/employee/webui/EmpDetailsPG
&retainAM=Y&addBreadCrumb=Y&empNum={@EmployeeNumber}
```

Figure 2: Example of display-only text table column links.



See Implementing the View for detailed information about using URL tokens.

Display-Only Text Field Link

The important characteristic of a display-only *text field* link is it renders aligned with other text fields, and it includes a prompt.

To achieve this, simply create a **messageStyledText** bean and configure its properties as described in the Display-Only Table Column Link section above.

Plain Link

If you want to display a simple link without a prompt, add a **link** item to your page and set its `Destination URI` property. You can either set its `Text` property, or you can bind the link to a view object by setting its `View Instance` and `View Attribute` properties.

"Mailto"Action Link

If you want a link to send an email when selected, for any component that can be configured as a link, simply set the destination property to `mailto:<emailAddress>`.

For example, the `Destination URI` property for the "Buyer" field shown in Figure 3 above is defined as `mailto:{@BuyerEmail}`.

Note the use of token replacement to obtain the `BuyerEmail` value from a view object attribute.

Form Submit Links

If you need to perform specific actions when users select a link or an image, you can configure them to submit the page form instead of simply navigating to their target pages. See Submitting the Form for instructions on how to implement this.

Related Information

- BLAF UI Guidelines
  - Buttons (Links)
- OA Framework Developer's Guide
  - Submitting the Form
  - Implementing the View
  - OA Framework File Standards
  - OA Framework Controller Coding Standards
  - Buttons (Action / Navigation)
  - Instruction Text
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageStyledTextBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OALinkBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`

Charts and Graphs

Overview

The OA Framework provides a user interface (UI) web bean that allows for graphs and graphs with tables. This UI bean is based on the Business Intelligence (BI) Web Bean, but does not provide all the options and configurations allowable in the BI web bean. The Gantt chart is also based on the BI web bean, but further aggregates functionality from the OA Hgrid component.

As of Oracle E-Business Suite Release 12.2.9, the adoption of a new technology introduced significant enhancements to graph delivery. Examples of the differences between this and the previous BI Bean usage are given later in this section.

A new profile option, FND: Enable JET Graph and Charts (FND_ENABLE_JET_GRAPH_CHART), has been created to enable the use of JET graphs instead of BI Bean graphs. Values are 'JET Graph Only' and 'None'. The default value is 'JET Graph Only'.

Contents

The following topics are covered in this document:

- Overview
- Chart and Graph Primer
  - About Graphs
  - About Gantt Charts
  - Supported Chart and Graph Types
    - Bar Graphs
    - Line Graphs
    - Area Graphs
    - Pie Graphs
    - Combination Graphs
    - Secondary Graph Typesß
      - Scatter Graph
      - Stock Graph
    - Gantt Chart
- Graphs
  - Declarative Implementation
  - Runtime Control
  - Laying Out Graphs in a Matrix
  - Using Classic Graphs
  - JET Technology Differences
  - Personalization Considerations
  - Known Issues
- Charts
  - Declarative Implementation
  - Usage Notes
  - Runtime Control
    - Adding Dependency Lines
    - Using FireAction on the Project Column
    - Using Save Model on the Hierarchy Column
    - Per Row Dynamic Poplist
    - Optimizing Child Row Retrieval
  - Personalization Considerations
  - Known Issues
- Related Information

Chart and Graph Primer

About Graphs

The OA Framework Graph web bean is a complex object with numerous variations. It would be impractical to provide examples of each variation. However, we can discuss the general characteristics of the Graph web bean, and develop a common vocabulary to help with your understanding.

Before looking at the different kinds of graphs you can create and the instructions for creating them, ensure you are familiar with the following graph vocabulary.

Figure 2: A Sample Graph with labeled components

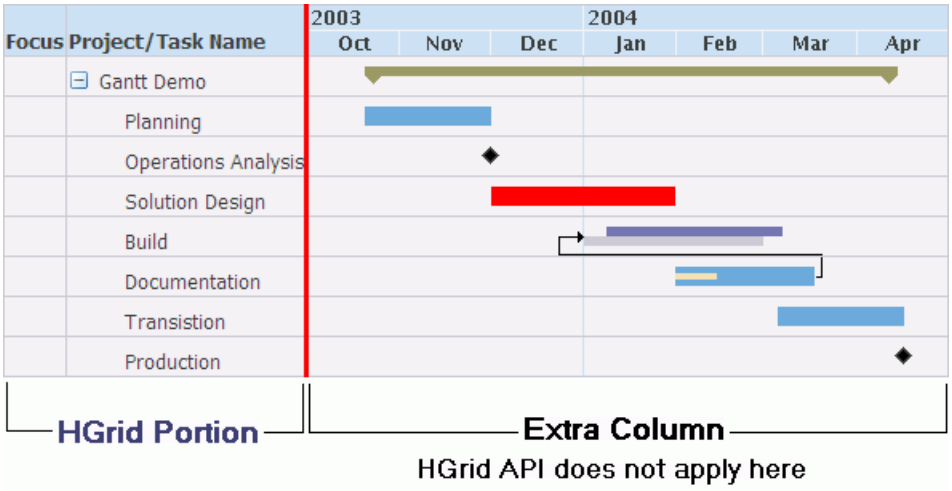


About Gantt Charts

Gantt charts are used for visualizing project planning and scheduling information. A Gantt chart is a matrix. Project tasks that make up the project are listed on the left vertical axis of the Gantt chart, while the total time span of the project divided into increments is listed on the horizontal axis of the Gantt chart.

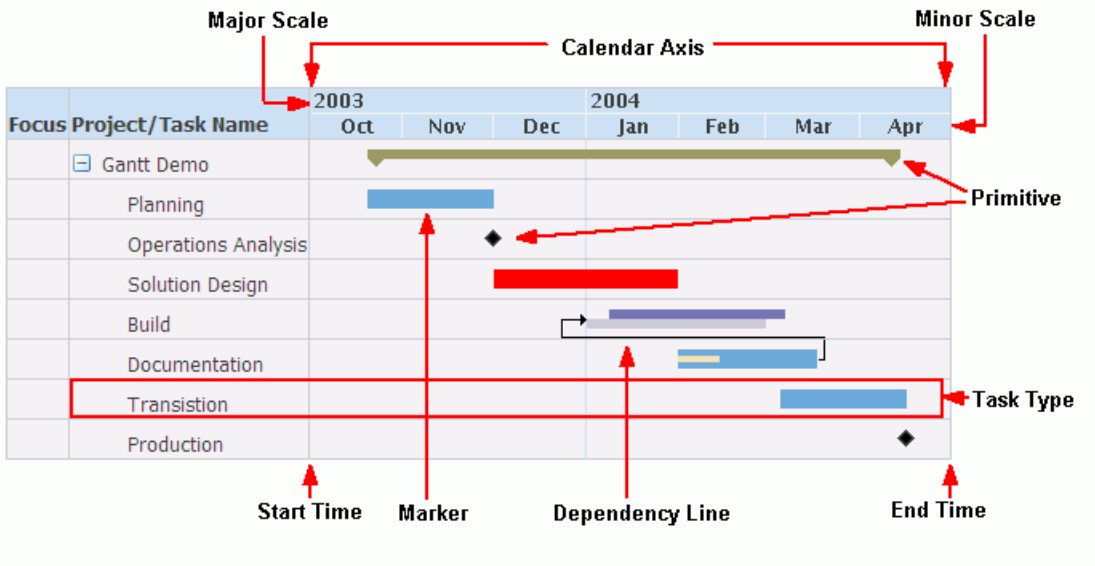
A Gantt chart is based on the HGrid web bean (see Figure 1). An extra column is added to the HGrid to display the graph area that contains the task bars and the calendar axis of the chart. Because this requires a special HGrid configuration, the HGrid web bean is not programmatically accessible. Instead, its properties are exposed at the Gantt web bean level.

Figure 1: Gantt Design Concepts



Familiarize yourself with the following Gantt chart vocabulary.

Figure 3: A Sample Gantt Chart with labeled components



In a Gantt chart:

- The **Project** column is the tree component of the HGrid and lists the project(s) and project tasks.
- The **Resource** column identifies additional information such as who is responsible for the task. This column is set as an indexed child of the Gantt web bean; you can add as many resource columns as needed.
- The last column in the HGrid displays the graph area of the Gantt chart. A row in a Gantt chart HGrid provides the whole visual representation of a single task.
- The **Calendar axis** is a date table appearing at the top of the last column of the HGrid. Horizontal bars, or markers, representing tasks, are positioned relative to this axis.
- The **Minor scale** determines the calendar increments shown on the second row of the Calendar axis. Valid choices are: days, weeks, months, quarters, half years and years. In Figure 3 above, the Minor scale is months. For the minor scale, you can specify a Start Time and an End Time.
- The **Major scale** determines the calendar increments shown on the first row of the Calendar axis. It has the same valid choices as the minor scale. In Figure 3 above, the Major scale is years.
- A **Start Time** and an **End Time** define the time span of the chart.
- A **Primitive** is a visual component defined by its geometric shape, color and pattern.
- A **Marker** is a graphical representation of a task in the graph area. It is composed of up to three primitives representing the task's start, middle and end. A marker's start time and end time are bound to the task's DataObject keys. For example, the marker for a default "normal" task type spans the time returned by the GanttConstants.START key to the time returned by the GanttConstants.END key.
- A **Task Type** is the type of graphical representation assigned to a task. It is represented by one or more combination of markers. For example, the default "milestone" task type is drawn with a single "milestone" marker that is defined as a black diamond primitive.

  There are 4 task types in the default set, they are: "normal" (TaskTypeMap.NORMAL_TASK), summary" (TaskTypeMap.SUMMARY_TASK), "milestone" (TaskTypeMap.MILESTONE_TASK) and "variance" (TaskTypeMap.VARIANCE_TASK).

  - milestone
  - normal without percentage complete specified
  - normal with percentage complete specified
  - summary
  - variance

  You can also add customized Task Types to the TaskTypeMap. The customized Task Type may be constructed with a customized marker.

  For example, you can define a customized Marker called CRITICAL_MARKER, which is red in color, and use it in the customized Task Type, "criticalTask", as such:

  criticalTask

```
Primitive red_bar = new Primitive(Primitive.FULL_BAR, Primitive.SOLID, Color.red);
Marker critical = new Marker(null, red_bar, null, CRITICAL_START, CRITICAL_FINISH);
map.addMarker(CRITICAL_MARKER, critical);
// get default set of Task types
```

```
TaskTypeMap type_map = TaskTypeMap.getDefault();
type_map.addType(CRITICAL_TASK, new Object[]{CRITICAL_MARKER, MarkerMap.PROGRESS_BAR});
```

- A **Dependency Line** is a black line that connects two or more tasks together to show a dependency between these tasks.

Supported Graph and Chart Types

The following list defines the graph types supported by the Graph bean.

- Primary Graph Types
  - absolute area
  - absolute line
  - combination graph
  - horizontal clustered bar
  - horizontal percent bar
  - horizontal stacked bar
  - percent area
  - percent line
  - pie
  - stacked area
  - stacked line
  - vertical clustered bar
  - vertical percent bar
  - vertical stacked bar
- Secondary Graph Types (Secondary graph types are special usage or less common graphs that are associated with particular data types or ways to display unique cases of data. Do not use secondary graph types if the data can be adequately represented by a primary graph type.)
  - point
  - scatter
  - vertical high-low-open-close stock
  - gantt (***see exception note below)

Note: This is a significantly smaller list than the graph types supported by the BI Bean.

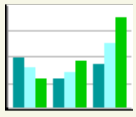
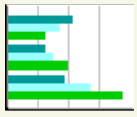
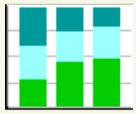
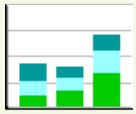
Attention: All the above graphs are created with the **graphTable** region style, with the exception of Gantt charts, which are created with the **gantt** region style.

The following section provides you with thumbnail examples of each graph type, including Gantt charts.

Note: All images in this section are thumbnail images, and are only intended as sample representations of real graphs. It is not recommended that graphs be this small.

Bar Graph

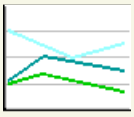
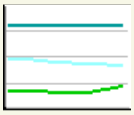
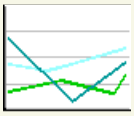
A standard graph with vertical and horizontal axes where data is represented as a series of bars. Subtypes include: clustered bar graph, stacked percentage bar graph, absolute percentage bar graph, and dual-Y graph.

Graph Thumbnail	Graph Type	Description	Usage Notes
 	Vertical and Horizontal Cluster Bar Graph	Each cluster of bars represents a column of data, for easy comparison of the values in a column.	Trends over time. Comparison of items at the same time. Percentage or changes in percentage. Relationship of parts to the whole. Changes in all parts of a whole.
 	Vertical and Horizontal Stacked Percentage Bar Graph	Bars always add up to 100%.	See Cluster Bar Graph. Useful when viewing proportions of a total percentage.
 	Vertical and Horizontal Stacked Absolute Bar Graph	Bars always show absolute values.	See Cluster Bar Graph

	Useful when showing accumulations or cumulative data.

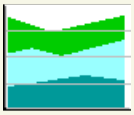
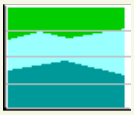
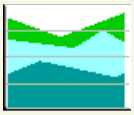
Line Graph

A standard graph using vertical and horizontal axes where data is represented by a line, or by series of data points connected by a line. It is optional to display the marker points. If the graph contains only marker points (and no line), then it is a point graph. Subtypes include: stack line graph, percentage line graph, absolute line graph, and dual-Y graph.

Graph Thumbnail	Graph Type	Description	Usage Notes
	Vertical Stacked Line Graph	A graph in which the lines are stacked. The values of each series are added to the values for previous series. The size of the stack represents a cumulative total.	Shows trends over time. Shows comparisons of items at the same time. Shows rate of data change See Pie Graph for more examples.
	Vertical Percentage Line Graph	Lines are stacked and always add up to 100%.	To show differences or trends in parts of a whole. Shows cumulative relationship between the data (out of 100% or total data values) rather than exact data values.
	Vertical Absolute Line Graph	A graph in which each data marker reflects the exact data values	See Vertical Stacked Line Graph Useful when showing accumulations or cumulative data.

Area Graph

A standard graph using vertical and horizontal axes where data is represented as a filled in area. Subtypes include: stacked area graph, percentage area graph, absolute area graph, and dual-Y graph.

Graph Thumbnail	Graph Type	Description	Usage Notes
	Vertical Stacked Area Graph	Area markers are stacked. The values of each series are added to the values for previous series. The size of the stack represents a cumulative total.	Shows trends over time. Shows rate of data change. Shows percentage or changes in percentage. Shows relationship of parts to the whole. Shows changes in all parts of a whole.
	Vertical Percentage Stacked Area Graph	Area markers show the series percentage and always add up to 100%.	Useful when viewing proportions of a total percentage.
	Vertical Absolute Stacked Area Graph	Each area marker reflects exact data values.	Useful when showing accumulations or cumulative data.

Pie Graph

A graph in which data is represented as sections of one or more circles, making the circles look like sliced pies. Subtypes include: pie, multiple-pie graph, pie bar, ring, multiple-ring, and ring bar graph.

Graph Thumbnail	Graph Type	Description	Usage Notes
	Pie Graph	Graph in which one group of data is represented as sections of a circle, making the circle look like a sliced pie.	Shows relationship of parts to a whole. Shows percentage or change in percentage. Shows changes in all parts of a whole

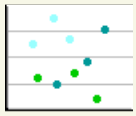
Combination Graph

A graph in which data is represented in a combination of two graph types against a single Y axis. Subtype includes: dual-Y graph.

Graph Thumbnail	Graph Type	Description	Usage Notes
	Combination graph	Emphasizes one or more series of data. Must have at least two series to use this graph type. Shows the relationship of one series to another.	A single graph with one or more graph types. You can have a combination of one or more graph types, where each series plotted as "data" is assigned a combination graph type (bar, line or area). For example, with two series plotted as "data", the first series can be set as bar graph type and the second series can be assigned a "line" type. Most often used as a Dual-Y graph, where not only do different series correspond to a different graph type, but also to different Y axes.

Scatter Graph

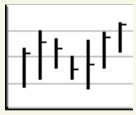
A graph in which data is represented by the location of markers in an area bound by horizontal and vertical axes, where one measure is plotted against another to show correlation.

Graph Thumbnail	Graph Type	Description	Usage Notes
	Scatter Graph	Data is represented by the location of data markers.	Shows correlation between two different measures, or two dimension members in the same measure. Especially useful with a number of data items to show the general relationship between them.

Stock Graph

A graph specifically designed to show, at a minimum, four values for each stock (high, low, opening, closing) during a certain time period. A stock graph may additionally show opening stock volume.

For displaying stock prices over long time periods, it may be preferable to use a line graph, either alone or in combination with a stock graph.

Graph Thumbnail	Graph Type	Description	Usage Notes
	High-Low-Open-Close Stock Graph	Data shows the high, low, opening, and closing prices of a stock. Each stock marker displays four separate values.	Use to show the high, low, opening, and closing prices of a stock.

Note: JET graphs have four mandatory values: High, Low, Open, Close. In contrast, BI Beans graphs only needed High, Low, Close.

Gantt Chart

A graph used extensively in project management applications to track individual events and the overall progress of a complex project.

Graph Thumbnail	Graph Type	Description	Usage Notes
	Gantt Chart	Data is represented by the location and size of data markers. Location indicating a date, and size indicating duration.	Useful for visualizing project planning and scheduling information.

Graphs

Declarative Implementation

To add a Graph web bean to your page, follow these general instructions.

Note: All code that you write (and declare) must comply with the OA Framework Standards and Guidelines in Chapter 8. Please pay special attention to the performance standards.

Step 1: Create an application module for your Graph web bean, if a root UI application module does not already exist. If a root UI application module exists, your work can be done there, or you can create a new application module and nest it within your root UI application module.

Step 2: Create a view object for your Graph web bean, and add it to the application module defined in Step 1. Note that the graph and the data table (if used) must be based on the same view object. All graph columns must also be based on the same view object. As an example, the following query could be used for a graph-related view object.

```
SELECT position_code, count(position_code)
FROM fwk_tbx_employees
GROUP BY position_code
```

In this example, `position_code` defines the column to plot on the X-axis and `count(position_code)` defines the column to plot on the Y-axis.

Note: A graph does not have to plot all columns specified for the graphTable. When you want to render multiple graphs, you can specify each graph to pick a different subset of columns to plot.

Step 3: Add your graph to a page. Select the parent region in the OA Extension Structure pane, then choose **New > Region** from the context menu. (You can alternatively create this as a reusable region).

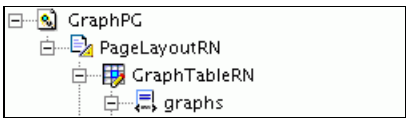
Note: OA Framework does not support graphs under the Detail Disclosure region of a table, under a table or Advanced table region or under a layout region beneath a table or Advanced table region.

Set the following properties on the new region:

- Give the region a standards-compliant `ID`.
- Set the `Region Style` property to **graphTable**.
- Set the `Graph Render Style` property to one of the following:
  - **graph** - to display only a graph. If you define multiple graphs, the graphs render one below the other.
  - **both** - to display a data table as well as a graph, with the graph rendering below the table. If you define multiple graphs, a poplist of graph titles below the table lets you choose the graph to display. The first graph defined is displayed by default under the graph table.
- Create a controller and specify it in the `Controller Class` property. (See the Runtime Control section.)

A default **graphs** container is automatically created.

Figure 4: Sample structure panel after creating a graphTable region



Step 4: OA Extension automatically adds a graph to the **graphs** container. You can also add other graphs to the **graphs** container by selecting the **graphs** container in the Structure pane, and choosing **New > Graph** from the context menu. Set the following properties on the graph. Required properties are marked with an asterisk (*).

- Give the graph a standards-compliant `*ID`.
- Set the `Title` property to the title you want to display for the graph. Long titles are truncated and appended with an ellipse (...).
- Set the `Size` property to the graph size you wish to display:
  - **very-small** - generates a graph of 200 x 125 pixels (width x height).
  - **small** - generates a graph of 295 x 250 pixels. (The default size.)
  - **medium** - generates a graph of 375 x 295 pixels.
  - **large** - generates a graph of 450 x 325 pixels.
  - **custom** - generates a graph of custom size, based on the values, in pixels, that you must specify for the `width` and `height` properties.
- Set the `*Graph Type` property to the graph type you wish to define. Valid values are:
  - **absolute area**
  - **absolute line**
  - **horizontal clustered bar**
  - **horizontal percent bar**
  - **horizontal stacked bar**
  - **percent area**
  - **percent line**
  - **pie**
  - **point**
  - **scatter**
  - **stacked area**
  - **stacked line**
  - **vertical clustered bar - default**
  - **vertical high-low-open-close stock**
  - **vertical percent bar**
  - **vertical stacked bar**
- Set the `*Y Axis Label` and the `X Axis Label` properties as appropriate.
- You can optionally set the `Aggregate Function` property to apply an aggregate function on all columns defined as data, after they are grouped by the specified groupLabels. Valid values are:
  - **none**
  - **avg**
  - **max**
  - **min**
  - **sum**
- Set the `Display Data Markers` property to **True** if you wish to plot the data markers on the graph. The default is False.
- Set the `Display Data Bubble Text` property to **True** if you wish to display bubble text when a user moves the mouse over a data point on the graph. When this property is set to True, the image map for the graph is generated. The bubble text for each data point includes the following: Group, Series, and Value. The default is True.
- For a **combination graph** with multiple series of data plotted as a combination of graph types against a single Y axis, also set the `Allow Combination Graph` property to **True**. For non-combination graphs, set this property to False. Note that you can only set the `Graph Type` property to one of the following for a combination graph:
  - **horizontal clustered bar**
  - **vertical clustered bar**
  - **horizontal stacked bar**
  - **vertical stacked bar**
  - **absolute line**
  - **stacked line**
  - **absolute area**
  - **stacked area graph**

  Attention: If the `Allow Combination Graph` property is set to True on a graph with a Graph Type other than the ones listed above, OA Extension displays a warning at design time and you will also get a run time exception.

- For a **dual-Y combination graph** with multiple series of data plotted as a combination of graph types against two Y axes, also set the `Allow Combination Graph` property to **True** and the `Display Secondary Axis` property to **True**. For non-combination graphs, set this property to False. Note also that you can only set the `Graph Type` property to one of the allowed graph types for a combination graph.
- For a **dual-Y (non-combination) graph** with multiple series of data plotted as one graph type against two Y axes, also set the `Display Secondary Axis` property to **True** and enter a value for the `Secondary Y Axis Label` property. Note also that you can only set the `Graph Type` property to one of the following for a dual-Y (non-combination) graph:
  - **horizontal clustered bar**
  - **vertical clustered bar**
  - **horizontal stacked bar**
  - **vertical stacked bar**
  - **absolute line**
  - **stacked line**

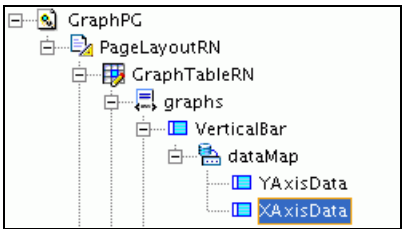
Step 5: Define the columns of the graphTable (graphData) by expanding the default dataMap container beneath the graph in the Structure pane. OA Extension automatically creates two default graphData columns for you and labels them graphDataN. Select a graphData column and in the Property Inspector:

- Give the column a standards-compliant `ID`.
- Set the `View Instance` and `View Attribute` properties to the appropriate view object and view object attribute from which this column gets its data.
- Set the `Purpose in Graph` property to one of the following valid values:
  - **data** - indicates that this column is to be plotted on the Y-axis. You must define one or more data columns. If you define a column as data, you can optionally enter a value for the `Prompt` property. The value of the `Prompt` property is used for the legend when multiple data columns are defined or when no seriesLabels column is defined. See the Notes About Graph Columns for more details.
  - **groupLabels** - indicates that this column is to be plotted on the X-axis as the group labels. When there are many groupLabels, the graph attempts the following options in the order listed to prevent the labels from overlapping:
    1. Stagger labels.
    2. Rotate labels.
    3. Skip labels.
  - **seriesLabels** - indicates that this column determines the Series in the graph. The `View Attribute` property for this column defines the values for the legend.

- For high-low-open-close stock graphs, set the `Stock Value Type` property to **high**, **low**, **open**, or **close** as appropriate. For all other graph types, set the `Stock Value Type` property to **none**.
- For a **combination graph** with multiple series of data plotted as a combination of graph types against a single Y axis, you must map at least two data columns. Set the `Combination Graph Type` property to bar, line or area and optionally specify a value for the `Prompt` property. Then set the `Combination Graph Type` property to bar, line or area for the second data column and optionally specify a value for the `Prompt` property. Repeat the latter step for other data columns that you may have.
- For a **dual-Y combination graph** with multiple series of data plotted as a combination of graph types against two Y axes, you must map at least two data columns:
  - For the first data column, set the `Combination Graph Type` property to bar, line or area, optionally specify a value for the `Prompt` property and set the `Secondary Axis` property to **True** or **False**.
  - For the second data column, set the `Combination Graph Type` property to bar, line or area, optionally specify a value for the `Prompt` property and set the `Secondary Axis` property to **True** or **False**. Just be sure that at least one column has the `Secondary Axis` property to **True.**
  - Repeat the latter step for other data columns that you may have.
- For a **dual-Y (non-combination) graph** with multiple series of data plotted as a a single graph types against two Y axes, you must map at least two data columns:
  - For the first data column, set the `Secondary Axis` property to **True** or **False** and specify an optional value for the `Prompt` property.
  - For the second data column, specify an optional value for the `Prompt` property and set the `Secondary Axis` property to **True** or **False**. Just be sure that at least one column has the `Secondary Axis` property to **True.**
  - Repeat the latter step for any other data columns you may have.

Ensure you have at least one column defined as data, and only one column defined as groupLabels. If you need to define another column as seriesLabels, select the dataMap container and choose **New > graphData** from the context menu. Set the properties as described above.

Figure 5: Sample structure panel after creating a graph



Notes About Graph Columns

- **Pie Graphs** - Only two columns are required to plot a pie graph: data and groupLabels. Values returned by the groupLabels column are plotted as pie slices.
- **Combination Graphs and Dual-Y Graphs** - No column can be mapped as seriesLabel since the series have to be derived from the explicitly defined multiple data columns for the combination graph, rather than the resulting rows of any view object attribute.
- **Scatter Graphs** - By definition, the two columns, data and groupLabels, required for the scatter graph must be number columns.
- **Legend Labels** - The space that a legend area can occupy is one-fourth the size of the graph. If there are too many legends,

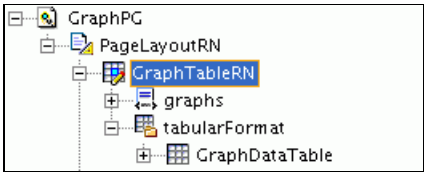
the legend text is truncated and appended with an ellipse (...). The legend displayed for a graph is dependent on its graph column definitions. There are three possible scenarios from which legend labels can be derived:

- **From static data** - If no seriesLabels column is defined for the graph and a column is defined as data, then the value specified in that data column's `Prompt` property appears as the legend label. Note that if no value is specified for the `Prompt` property, then the value returned from the data column's `View Attribute` property is used as the legend label.
- **From query data** - If a column is defined as seriesLabels, the values returned from the `View Attribute` property of this column is used as the legend. If a single data column is also defined, it's `Prompt` property value, if defined, is ignored.
- **From both query and static data** - If a seriesLabels column and **multiple** data columns, with `Prompt` property values are defined for a graph, the legend label is a concatenation of the values specified in the `Prompt` property and the view attribute value returned by the seriesLabels column. For example, if you have 2 data columns with `Prompt` set to Salary and Commission, and one seriesLabels column, returning view attr values of *slvaN* the concatenation looks like:

  "Salary, slva1", "Salary, slva2", "Commission, slva1" and "Commission, slva2"

Step 6: If you want to add a data table to your graph, select your **graphTable** region in the Structure pane, and choose **New > tabularFormat** from the context menu. OA Extension automatically creates a table region for you under the tabularFormat container.

Figure 6: Sample structure panel after creating a tabularFormat showing the table created



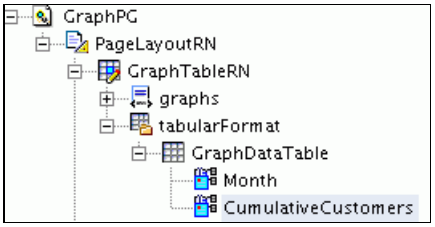
Step 7: For the data table, you can define new columns in the table or extend from an existing table. To extend from an existing table, select the table region in the Structure pane, and in the Property Inspector, set the Extends property to the table you want to reference.

To create a new table, add columns to the table region by selecting the **table** region and choosing **New > Item** from the context menu for each column you wish to create. For each item (or table column), set the following properties:

- Give the item a standards-compliant `ID`.
- Set the `Item Style` property to **messageStyledText**.
- Set the `Prompt` property.
- Set the `View Instance` and `View Attribute` properties.

These table columns should be a superset of the information shown in your graph. For more information about tables, see Tables - Classic.

Figure 8: Sample structure panel with items in table



Runtime Control

To complete your graph implementation:

Step 1: Create a controller and associate it to your graphTable region in the Property Inspector by specifying the controller name in the Controller Class property.

Warning: It is mandatory for the view object associated with the graph to be executed. If it is not executed, a runtime exception is thrown.

Step 2: In the `processRequest` method of this controller class, add an `initQuery` method for the view object you defined for the graph. Any parameters that are set must be bound before executing the query.

Step 3: Add an `initGraphQuery()` method (you may name this whatever you would like) to the application module you defined for your Graph web bean. In this method, call your view object's `initQuery()` method. For example:

```
public void initGraphQuery()
```

```
{
 PositionGraphVOImpl vo = getPositionGraphVO1();

  if (vo == null)
    {
     MessageToken[] tokens = { new MessageToken("OBJECT_NAME","PositionGraphVO1")};
      throw new OAException("ICX", "FWK_TBX_OBJECT_NOT_FOUND",tokens);
    }
  // Per Back Button guidelines, never do a blind query without first checking
  // to see if it's necessary.

  if (!vo.isPreparedForExecution())
    {
     vo.initQuery();
    }
} // end initGraphQuery()
```

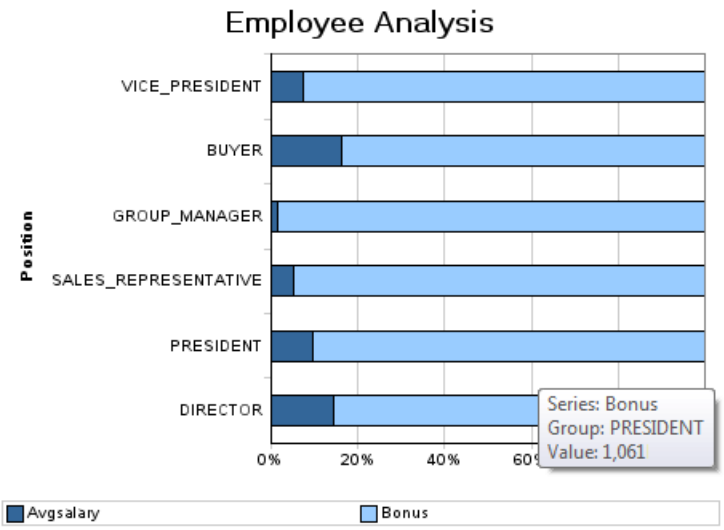
Step 4: In your controller's `processRequest` method, invoke the AM's `initGraphQuery()` method.

Destination URI

You can programmatically define a destination URI on a graph, that allows users to drill down to additional information. If the `Destination URI` property is defined on a graph, the URI specified is used as the drill-down URL. The Destination URI can be an absolute URL or a relative URL to an OA Framework page. The group and series values for each data point are appended to the specified destination URI, so that the resulting URL is specifically associated to a particular data point plotted in the graph. The group and series values can be obtained by the destination page using the parameters OAGgroup and OAGseries.

For the drill down feature to work, you must also set the `Display Data Bubble Text` property on the graph to **True**. Enabling this property generates an image map, which is required for the drill-down feature to work. When a user selects any data point in the graph, the specified URL is launched.

As an example, consider a horizontal percent bar chart with Display Data Bubble Text enabled. The graph displays the bonus versus average salary distribution for a given position. In this case, the series are the Bonus and the AvgSalary values and the group is Position. If you set the `Destination URI` property and enable the `Display Data Bubble Text` property, the OAGgroup and OAGseries automatically append to the Destination URI so that the landing page, a drill-down URL, has the correct context.



Example 1

If you set the Destination URI to `OA.jsp?page=/oracle/apps/fnd/<product>/DrillDownPG`, the resulting URI looks as follows if you select on the Bonus portion of the graph for Position PRESIDENT:

```
http://<server>:<port>/OA_HTML/OA.jsp?
page=/oracle/apps/fnd/<product>/webui/GraphPG&OAGseries=Bonus&OAGgroup=PRESIDENT
```

The drill-down page retrieves the parameters using `pageContext.getParameter("OAGgroup")` and `pageContext.getParameter("OAGseries")` to display details about the Bonus of PRESIDENT.

Example 2

To set an exact value for the Bonus Series on the drill-down URL, use a token substituted value when you set the Destination URI. In the Destination URI example below, BonusAttr is the encoded attribute value:

```
OA.jsp?page=/oracle/apps/fnd/<product>/DrillDownPG&bonus={@BonusAttr}
```

In addition to OAGgroup and OAGseries, the value of BonusAttr also gets passed as a part of the drill-down information, so the resulting URL may look like this, where the retrieved value of BonusAttr is 198722:

```
http://<server>:<port>/OA_HTML/OA.jsp?
page=/oracle/apps/fnd/<product>/webui/GraphPG&bonus=198722&OAGseries=Bonus&OAGgroup=PRESIDENT
```

Setting the Destination URI Programmatically

To programmatically set the drill-down property, you can add code, similar to the sample below, to the `processRequest` method in your controller:

```
//for FWK_TEST_GRAPHSERVLET -Programmatic test for drill down and image map
if(webBean.getUINodeName().equals("region2"))
{
  OAGraphTableBean g1 = (OAGraphTableBean)webBean;
  //Test -If image map is disabled, drill down should not work
  g1.setDisplayToolTip(0,false);
  g1.setDrillDownUrl(0,"http://www.example.com");
}

//Test image map and drill down
if(webBean.getUINodeName().equals("region3"))
{
  OAGraphTableBean g2 =(OAGraphTableBean)webBean;
  g2.setDisplayDataMarkers(0, true);
  g2.setDrillDownUrl(0, "OA.jsp?
page=/oracle/apps/ak/pages/FWK_TEST_AGR_SUM&akRegionApplicationId=601&sal=
{@Salary}");
  //Test line graph with data markers and drill-down
  g2.setDisplayToolTip(1,true);
  g2.setDisplayDataMarkers(1, true);
  g2.setDrillDownUrl(1,"http://www.example.com");
}
```

Laying Out Graphs in a Matrix

A matrix layout for graphs provides you with better layout control when you need to display multiple graphs on a page. Rather than lay out multiple graphs one below the other, you can better utilize the layout of the page by displaying the graphs in a matrix. Although the number of graphs that can be displayed in a row can be set to any valid number, you should follow the BLAF UI guidelines carefully to avoid creating pages that require too much horizontal scrolling.

To lay out your graphs in a matrix:

Step 1: Follow the instructions described in the Declarative Implementation and Runtime Control sections to define more than one graph.

Step 2: In OA Extension, select your **graphTable** region and set the following properties on this region:

- `Graph Render Style` - set to **graph.**
- `Graphs per Row` - set to the number of graphs you wish to display in each row of the matrix.

Example

Suppose you define five graphs in your graphTable region and you set `Graph Render Style` to **graph** and `Graphs per Row` to **2** on the region. When the region renders, it will display two graphs in the first row, two graphs in the second row, and one graph in the third row.

Note The matrix layout for graphs is not applicable if `Graph Render Style` is set to **both**. When this property is set to **both**, only one graph displays beneath the data table, along with a poplist of available graphs.

Using Classic Graphs

To enhanced graphs as the default, the "Use Classic Graphs" property can be set to 'True' at a graphtable level. This is the preferred and recommended approach. Another way to render classic graphs at a page level, where you do not want JET graphs to be

displayed even when the profile level is set to do so, is to update the page XML to add a new user-defined attribute - attribute1 : "BIBEANS" for the pagelayout region. For any pagelayout where user:attribute1 is set to "BIBEANS", a BI Beans graph will be shown instead of a JET graph.

JET Technology Differences

This section describes the use of the JET technology that was introduced for graphs with Oracle E-Business Suite Release 12.2.9.

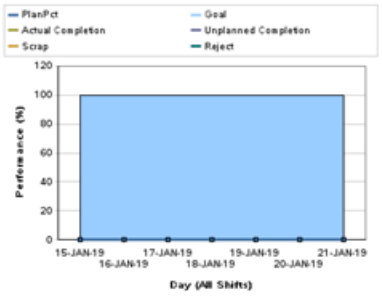
Key Points

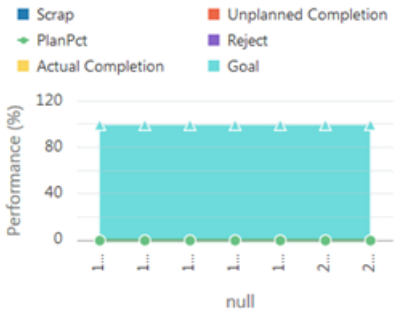
When looking at the examples below, notice that:

- The legend width is greater in BI Bean graph than a JET graph.
- There is a border surrounding the legend area in a BI Bean graph but not a JET graph.
- There is a difference in space management between a BI Bean Graph and a JET graph.
    - In such cases, increasing the graph size should resolve any overlap issues.

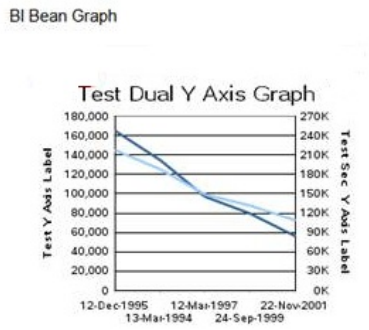
Examples

BI Bean Graph (left) vs. JET Graph (right)

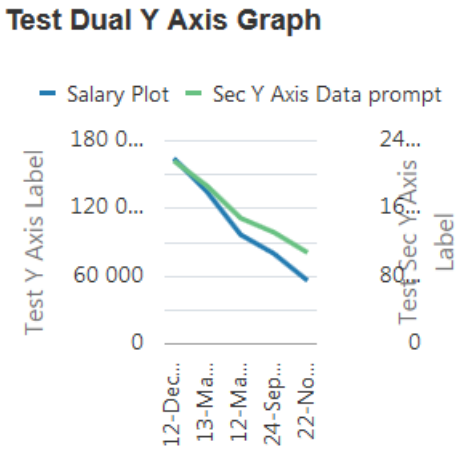


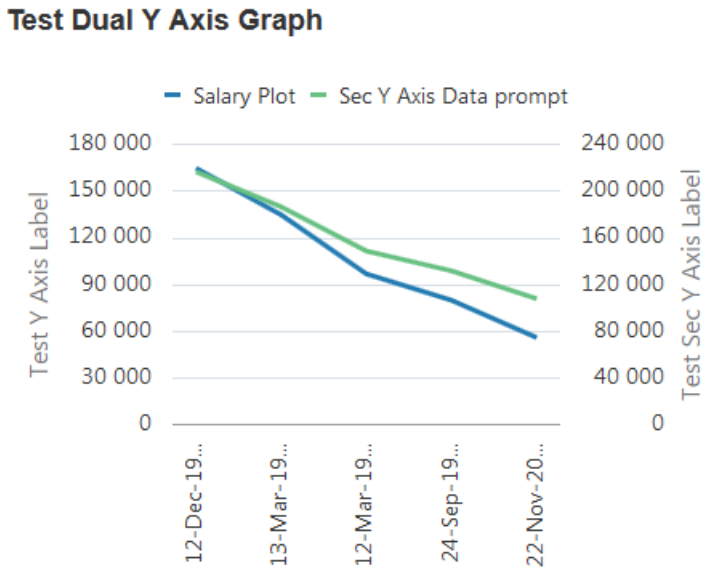


BI Bean Graph

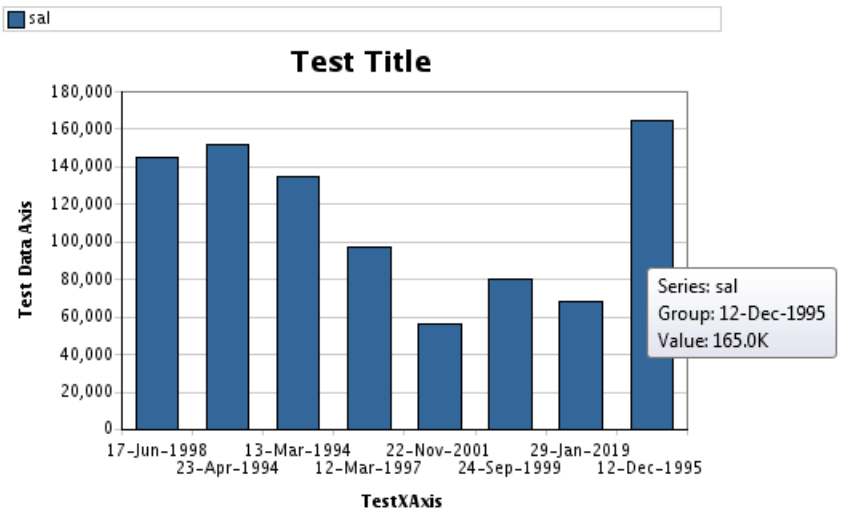


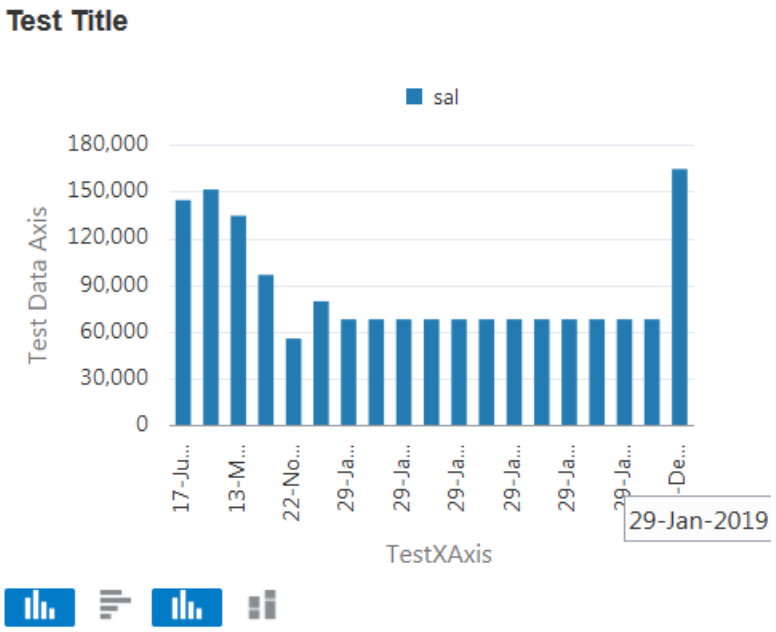
JET Graph (Small) **JET Graph (Large)**





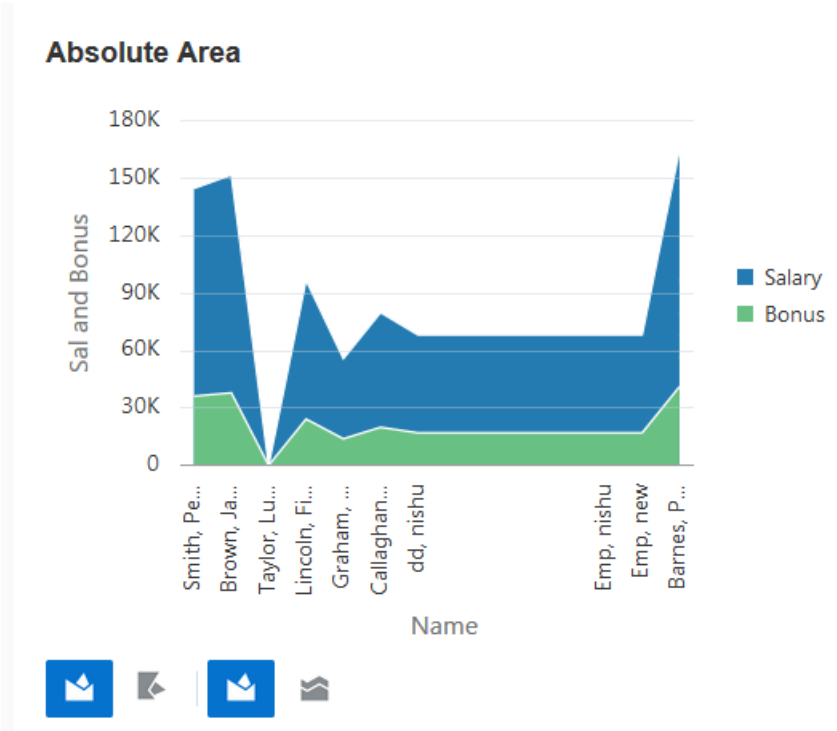
JET graphs do not support multiline labels. The rendering of labels is set to auto mode. JET determines if there is enough space to keep the labels horizontal, or show them vertically with ellipses (...)



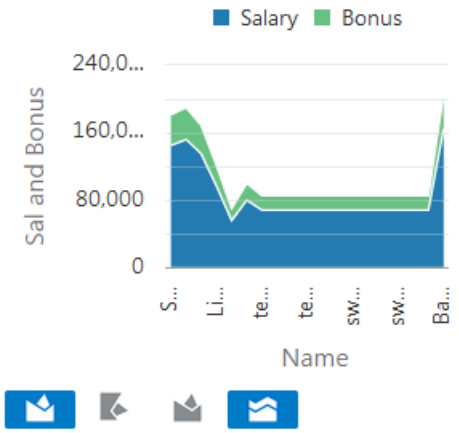


BI Bean graphs displays the legend area at the top, then the title, followed by the image. JET graphs show the title first, then the legend area, followed by the graph. The images above illustrate this, with a BI Bean graph on the left and the equivalent JET graph on the right.

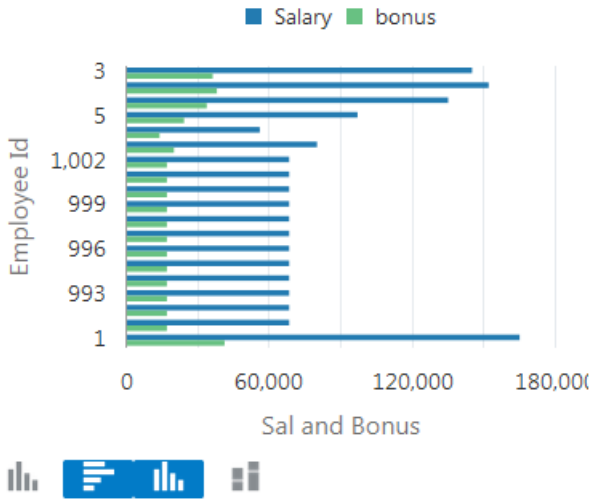
The following examples illustrate some other characteristics and options pertaining to JET graphs and charts.



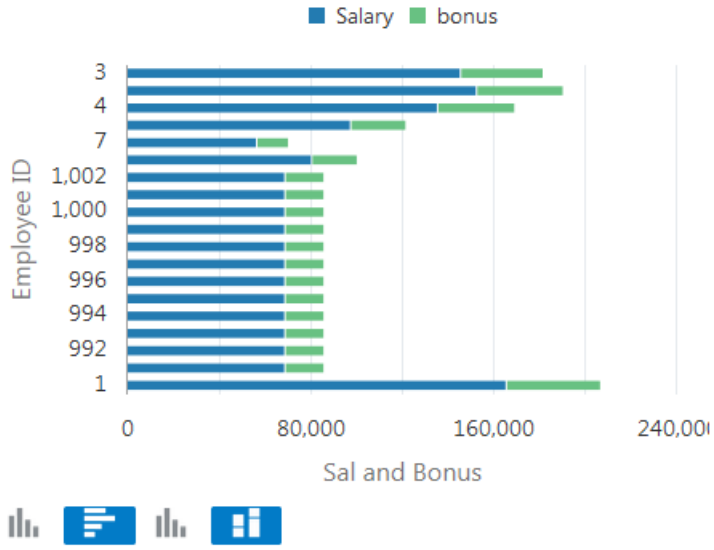
Stacked Area



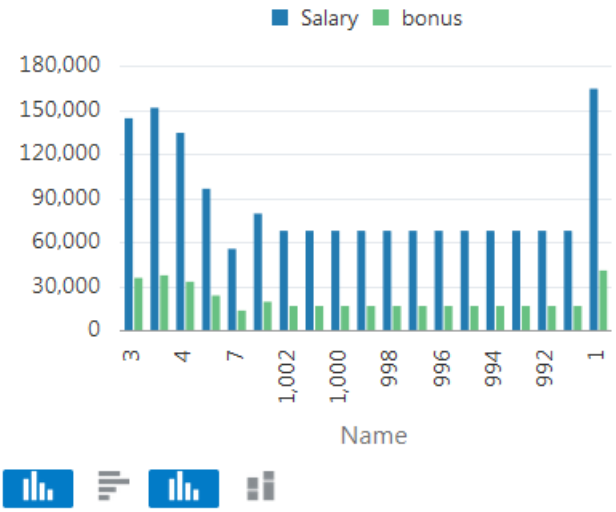
H Clustered Bar



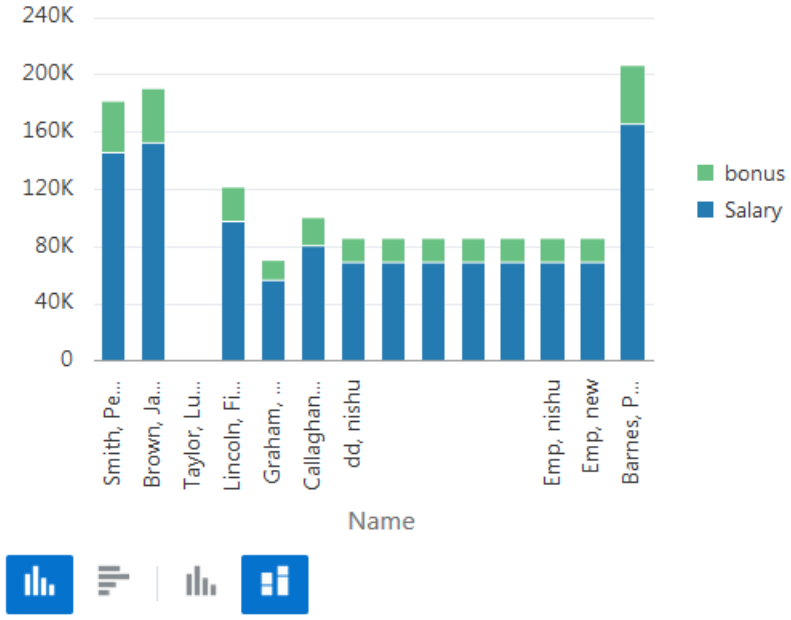
Horizontal Stacked Bar



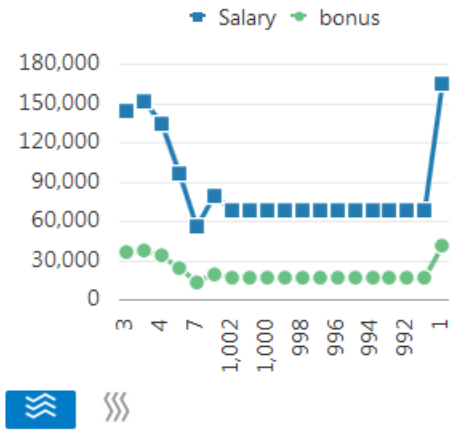
Vertical Clustered Bar



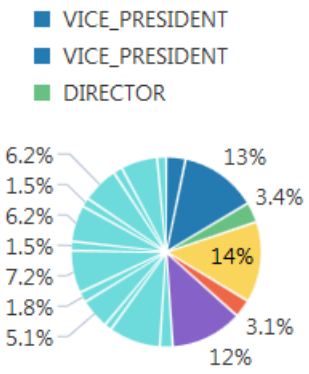
V Stacked Bar



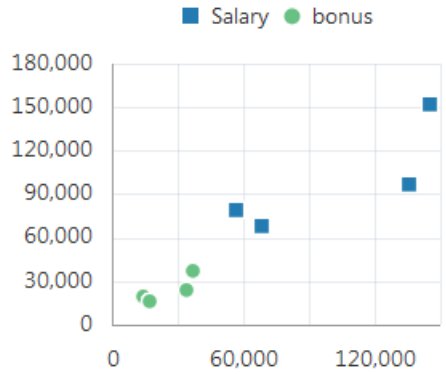
Absolute Line



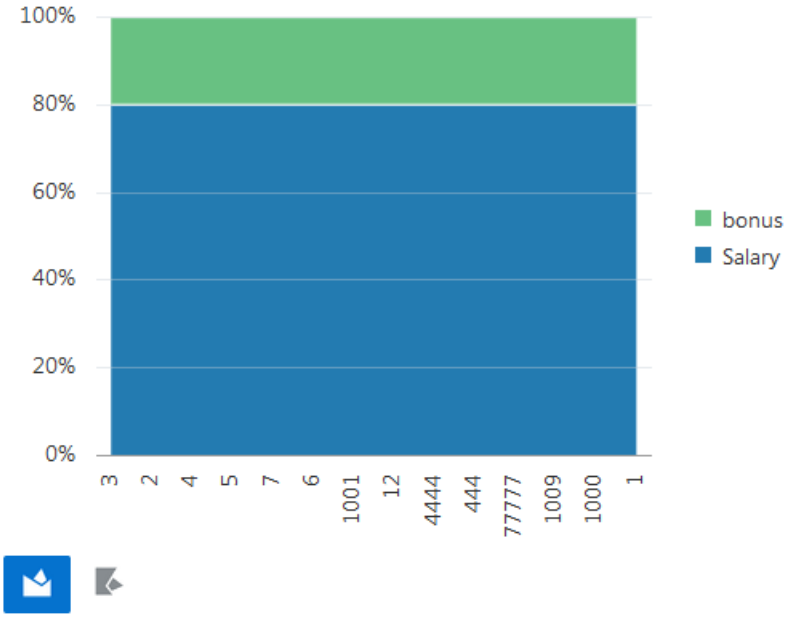
pie



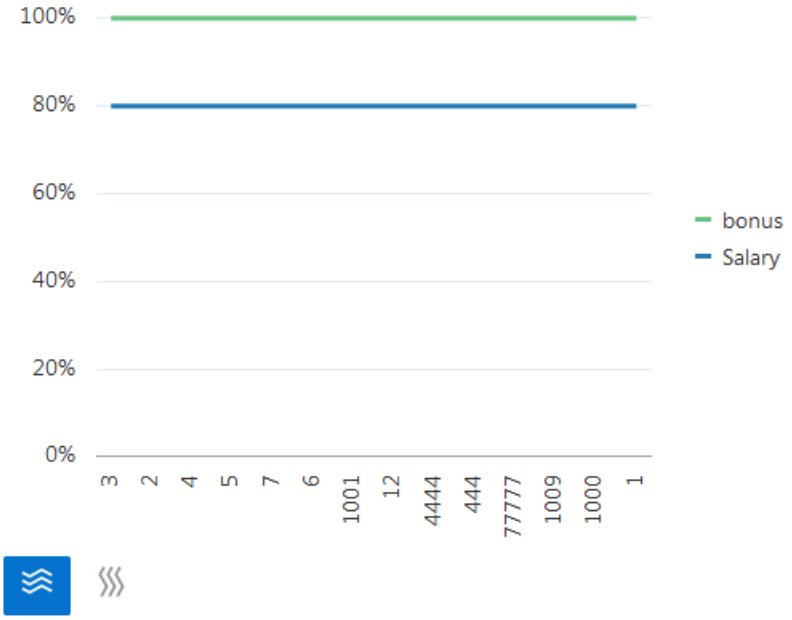
Scatter

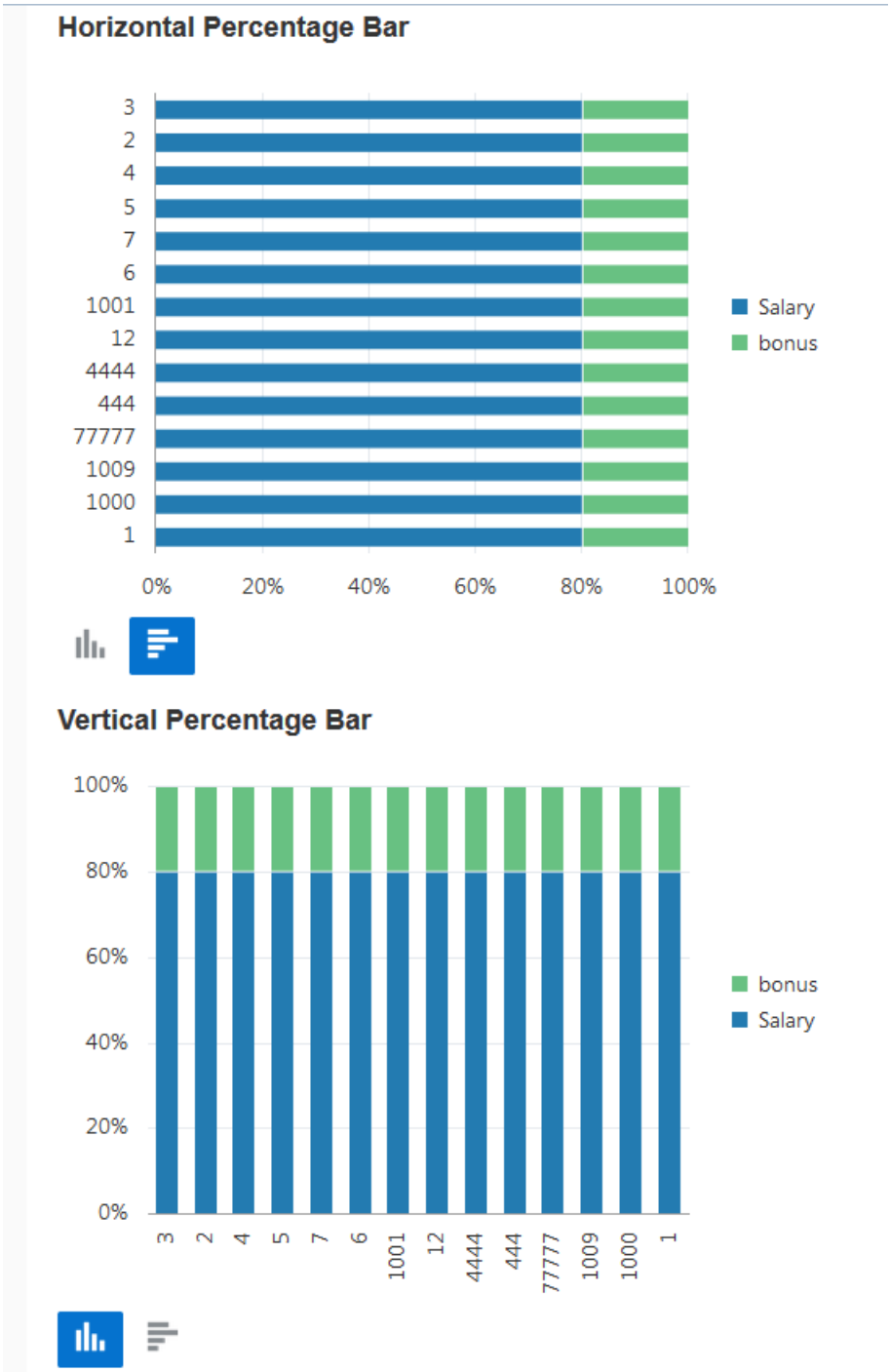

Salary
bonus
180,000
150,000
120,000
90,000
60,000
30,000
0
0
60,000
120,000

Percent Area



Percent Line





Personalization Considerations

- See a summary of Charts and Graphs personalization considerations in Oracle Application Framework Personalization Guide.

Known Issues

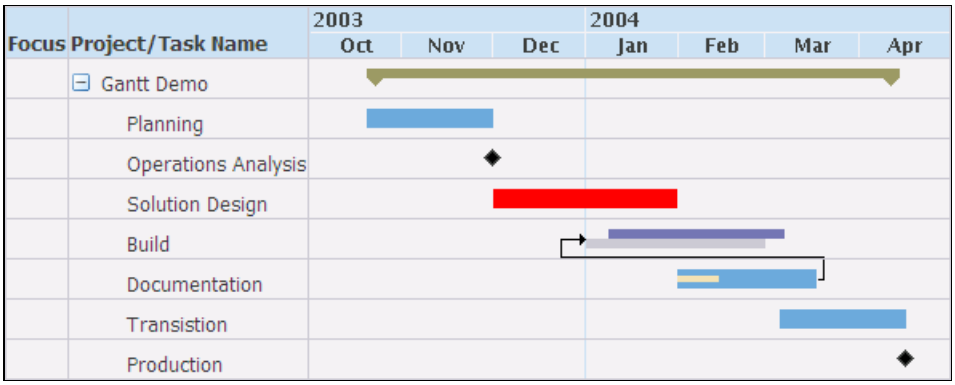
- None.

Charts

Declarative Implementation

To add a Gantt chart to your page, follow these instructions.

Figure 9: Sample of completed Gantt chart



Note: All code that you write (and declare) must comply with the OA Framework Standards and Guidelines in Chapter 8. Please pay special attention to the performance standards.

Step 1: Create an application module for your Gantt chart web bean, if a root UI application module does not already exist. If a root UI application module exists, your work can be done there, or you can create a new application module and nest it within your root UI application module.

Step 2: Create 2 view objects and 1 view link for your Gantt chart, and then add them to the application module discussed in Step 1. The first view object should query the Project/Task Name list. For example, it may look like this:

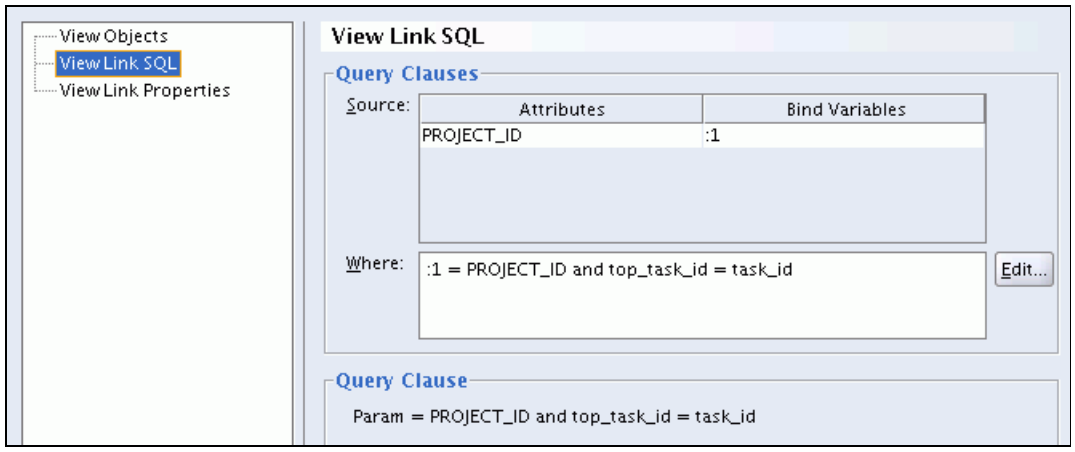
```
SELECT project_id, name, start_date, completion_date,
  start_date start_from, 'summary' task_type,
  completion_date end_to, '' text_right
FROM pa_projects_all
WHERE project_id in (2667, 2225)
```

The second view object should query the details of each task. For example, it may look like this:

```
SELECT project_id, top_task_id, task_id,
  task_number, task_name, scheduled_start_date start_from,
  scheduled_finish_date end_to, task_manager_name text_right,
  decode(scheduled_start_date, scheduled_finish_date, 'milestone',
  decode(top_task_id, task_id, 'summary', 'normal')) task_type
FROM pa_tasks_v
```

The view link should link the tasks (first view object) with the task details (second view object), as shown in the example below:

Figure 10: Example of View Link SQL



Step 3: Add your Gantt chart to a page. Select the parent region in the JDeveloper Structure panel, then choose **New > Region** from the context menu (note that you can alternatively create this as a reusable region). Set the following properties on the new region:

- Give the region a standards-compliant `ID`.
- Set the `Region Style` property to **gantt**.
- The following property values are examples, and are set based on values from our sample view objects shown above.
  - Set the `Task Start Date View Attribute` property to the attribute that returns the start date for the marker. In the example shown in Step 2, this would be the StartFrom view attribute.

- Set the `Task End Date View Attribute` property to the attribute that returns the end date for the marker. In the example shown in Step 2, this would be the EndTo view attribute.
- Set the `Axis Start Date View Attribute` property to the attribute that returns the start date for the axis. In the example shown in Step 2, this would be the StartDate view attribute.
- Set the `Axis End Date View Attribute` property to the attribute that returns the end date for the axis. In the example shown in Step 2, this would be the CompletionDate view attribute.
- Set the `Bar Type View Attribute` property to the attribute that returns the task type for the chart. In the example shown in Step 2, this would be the TaskType view attribute. This is from the standard task types, or any additional custom task types you have added.
- Set the `Right Text View Attribute` property to the attribute that returns the text shown to the right of the marker. In the example shown in Step 2, this would be the TextRight view attribute.
- There are additional properties we don't use in this example, they are:
  - `Left Text View Attribute` property is for any text added to the left of the marker.
  - `Completed Through View Attribute` property is the date that designates how much progress has been completed on a task.
  - `Percent Complete View Attribute` property is the percentage of task complete. Ignored if non-null value is returned for the `Completed Through View Attribute` property. If null is returned for both properties, the progress bar will not be shown.
  - `Actual Start Date View Attribute` property is used for variance task types, and is the start date for the actual bar.
  - `Actual End Date View Attribute` property is used for variance task types, and is the end date for the actual bar.
  - `Baseline Start Date View Attribute` property is used for variance task types, and is the start date for the baseline bar.
  - `Baseline End Date View Attribute` property is used for variance task types, and is the end date for the baseline bar.
  - These following properties are not used in this example but allow you to define dependency lines in the Gantt chart. In order to draw dependency lines, a unique ID needs to be assigned to every task in the Gantt chart. For each task that has a dependency on other tasks, the task IDs of those other tasks must be set as predecessors of the original task. This relationship is defined as a view link.
    - `Show Dependency Lines` - set to **True** to render dependency lines. The default is False.
    - `Task ID View Attribute` - specify the view attribute name that returns a unique task ID.
    - `Predecessor Accessor` - specify the acessor name in the view link for getting the rowset which contains predecessors.
    - `Predecessor View Attribute` - specify the view attribute in the predecessor rowset for getting the task ID of a predecessor.

  Note You must have a working Gantt chart before you add dependency lines to the Gantt chart.

- Set the `Auto Scale` property to **true** or **false**. false is recommended as you set your own scaling with the `Axis Major` and `Axis Minor` properties. When set to true, the `Axis Major` and `Axis Minor` properties are ignored.
- Set the `Axis Major` property to **days, weeks, months, quarters, half-years,** or **years**.
- Set the `Axis Minor` property to **days, weeks, months, quarters, half-years,** or **years**.
- Set the `Show Current Date` property to **true** or **false**.
- Set the `Show Bubble Text` property to **true** or **false**.

Step 4: At this point, it is important to refer back to Figure 1, and reacquaint yourself with the fact that a Gantt chart is essentially composed of 2 components. The timeline side of the Gantt chart is driven off of the Gantt web bean. The project/task list side of the chart is a HGrid component. You have just created the timeline side, now you must create the HGrid for the project/task list side.

Select the **gantt** region in the Structure pane, and choose **New > Tree** from the context menu to create a HGrid tree component. For more information, refer to the HGrid tree component documentation.

- Give the region a standards-compliant `ID`.
- Set the `Text` property to an appropriate value for the column header of your project/task list. For example, use Project/Task Name.
- Beneath your tree component, a member category is created, along with an initial item in that category labeled as nodeDef1. Set the properties for nodeDef1. This node is going to be the first column you want to add to your HGrid side. In our example, this is the Project/Task Name data from the first view object.
- Set the `View Instance` property to your first view object.
- Set the `View Attribute` property to the appropriate attribute from your first view object. For example, in our example, this would be Name.
- Set the `Text` property to an appropriate value for the column header of your project/task list. For example, use Project/Task Name.
- Add a new item to the member category. Right-click on members, and select **New > childNode**. The childNode is going to provide the means to link the tasks with the task details through the view link you created. The childNode is labeled as **childNode1**.
- Set the `View Link Instance` property of **childNode1** to your view link instance. Be sure to use the instance, not just the

view link name.

- Add a new item to the childNode1 category. Right-click on **childNode1**, and select **New > members**. There are 2 items created, nodeDef2 and childDef2. For our example, set nodeDef2, and ignore childDef2 as it is not needed.
- Set the `View Instance` property to your second view object's second instance. This sounds a bit confusing, however, when you add your view link to your application module, both a view link instance (usually VLname1), and a second instance of your base view object (usually VOname2) is created. Since we are tying the data together through the view link, the view object Instance that you must use in this reference is the one used by the view link.
- Set the `View Attribute` property to the appropriate attribute from your first view object. In our example, you would set it to TaskName.
- Finally, because the test data for the Gantt chart resides in another database, change your project settings. Select your project, choose **Project Settings** from the context menu, then navigate to **Common** > **Oracle Applications** > **Runtime Connection**. Change the following settings to reflect the location of your test data:
    - `DBC File Name`
    - `Username`
    - `Password`
    - `Responsibility Key`

Note: If you intend to support the Export feature on a Gantt chart, then different viewAttributeNames cannot be used at different levels in the hierarchy column (Project). All levels of the hierarchy column (that is, all nodeDefs) should have the same viewAttributeName. This is analogous to the definition of all other columns of a HGrid or Gantt. This restriction does not apply if the Export feature is not being used.

Step 5: To define table actions, select your gantt region in the Structure pane of OA Extension. Display the context menu and under **New**, choose the **tableActions**. This automatically creates a tableActions named child consisting of a flowLayout region.

Step 6: Specify a standards-compliant ID for the region and leave the `Region Style` as **flowLayout** or set it to **rowLayout**.

Note: If you have only buttons to add to the table actions area, then you can use either layout styles, flowLayout being preferrable. However, if you are adding message web beans such as messageChoice or messageTextInput, along with buttons to the table action area, then you should use the rowLayout style. Using a flowLayout instead of a rowLayout in this case may cause alignment problems.

Step 7: Under the Layout region, layout the children you want to render as table actions, such as submitButton or messageChoice. Select the Layout region, and choose **New** > **Item** from the context menu. Select the new Item that is created and set the item style as appropriate.

Enabling Search on a Gantt Chart

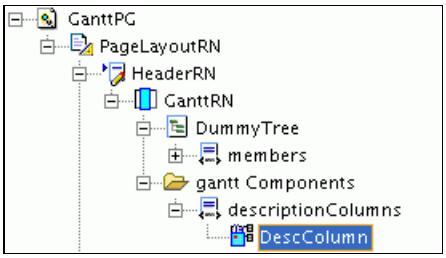
You can enable the ability to search the HGrid component of the Gantt chart. Refer to the Enabling Search on an HGrid section of the Chapter 4: HGrid topic for detailed steps on how to accomplish this.

Usage Notes

Usage in Non-screen reader mode

There is a named child under the gantt component, and it is called **descriptionColumns**.

Figure 11: Example of descriptionColumns named child



This named child allows developers to specify information to be displayed for replacing the graphical component in screen reader mode.

Developers can specify as many **messageStyledText** items as they need beneath it.

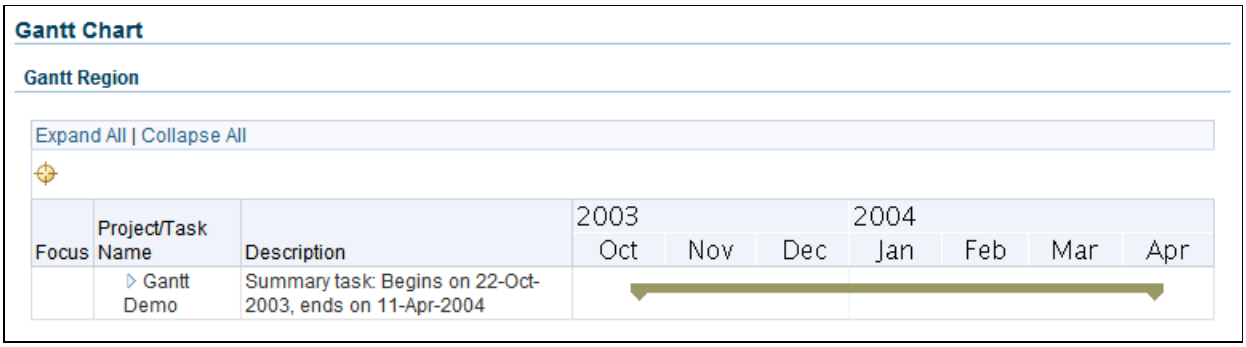
There is an additional property, `Render Description Columns`, on the gantt component. Developers can set it to **true** for rendering Description columns. This can be useful even in non-screen reader mode for people with poor vision.

Figure 12: Example of 1 description column and Render Description Columns set to true.



If the `Render Description Columns` property is set to **true**, and there are no description columns defined, the gantt bean generates a default description for you.

Figure 13: Example of a default generated description



Partial Page Rendering (PPR) in a Gantt Chart

If you map an updateable column in a Gantt chart to a view attribute that is used for drawing the Gantt chart, PPR can be enabled so that you can automatically refresh the Gantt chart. For example, suppose you add a messageTextInput item to a gantt region and its view attribute is also the "start date" view attribute for drawing the Gantt chart. When a user changes the date in the column, the Gantt chart is automatically re-drawn to reflect that date.

Adjusting the Width of the Gantt Chart

OA Framework does not support adjusting the width of a Gantt chart by any means. Even though the Gantt chart is based on the HGrid component (which does allow you to set the width), you should not attempt to alter the width of a Gantt chart by any programmatic means as this may cause distortion of the right image column. For example, if you alter the width of the Gantt chart to 100%, which ties it to the width of the browser, the Gantt chart time line gets distorted when you start to alter the dimensions of the browser window.

Runtime Control

To complete your Gantt chart implementation:

Step 1: Create a controller and associate it with your Gantt chart's parent region.

Step 2: Add an `initQuery()` method to the first view object you used for your Gantt chart.

Step 3: Add an `initGanttQuery()` method (you may name this whatever you would like) to the application module you used for your graph. In this method, call your view object's `initQuery()` method. For example:

```
public void initGraphQuery()
    {
      TaskListVOImpl vo = getTaskListVO1();
      if (vo == null)
      {
       MessageToken[] tokens = { new MessageToken("OBJECT_NAME","TaskListVO1")};
       throw new OAException("ICX", "FWK_TBX_OBJECT_NOT_FOUND",tokens);
      }

      // Per Back Button guidelines, never do a blind query without first checking
      // to see if it's necessary.
```

```
    if (!vo.isPreparedForExecution())
    {
      vo.initQuery();
    }

  } // end initGanttQuery()
```

Step 4: In your controller's `processRequest` method, invoke the application module's `initGraphQuery()` method.

Adding Dependency Lines

If you wish to add dependency lines to your Gantt chart, you should add them declaratively. If you must add them programmatically, you can do so by adding similar lines of code to the processRequest method of your controller:

```
myGanttChart.setDependencyLinesVisible(true);
myGanttChart.setPredecessorsAccessorName("ParentPredecessorsVO");
myGanttChart.setTaskIdViewAttributeName("TaskId");
myGanttChart.setPredecessorsViewAttributeName("TaskId");
```

Using FireAction on the Project Column

If you want to configure the hierarchy (Project) column of a Gantt chart to perform a form submit when selected, see the Using FireAction on the Hierarchy Column section of the Chapter 4: HGrid topic.

Using Save Model on the Hierarchy Column

If you wish to implement a Save Model ("Warn About Changes" dialog with links), on the hierarchy column of the Gantt chart, you must code it manually, by including the following code example:

```
OATreeDefinitionBean webBean = ...
webBean.setAttributeValue(WARN_ABOUT_CHANGES, Boolean.TRUE);
```

Per Row Dynamic Poplist

OA Framework provides programmatic support for the implementation of a choice list (poplist or pulldown) in an updatable Gantt chart, such that its poplist values can vary on a row-by-row basis.

Refer to Dynamic Poplists in Standard Web Widgets for programmatic instructions.

Optimizing Child Row Retrieval

When any given level of a Gantt chart is rendered, the rendering engine must first determine whether the node is expandable so that it can render the Expand icon if necessary. To do this, the rendering engine checks whether the node has children by using the Gantt chart's BC4J view links. This translates to firing the detail view object query just to check for the presence of children, which for large Gantt charts, can be a serious performance drain.

Using treeChildPresentVOAttr

Since data models in Oracle E-Business Suite often have a master row level attribute that keeps track of the presence or absence of children, you can optimize performance in this case. You can instruct the Gantt chart to use this attribute instead of firing a detail view object query to determine whether the expansion icon needs to be rendered. In order to do this, set the `treeChildPresentVOAttr` attribute on the `<oa:childNode>` in the metadata. Unfortunately, since this attribute is currently not supported in metadata, you must set this attribute programatically on the `oracle.apps.fnd.framework.webui.beans.nav.OATreeChildBean`, which is the runtime web bean corresponding to `<oa:childNode>`. For example:

```
OATreeChildBean.setChildPresentVOAttr(String childPresentVOAttr)
```

The String parameter in this case is the name of a master view object row attribute that returns "Y" or "N" to indicate whether there are any children.

Important: All Applications should use this feature to avoid unnecessary queries against the database.

Using treeLevelChildCountAttr

Some Oracle E-Business Suite data models also maintain information about the number of children of a master row, at the master row itself. If you have such an attribute available, you may use it instead of `treeChildPresentVOAttr`. The

`treeLevelChildCountAttr` has two advantages:

- You can use this attribute to determine whether a node has children and hence, whether an expansion icon needs to be rendered for a node.
- For a Gantt chart that uses record navigation, when a node is expanded, you can use this attribute to properly adjust the record navigation links ("Previous" and "Next") without triggering the fetch of all rows being displayed.

In order to do use this attribute, set the `treeLevelChildCountAttr` attribute on the `<oa:nodeDef>` in the metadata. Unfortunately, since this attribute is currently not supported in metadata, you must set this attribute programatically on the `OATreeDefinitionBean`, which is the runtime web bean corresponding to `<oa:nodeDef>`. For example:

```
OATreeDefinitionBean.setChildCountVOAttr(String childCountVOAttr)
```

The String parameter in this case is the name of a master view object row attribute that returns an `oracle.jbo.domain.Number` indicating the number of children. Note that this number must match the number of rows actually returned by the detail view object query.

Personalization Considerations

- See a summary of Charts and Graphs personalization considerations in the Oracle Application Framework Personalization Guide. Also, see a summary of Standard Web Widgets personalization considerations if you plan to implement a dynamic poplist in a Gantt chart.

Accessibility Considerations

- In bar charts, `shortDesc` is automatically populated with a default descriptor that contains series, group, and value information. However, `shortDesc` can also be explicitly set by the application to provide alternative or more detailed information.

Known Issues

- None

Related Information

- BLAF UI Guideline(s)
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAGraphTableBean`
- ToolBox Tutorial Lessons
- ToolBox Tutorial Sample Library

Classic Tables

Note: This document describes the implementation of tables, based on `oracle.apps.fnd.framework.webui.beans.table.OATableBean`. We recommend new tables be implemented as Advanced Tables, based on `oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean`.

Overview

A table is a UI component used for tabular representation of data derived from the contents of a view instance, with accompanying features like column sorting, row-level selection/action, user-controlled row insertion, column totaling, hide/show of supplemental rows and/or column content, and more.

A table can contain instruction text and tip, a navigation bar, selection column, control bar, add row(s) button, column tabulation, and a recalculate button. In addition, each column can be a simple web bean (such as a text input field, a link, a poplist, and so on), or a composite container that groups these simple beans. A common example of a composite container is a flow layout containing an image and a textual description.

Please refer to the Tables and Table Navigation/Action Methods and `OATableBean` javadoc for more guidelines and information about tables.

This document describes how to define a table and discusses how to implement the various features and components of a table:

- Defining a Table
  - Declarative Implementation
  - Runtime Control
    - Event Handling
    - Avoiding Stale Data Checks for Read-Only Tables
  - Personalization Considerations
- Classic Table Accessiblity Support
- Partial Page Rendering (PPR) and Tables
- Table Features and Components
  - Row Headers
  - Column Headers
  - Navigation
  - Selection and Control Bar
  - Table Actions
  - Sorting
  - Adding Rows
  - Totaling
  - Detail Disclosure
  - Table-in-Table
  - Formatting a Table
  - Table Row Dynamic Poplist
  - Table Performance Issues
  - Row-Level Error Prefix
- Export Table Data
- Rich Table Interactions
- Known Issues
- Related Information

Defining a Table

Declarative Implementation

Create a table by specifying appropriate information in Oracle JDeveloper OA Extension. Currently, you can declaratively specify the following features/components/attributes on a table:

- Number of rows to display in a table
- Header text for individual table columns
- Sorting and initial sorting of a table on up to three columns
- Totaling a column
- Single selection and multiple selection on a table
- Detail Disclosure
- Row Headers
- Wrap Settings

Following is a brief outline of how to declaratively implement a simple table in OA Extension. To modify the default behavior of a table or to implement other features of a table, refer to the specific table features and components for more detail.

Step 1: To define a table, create a new region and set its `Region Style` property to **table.**

> **Note:** The table region *must* be a recursive child of a page layout region. The page layout region *must* have its `Form` property set to **true**. This is required as a table is capable of form submissions and therefore should be enclosed in a form.

Step 2: Using the OA Extension Property Inspector, set the following properties on the new **table** region:

Properties	Description
`Region Style`	Set to **table**, as mentioned in step 1. At runtime, OA Framework instantiates an `OATableBean`.
`ID`	Specify an identifier that uniquely identifies the table in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
`Text`	Specify text that appears as the title above the table.
`Additional Text`	To meet the standards of 508 (see Accessibility in Oracle E-Business Suite), specify summary text about the table. This text maps to the HTML summary tag on the table.
`Number of Rows Displayed`	Set to the number of rows you wish to display in the table. The default is **10** as per Oracle Browser Look and Feel (BLAF) Guidelines.
`Controller Class`	(Optional) If you want to handle specific table events and/or perform table-specific actions, you can centralize this type of code in a controller and then specify the name of that controller in the `Controller Class` property for the table. Refer to Implementing the Controller for additional information and examples of Controllers.
`Width`	Set the width of the table in pixels or as a percentage. If you set the Width to **100%**, the Next and Previous links that allow you to navigate among the rows in a table, remain visible above the table, without horizontal scrolling. This is especially useful when you have a wide table that requires horizontal scrolling.

`Message Appl Short Name`	If the table is empty when the page renders, and you want to display a custom message in the empty table, then set this property to the application short name of the product that owns the message to display. The default message that displays is "No search conducted" if no search has been conducted on the table, or "No data exists" if a search has been conducted but no rows were found. Setting this property and the `Message Name` property overrides the default message.
`Message Name`	If the table is empty when the page renders, and you want to display a message in the empty table other than the default message, then set this property to the name of the message to display. For example, you may define a message with the text "Empty table text" in `FND_MESSAGES` under the product FND, and name the message `FND_EMPTYTAB_TEXT`. To override the default message that displays in the table, set the `Message Appl Short Name` property to FND, and set the `Message Name` property to FND_EMPTYTAB_TEXT.

Step 3: While no other properties are required, there are other optional properties you may set on the table to modify the default behavior of your table. You can see the full list of properties on a table in the OA Extension Property Inspector when you create a region of Region Style table. Some of these properties are discussed in more detail when you implement specific features in a table. You can also refer to the Oracle E-Business Suite Component Reference for additional information.

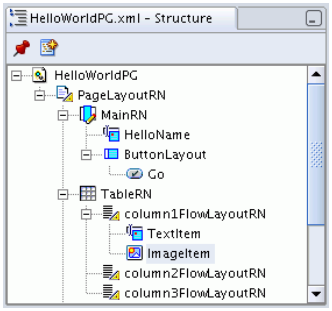
Step 4: For each column you wish to render in your table, define a region and/or item (also referred to as a leaf item) directly to your table region. These regions and/or leaf items are added as indexed children of the table.

You can add regions of the following `Region Style` to a table. These regions group their underlying leaf items into composite columns:

- **flowLayout** - Defines a flow layout region. For example, you can define a flowLayout region to display an image and description.
- **stackLayout** - Defines a stack layout region.
- **hideShow** - Defines a Hide/Show region.
- **messageComponentLayout** - Defines a messageComponentLayout region.
- **switchers** - Defines a table content switcher. A table content Switcher contains two or more leaf items representing possible display alternatives. The item that displays in the Switcher column at runtime depends on the nested child item name that is returned from a specific view object attribute mapped to the Switcher column. Refer to the discussion on Table Content Switchers for more information.

Note: The hideShow, flowLayout, messageComponentLayout and stackLayout regions group their underlying leaf items into composite columns. An example of a composite column is a flowLayout region that displays an image and a description. Another example is a messageComponentLayout region that shows item prompts for the items that are rendered in a cell. The messageComponentLayout region has two leaf items, both of which could be messageStyledText to render a prompt and a value for each cell under the column. The figure below shows an example of a composite column region (Column3FlowLayoutRN) containing two leaf items (TextItem and ImageItem).

You do not have to set any table-specific properties on these composite column regions.



Attention: Defining a graphTable region under a table region or under a layout region beneath a table region, is not supported.

Step 5: The properties you can set on the leaf items of a table vary depending on the item style of the child. You can see the full list of properties relevant to a specific leaf item in the OA Extension Property Inspector. Following is a list of some optional properties you can set, that are common to leaf items of the table:

Optional Properties	Description
`View Instance`	Enter the BC4J view object that provides data for the table. This *must* be the same for all leaf items in a table. **Note:** Unless your tables are read-only, where no events are ever invoked on the tables (navigation, sorting, etc.), you can not have two tables on the same page based on the same view object. Table event handling results in modifications to the state of the underlying view object (range start, range size, etc.) and to the values of the view attributes. If you have two tables based on the same view object on the same page, any event on one table alters the content and state of both tables.
`View Attribute`	Enter the name of the particular attribute within the view object that provides data for this column.
`Read Only`	Set this property to **False** if you want the column to be updateable.
`Total Value`	Set this property to **True** to enable Totaling on this column. See the section on Totaling for additional information.
`Sort Allowed`	Set this property to **yes** to enable sorting on the column. See the section on Sorting for more information.
`Initial Sort Sequence`	Set this property to specify the level at which this column is sorted when the page first renders. Refer to the Sorting section for more information.
`Export View Attribute`	Allows you to specify a different view attribute from which to export data for the leaf item. **Note:** If you specify a value for this property, and a user selects an export function to export data from the table's parent page or region, the data that is exported for this leaf item is from the view attribute specified in the `Export View Attribute` property, and may not be identical to the data displayed.
`Display Vertical Gridlines`	As of Release 12.2.5, specify **True** or **False** to indicate whether to display the vertical gridlines of the table. This property is useful for turning off gridlines in tables displayed on tablet clients.
`No Wrap`	Set this property to **True** to disable wrapping for the cell contents. See Column Formatting for additional information.

Note: Refer to the Oracle JDeveloper OA Extension online help or to the Oracle E-Business Suite Component Reference for more information about the specific properties you can set on the children of a table.

Attention: If you implement a Selection column in your table, note that the Selection column is always rendered as the first column in the table in accordance with the Oracle Browser Look-and-Feel (BLAF) Guideline: Tables. Conversely, if the first column in your table is a checkbox or radio button item, the item will behave as a Selection checkbox or radio button, respectively. OA Framework does not support any other additional checkbox or radio button item in the table.

Runtime Control

When OA Framework first renders your table, it does not automatically execute the query on the view instance for performance reasons. To complete this implementation, you *must* perform one programmatic step. Include code in your controller to bind in your parameters and execute the view instance to populate the table with data. You can find an example of how this is done in `oracle.apps.fnd.framework.toolbox.labsolutions.server.EmployeeFullVOImpl.java` and `oracle.apps.fnd.framework.toolbox.labsolutions.server.EmployeeAMImpl.java`, of Task 3: Implement the View Object Query of the Toolbox Tutorial Lesson 3: Details Page exercise.

Note: If a table is empty, then by default, it can display one of two possible messages. If the query associated with the table's view object has not been executed, the default text that appears in the empty table is "No search conducted." However, if a search has been conducted, but fetches no rows, the default text that appears in the empty table is "No data exists." You can override both of these defaults by including code in your controller to display alternate text if the table is empty.

The table constructs the bean hierarchy, executes any attached controller, and then performs post-processing that sets up most of the default behavior of the table in an `OATableBean` method called `prepareForRendering`.

To change the default behavior at runtime, please refer to the sections describing the table features and components you want to modify. Generally the modifications fall under two possible scenarios with respect to `prepareForRendering`:

Scenario 1: Modify the table before calling the `prepareForRendering` method.

Suppose the modification you plan to make involves calling a public API on the `OATableBean`. The API is invoked in the controller and causes changes in the behavior of the table before `prepareForRendering` is called. In this case, `prepareForRendering` takes these changes into account when it prepares the default behavior of the bean.

An example of this scenario is when you want to include the Add Another Row button to the column footer of a table. To accomplish this, you *must* call `setInsertable(true)` in your controller before calling `prepareForRendering`.

Scenario 2: Call `prepareForRendering` first, then modify the table.

Some modifications require you to explicitly invoke `prepareForRendering` within a controller first and then alter the resulting table. To support this, OA Framework executes `prepareForRendering` once and only once. It detects if `prepareForRendering` has been executed and does not execute it again so as to preserve any modifications you make to the table after explicitly calling `prepareForRendering`.

An example of this scenario is changing the default label on the **Add Another Row** button in the column footer. Your controller code first calls `setInsertable(true)` and then calls `prepareForRendering` to render the **Add Another Row** button, as described in scenario 1. After `prepareForRendering` sets up the default table behavior, your code gets a handle on the column footer by calling `getColumnFooter` and then uses the `setText` API from `oracle.cabo.ui.beans.table.AddTableRowBean` to change the button text.

```
// A controller attached to a region above the table
processRequest(...)
{
  ...

  OATableBean tableBean = (OATableBean) webBean.findChildRecursive("<tableName>");
  if (tableBean != null)
  {
    // enable row insertion
    tableBean.setInsertable (true);

    // prepare table properties
    tableBean.prepareForRendering(pageContext);

    // override the add row button attribute
    OAAddTableRowBean addRowBean = (OAAddTableRowBean)tableBean.getColumnFooter();
    addRowBean.setText("Add 5 Rows");

  }
  ...
}
```

As of Release 12.2.5, you may call the `setVerticalGridlinesDisplayed` API in `OAAdvancedTableBean` to set the `verticalGridlinesDisplayed` property at runtime or call the `isVerticalGridlinesDisplayed` API to get the property value.

Event Handling

There are two event points in the controller at which you can hook in code to modify the behavior of a table: processRequest and processFormRequest.

- **`processRequest`** - As mentioned earlier, OA Framework calls `prepareForRendering` after the `processRequest` phase. You can include code in the controller of the table bean or its parent bean to explicitly call `prepareForRendering` and modify the table.
- **`processFormRequest`** - Table events are HTTP requests that are trapped and processed by OA Framework and handled during the `processFormRequest` phase.

The various table events are:

- Navigation - user selects the **Next** or **Previous** link to navigate between different ranges of rows.
- Sorting - user selects a beveled column heading to sort that column in ascending or descending order.
- Insertion of a new row - user selects the **Add Another Row** button.
- Recalculate column Total - user selects the **Recalculate** button to update the column total.
- Detail Disclosure - user selects the **Hide** or **Show** link to collapse or expand the detail disclosure region.
- Table Control - user selects the **Action/Navigation** button in the table Control bar.

Familiarize yourself with the set of UIX "hidden" fields so you can capture these events and implement custom behavior on them. The "hidden" fields are UIX attributes in the `UIConstants` interface. These parameters are set *only* when a form submit event is induced from a table. They are:

- `SOURCE_PARAM` - indicates the source of the event that is generating the current browser request. This maps to the name attribute of the web bean. If you wish to check whether a table generated an event, you include in your code:

  ```
  if (tableBean.getName().equals(pageContext.getParameter(SOURCE_PARAM)))
  {
    ...
  }
  ```

- `EVENT_PARAM` - indicates the event generated by a web bean (a table, in this case). The possible events generated by a table are:
  - `GOTO_EVENT` - when **Next** or **Previous** navigation links are selected
  - `SORT_EVENT` - when a column header is selected to sort that column
  - `HIDE_EVENT` - when the **Hide** link of a detail disclosure is selected
  - `SHOW_EVENT` - when the **Show** link of a detail disclosure is selected
  - `ADD_ROWS_EVENT` - when the **Add Another Row** button is selected
  - `UPDATE_EVENT` - when the total row **Recalculate** button is selected
- `VALUE_PARAM` - indicates a value that is relevant to a particular event:
  - When a detail disclosure **Hide/Show** is selected, the value parameter contains the row index corresponding to the row whose **Hide/Show** was selected.

    - When the **Next** or **Previous** link of table navigation bar is selected, the value parameter contains the index of the first row of the current range. For example, when the row range 1-10 is displayed, the value is 1 and when the row range 11-20 is displayed, the value is 11.
- `SIZE_PARAM` - indicates the number of rows currently displayed in the table (relevant only to the navigation event).
- `STATE_PARAM` - indicates the current sort state (ascending or descending) of the column on which sorting is invoked (relevant only for the sort event).

Example Usage

To check for the "Add Rows" event:

```
if (tableBean.getName().equals(pageContext.getParameter(SOURCE_PARAM)))
  && ADD_ROWS_EVENT.equals(pageContext.getParameter(EVENT_PARAM)))
{
  ...
}
```

Avoiding Stale Data Checks for Read-Only Tables

When you query a table, OA Framework calls `processFormData` on `OATableBean`, which in turn performs a primary key check between the rows currently displayed in the table and the rows in the view object result set. If a primary key check fails, a stale data error is thrown. In the case of a read-only table that has previously displayed search results, a stale data error is an indication that the submitted table rows from the displayed page cached by the browser and the latest view object rows are based on different query criteria. In other words, a scenario similar to the following took place:

1. The user performed a search to display results in the read-only table.
2. The user updated the search criteria and selected Go to requery the view object.
3. The user selected the browser's Back button to return to the previous results.
4. The user selected Go again, to requery the view object based on the newer search criteria specified in step 2.

For read-only tables, this kind of stale data check could be avoided for performance and browser Back button compliance reasons.

To avoid stale data checks for read-only tables use the following API on `OATableBean`:

```
setSkipProcessFormData(pageContext, true);
```

Warning: Do not call this method on updatable tables, as setting `skipProcessFormData` to true will clear all the changes done on the table.

Personalization Considerations

You can personalize limited aspects of a table.

You can personalize the `Row Header Column View Attribute` property of a Classic Table to render a separate column as a Row Header.

See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Classic Table Accessibility Support

To support accessibility for screen readers, you must identify a column in a classic table as a row header. Every row in the table must refer to a column as the row header that uniquely identifies the row so the screen reader can read it. Screen reader users must also define their Accessibility mode by setting the profile Self Service Accessibility Features to **Screen Reader Optimized**.

The underlying UIX `oracle.cabo.ui.beans.table.TableBean` supports accessibility by defaulting a table's first column as the row header. However, you can set a different column as the row header, especially if a user is likely to personalize the table by turning off rendering of the first column or if the first column can return no data. Note that for a classic table, you may only set an immediate child under the table as a row header column, and not a recursive child item under an immediate child.

To declaratively set a column as a row header, in the OA Extension Property Inspector, specify a column ID (String) for the `Row Header Column` property (XML attribute: `rowheadercol`) of the classic table region.

To programmatically set a column as a row header, use the following API on `OATableBean`:

```
setRowHeaderColumn(<columnName>);
```

As of Release 12.2, you may also personalize the `Row Header Col` property of a Classic Table to identify a column as the Row Header for screen reader accessibility.

Similarly, you must also specify column headers for your table. A data table needs column headers to meet accessibility guidelines. If you wish to implement a table without column headers, design it as a tableLayout region.

Note: To support accessibility for screen readers, the detail disclosure of a table row is created as part of that table row's cell rather than as a table within a separate detail row.

Partial Page Rendering (PPR) and Tables

In certain cases, you may want to include a mix of updatable and non-updatable rows in a table. This is accomplished by using partial page rendering at the row level for items in a table (as well as Table-in-Table). Refer to the Dynamic User Interface document for more details. You should first familiarize yourself with how to implement PPR in general, then refer to the section on PPR and Tables.

Table Features and Components

This section discusses the following table features and components:

- Row Headers
- Column Headers
- Navigation
- Selection and Control Bar
- Table Actions
- Sorting
- Adding Rows
- Totaling
- Detail Disclosure
- Table-in-Table
- Formatting a Table
- Table Performance Issues
- Row-Level Error Prefix

Row Headers

Previously, there was no declarative support to define row header information declaratively and as a result, these were not automatically built. OA Framework now provides declarative support of row headers in OA Extension.

	Column	Column
Row	Cell	Cell
Row	Cell	Cell
Row	Cell	Cell
Row	Cell	Cell
Row	Cell	Cell

Declarative Implementation

To implement a row header, you need only specify a value for the `Row Header View Attribute` property on the table in OA Extension.

The `Row Header View Attribute` is the view associated with the table that determines the values for the row headers. Just as the table column view attribute supplies values for the cells of that column on query execution, the `Row Header View Attribute` supplies values for the row header cells.

Runtime Control

You can also set the row header programmatically in the controller using the accessors provided on the table bean:

```
// To get the row header view attribute, if any
tableBean.getRowHeaderViewAttributeName();

// To set the row header view attribute
tableBean.setRowHeaderViewAttributeName();
```

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Column Headers

The column header provides the heading for the column and is defaulted from the Prompt value of the table children and is always left aligned.

Select	First Name	Last Name	Salary	Start Date
☐	Peter	Smith	145000	17-Jun-1998
☐	James	Brown	152000	23-Apr-1994
☐	Lucy	Taylor	135000	13-Mar-1994
☐	Fiona	Lincoln	97000	12-Mar-1997
☐	Charles	Graham	56000	22-Nov-2001

Declarative Implementation

The header text of a column is set by the value that you enter in the `Prompt` property for the corresponding item in OA Extension.

Runtime Control

The following three UIX properties associated with the `UIX TableBean` enables column headers in a table:

- `columnHeaderStamp`
- `columnHeaderData`
- `columnHeaderFormats`

To turn off column headers that OA Framework adds by default, use the following code example.

Note: The code renders the table inaccessible.

```
processRequest
{
  ...
  tableBean.prepareForRendering(pageContext);
  tableBean.setColumnHeaderStamp(null);
  ...

}
```

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

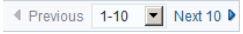
Navigation

The navigation bar allows you to traverse across different row ranges of a table.

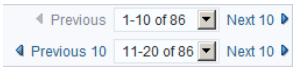
The navigation bar is rendered:

- At the top of the table if the number of rows in the table is less than 10
- At both the top and bottom of the table if the rows in the table is equal to or greater than 10.
- No navigation bar is displayed if the number of rows in the view instance is less than the table `Number of Rows Displayed` property.

Note: When a user first navigates to the page, OA Framework does not know how many rows are being returned to the table. The navigation bar simply shows the **Previous** and **Next** links.



Once the user navigates through all the rows, the navigation bar displays the row range as a poplist so you can navigate directly to a specific range of rows, as shown below:



Declarative Implementation

There are no declarative steps necessary to implement the default behavior of the navigation bar.

Note: If you have a wide table that requires horizontal scrolling, you can prevent the **Next** and **Previous** links from appearing off the screen, to the right, by setting the `Width` property of the table definition to **100%**.

Runtime Control

For event handling, you can specify the following code in your controller's `processFormRequest` method to check for a navigation event:

```
if (GOTO_EVENT.equals(pageContext.getParameter(EVENT_PARAM))
{
  ...
}
```

To check whether the **Next** link or **Previous** link is selected:

```
if ((tableBean.getName().equals(SOURCE_PARAM)) &&
  (GOTO_EVENT.equals(pageContext.getParameter(EVENT_PARAM))))
{
  String value = pageContext.getParameter(VALUE_PARAM);
  if (value != null)
  {
    int val = Integer.parseInt(value);
    int newRangeStart = val - 1;
    if (tableVO.getRangeStart() < newRangeStart)
    {
      // next pressed
      ...
    }
    else
    {
      // previous pressed
      ...
    }
  }
}
```

Navigation and Performance

The table rendering logic brings in only the rows you need in an incremental fashion. In other words, if your table display size is 10 rows, then only the first 10 rows from the query are brought into the middle-tier. If you press the **Next** link, another 10 rows are brought in from the database through the JDBC cursor. This is called Incremental Fetch logic. Time spent on network round trips is conserved by bringing in rows only on demand.

This is also the reason why the total row count is unknown when the table is first rendered and the poplist in the table navigation area does not initially appear. Once the fetch is complete, a poplist always displays, even as you navigate back and forth with the **Previous** and **Next** links.

Personalization Considerations

Not applicable.

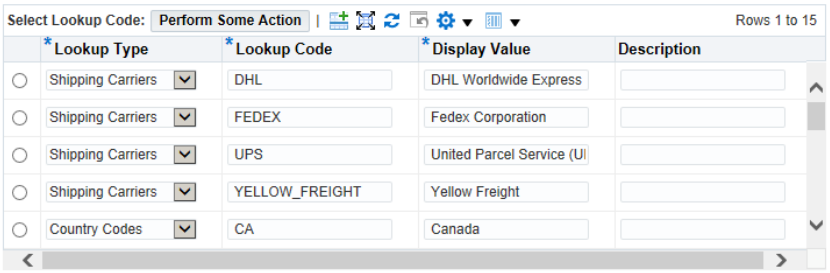
Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Selection and Control Bar

Table selection refers to the ability to select specific rows in a table. If single selection is enabled for a table, OA Framework renders a control bar and a Selection column that displays a radio button.

Table with Single Selection Radio Button



If multiple selection is enabled for a table, OA Framework renders a control bar and a Select column that displays a checkbox. Users can use the Select column to select a specific row or rows and then choose a control such as a button or poplist on the control bar to apply an action on the selected row(s).

Beginning in Release 12.2.6, for multiple selection, a new tri-state check box appears above the Select column to help you quickly select or deselect all or some of your rows as well as visualize the partial selection of your rows. Note that all disabled rows are excluded from selection.

Table with Multiple Selection Checkbox Where All Rows Are Unselected

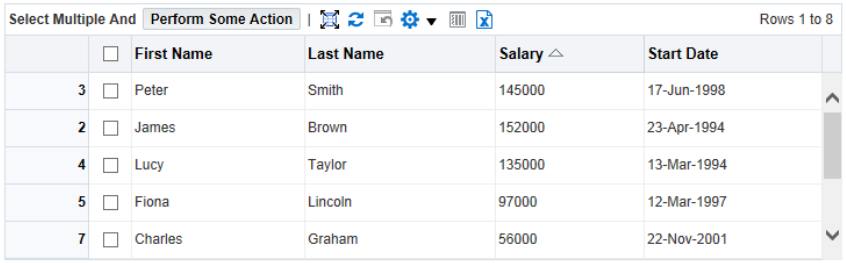
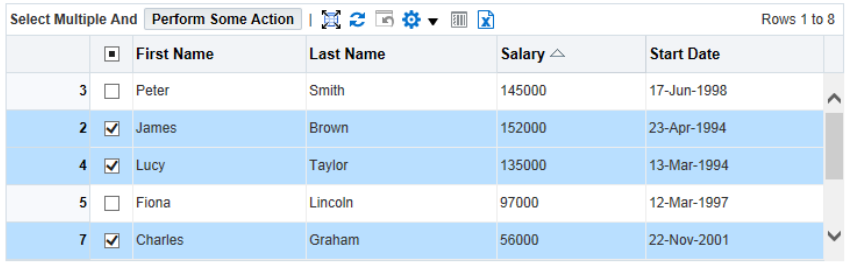


Table with Multiple Selection Checkbox Where All Rows Are Selected



Table with Multiple Selection Checkbox Where Some Rows Are Selected



Note: Each browser natively renders the partially-selected state of the tri-state checkbox so it's appearance will vary depending on the browser you use.

Accessibility Considerations

Release 12.2.6 supports the single selection radio button and the tri-state multiple selection checkbox in screen readers. The radio button or checkbox can be selected/unselected using the keyboard space bar. However, since the cell above the single selection column is empty and the cell above the multiple selection column displays the tri-state checkbox, there is no visual text to indicate the column scope for the Select columns.

Declarative Implementation

Step 1: To enable table selection, select your table region in the Structure pane of OA Extension. Display the context menu and under **New**, choose either the **singleSelection** or **multipleSelection** named child. If you choose singleSelection, OA Framework renders a Selection column with radio buttons, so you can select a single row in the table. If you choose multipleSelection, OA Framework renders a Selection column with checkboxes, so you can select multiple rows in the table.

Attention: The Selection column is always rendered as the first column in a table in accordance with the Oracle Browser Look-and-Feel (BLAF) Guideline: Tables. Conversely, if the first column in a table is a checkbox or radio button item, the item will behave as a Selection checkbox or radio button, respectively. OA Framework does not support any other additional checkbox or radio button item in the table.

Step 2: Using the Property Inspector, set the following properties on the singleSelection or multipleSelection named child:

Properties	Description
`ID`	Specify an identifier that uniquely identifies the item. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
`View Instance`	Specify a view object instance for this selection child.
`View Attribute`	Specify a view attribute for this selection named child. **Note:** The view attribute must be a String, such as "Y" or "N".

Step 3: (Optional) Add selection buttons to the control bar. In the Structure pane, select the selection named child that was created in the previous step. Display the context menu and choose **New > selectionButton**. This is a special type of submit button that is added as an indexed child of the selection bean. Once UIX finds that the selection bean has one or more indexed children, it renders a control bar in the table and puts the selection button(s) there.

Note: The **Select All** and **Select None** links in the table filter area appear when the table selection named child is multipleSelection.

Runtime Control

OA Framework applies the check/unchecked values of the table selection to the corresponding view attributes. Use your controller to identify which records are selected and then take programmatic action from there. A code example of how to locate rows with certain attribute values is available in the Iterating View Object Rows section of Chapter 5: View Objects in Detail.

Disabling One or More Rows of Single or Multiple Selection

To disable one or more rows of single or multiple selection in your table, add the following code to your controller `processRequest` method:

```
OATableBean tableBean = ...;
tableBean.setSelectionDisabledBindingAttr("Disabled");
```

In the example above, "Disabled" represents the Boolean view attribute that returns 1/"Y" if one or more rows is (are) to be disabled, otherwise it returns 0/"N". The view attribute "Disabled" *must* belong to the same view object as the one associated with the table items.

Setting the Table Selection Text

You can programmatically set the text in your table selection bar by calling the following in your controller:

```
tableBean.setTableSelectionText("<newText>");
```

This should precede any call to `prepareForRendering` in the controller.

Modifying the Control Bar

The following code illustrates how you can suppress the control bar:

```
processRequest
{
  ...

  // NOTE: this must be called prior to prepareForRendering()
  tableBean.setControlBarDisplayed(false);
  ...
}
```

The following controller code example illustrates how you can add an action poplist to the table selection control bar:

```
// first prepare the Table Bean for rendering
tableBean.prepareForRendering(pageContext);

// get a handle on the Table Selection Bean
OAWebBeanTable tableSelectionBean =
  (OAWebBeanTable)tableBean.getTableSelection();

// dynamically create your PopList and Button etc
OAMessageChoiceBean myPoplist =
  (OAWebBeanChoice)createWebBean(pageContext, MESSAGE_CHOICE_BEAN);

myPoplist.setListViewObject(...);
myPoplist.setListDisplayAttribute(...);
myPoplist.setListValueAttribute(...);

// The following is required according to the accessibility standards
// to set the alt text value for the poplist
myPoplist.setShortDesc(...);

OASubmitButtonBean myButton =
  (OASubmitButtonBean)createWebBean(pageContext, BUTTON_SUBMIT_BEAN);

myButton.setText(...);

// add the PopList, Button etc to the Table Selection Bean
tableSelectionBean.addIndexedChild(myPoplist);
tableSelectionBean.addIndexedChild(goButton);
```

Beginning in Release 12.2.6, OA Framework maintains, in both the server and client side, the user interaction states of the column header tri-state checkbox as well as the individual table row Select checkboxes.

State Transition of the Column Header Tri-state Checkbox

Present State	Event Trigger	New State
SELECTED	click (mouse) space bar (keyboard) tap (touch device)	UN_SELECTED
UN_SELECTED	click (mouse) space bar (keyboard) tap (touch device)	SELECTED
PARTIALLY_SELECTED	click (mouse) space bar (keyboard) tap (touch device)	UN_SELECTED

State Transition of Table Row Checkboxes

- SELECTED - Set when all rows of the table are selected excluding disabled rows.
- UN_SELECTED - Set when all rows of the table are not selected excluding disabled rows.
- PARTIALLY_SELECTED - Set when some rows of the table are selected excluding disabled rows.

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- In accordance with the BLAF guidelines, the selection bean (column) is always rendered with a default column header text of **Select** for the single or multiple selection. This default column header text cannot be changed. Also, the **Select All** and **Select None** links in the table filter area that appear when the table selection named child is multipleSelection cannot be changed.
- A table can only have either a single selection (radio group) or a multiple selection (check box group), but never both.
- Further, if a table has either a single selection or multiple selection, then no other column in the table can have a radio button or a checkbox item.
- A table can contain a radio button or checkbox only in the first column, and not in any other column.

Table Actions

Table actions are global table-level action components that are rendered in the control bar of the table. Previously, you had to define tableLayout-rowLayout-cellFormat web beans above the table to include global action items for the table. Now, using the UIX tableActions named child, you can define any number of items under a flowLayout or rowLayout. Generally, submitButtons and on rare occasions, poplists, are defined in this area.

The tableAction component is also supported in HGrid and Gantt regions.

Declarative Implementation

Step 1: To define table actions, select your table region in the Structure pane of OA Extension. Display the context menu and choose **New > tableActions**. This automatically creates a tableActions named child consisting of a flowLayout region.

Step 2: Specify a standards-compliant ID for the region and leave the `Region Style` as **flowLayout** or set it to **rowLayout**.

Suggestion: If you have only buttons to add to the table actions area, use either layout styles, flowLayout being preferable. However, if you are adding message web beans such as messageChoice or messageTextInput, along with buttons to the table action area, then you should use the rowLayout style. Using a flowLayout instead of a rowLayout in this case may cause alignment problems.

Step 3: Under the Layout region, layout the children you want to render as table actions, such submitButton or messageChoice. Select the Layout region, and choose **New > Item** from the context menu. Select the new Item that is created and set the item style as appropriate.

Runtime Control

There is no runtime control code necessary to implement table actions.

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Sorting

A table column can be sorted by a user at any time, and can also be automatically sorted when the page is initially rendered, (known as initial sorting). In the case of initial sorting, you can sort your table based on one, two or three columns of data. When a column is sortable, the column heading appears beveled, but does not display the sort direction arrow icon until the user selects the column heading to sort the column.

If a user selects a beveled column heading to sort, the sort direction is ascending and displayed as such through the sort direction arrow that appears. If the user selects the sort direction arrow again, it toggles to the opposite direction.

If a table is initially sorted, the column set as the first level of sorting displays the sort direction arrow icon, that is, until the user sorts another column, at which point the sort direction arrow icon appears in that column header.

Sorting is case sensitive, where all uppercase values precede lower case values.

Declarative Implementation

To enable sorting on a table column, you need to set the `Sort Allowed` and `Initial Sort Sequence` properties on the item (column) of interest. The following list identifies the possible combination of values you can set on these two properties and how they affect sorting:

Sort Allowed	Initial Sort Sequence	Result
no	none	Column is not sortable and is not initially sorted when the table is rendered.
yes	first,second, or third	The column is sortable and is also initially sorted as either the first, second, or third level of sorting performed on the table, in ascending direction (default), when the table is rendered.
yes	none	The column is available for sorting by the user, but is not initially sorted when the table is rendered.
ascending or descending	none	The column is available for sorting by the user, but is not initially sorted when the table is rendered. This usage should not be used. Instead, set Sort Allowed to yes, and Initial Sort Sequence to none to achieve the same result.
ascending or descending	first, second, or third	The column is sortable by user in the specified direction and is also initially sorted as either the first, second, or third level of sorting performed on the table.

Enabling Internal Sorting On Another View Attribute

Generally, you can specify sorting on a table when you define the table. In some cases, especially when you use a raw HTML column with a HTML link, you may want the sorting to occur on the link value displayed in the user interface rather than on the actual link value itself (with <a href ...). To accomplish this, set the `Sort By View Attribute` property on the region item, to the name of the view object attribute on which you want to sort.

If this property is not set or set to null, sorting is performed on the view attribute name associated with the table column.

Attention: If a table column contains a nested region, such as a switcher region, and you want to enable sorting on this region, it is mandatory that you set the Sort By View Attribute property on the region.

Runtime Control

The table bean invokes the following two methods on the view object to perform sorting:

- `viewObject.setOrderByClause(orderByClause);`
- `viewObject.executeQuery();`

Debugging can be performed by setting a break on these methods on the view object (or a superclass of the view object).

You can programmatically make a column sortable by using the `isSortable` and `setSortable` methods on `oracle.apps.fnd.framework.webui.beans.OAWebBeanDataAttribute`. Refer to the following code as an example:

```
// We assume that all children of the table are those that implement
// OAWebBeanDataAttribute
// Alternately, the exact bean type can be used, such
// as OAMessageStyledTextBean.
OAWebBeanDataAttribute webBean = (OAWebBeanDataAttribute)
  tableBean.findIndexedChildRecursive("<columnName>");
webBean.setSortable(true);
```

Enabling Internal Sorting On Another View Attribute

You can programmatically enable internal sorting on another view attribute.

For example, suppose you have a view attribute named Emplink. When you click on the sort icon of the table column for Emplink, you want the actual internal sorting to occur on the Ename attribute. You can programmatically control this as follows:

```
OATableBean table = (OATableBean)createWebBean(pageContext,  webBean, "EmpTable");
OAWebBeanDataAttribute empno =
  (OAWebBeanDataAttribute)table.findChildRecursive("<EmpnoItemID>");
empno.setSortByAttributeName("Ename");
```

If this sort by attribute is not set or set to null, sorting is performed on the view attribute name associated with the table column.

Sort with Manually Populated Transient Attributes

When you try to sort a column in a table that has one or more manually populated transient columns, the transient values are lost upon sorting. This is because the query is executed by OA Framework. In order to avoid this, use the following solution:

1. Encapsulate the logic to populate transient columns (view attributes) in a method.
2. On a sort event, redirect back to the same page in `processFormRequest`.
3. In `processRequest`, handle the redirect, by calling the tableBean `prepareForRendering` followed by a call to the method that populates transient columns.

Implementing Table Selection and Hide/Show

By default, when the selector item is checked in a table that supports table selection, the underlying view object attribute value is updated. This action leaves the view object with pending changes (even for a view-only table). When the view object is in this "dirty" state, operations such as table sorting or personalization are disabled. To implement a selector item without affecting the view object state, follow these steps:

Step 1: Define a transient view attribute for the selector item using the BC4J view object wizard in JDeveloper. Do *not* add a dynamic attribute by calling `ViewObject.addDynamicAttribute`.

Step 2: Following the OA Framework coding standards, always generate an `OAViewRowImpl` for each view object so you can call named accessors. In this case, override your `set<AttributeName>()` method as shown below:

```
public void setSelectFlag(String val)
{
  populateAttribute(SELECTFLAG, val);
}
```

This example assumes the selector's view attribute is named "SelectFlag," and the code is calling the `populateAttribute()` method to set its value without marking the view object as being "dirty."

A similar situation occurs when Hide/Show is present in a table. See Entity Object and View Object Setters in Chapter 5 for additional information.

Personalization Considerations

If you expect your users to personalize the sort order of a table, do not set explicit order by statements in your view objects. Instead, use the `Initial Sort Sequence` and `Sort Allowed` properties in OA Extension. When the table is rendered the first time, OA Framework provides an order by statement on the view object based on how you set these properties. For more information on how to personalize the sorting of data in a table, refer to Admin-Level Personalizations in the Personalization Guide.

If you define an end-user personalizable page, where the classic table under the query region, contains a nested region column (such as a table content switcher), users will be able to sort on that column in the Create/Update/Duplicate View page. Previously, users were not able to set a nested region column as a sort column in the **Sort Settings** because that column was not exposed in the **Column Name** poplist.

- When a user specifies the nested region column as the column to sort, the actual content that is sorted will depend on what you set the Sort By View Attribute property to for the column.
- If a user sets sorting on a nested region column using the Create View page, but finds that the change is not taking effect on the table, it is likely that the `Sort By View Attribute` property on the nested region column was not set. In this case, the personalization administrator would have to set the `Sort By View Attribute` property on that column using the Admin Personalization UI.
- If you do not want sorting on the nested region to be modified by users, you can set the `User Personalization` property to **False** so that the nested region does not display in the **Column Name** poplist of the Create View page's **Sort Settings**.

See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

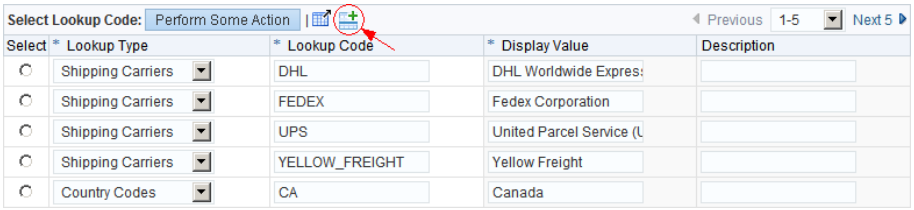
- Performed by re-querying the database.
- May *not* work well with view objects that contain custom 'expert-mode' SQL. Basically the view object `setOrderByClause` is invoked on the view object using the column name associated with this web bean. An alternative for 'expert-mode' SQL may involve overriding the view object `setOrderByClause` and performing custom logic.

  Note: The `orderByParameter` contains the column name and either **desc** or **asc**.
- Does *not* work with view objects that contain the view attribute expression "See the SQL...". To sort on these view attributes, modify the view object XML directly and change the view attribute expression to the SQL column name.
- Not allowed for tables that allow inserts.
- Not supported for tables containing updateable columns (unless the updateable columns are mapped to transient view object columns). No exception is thrown if the table contains updateable columns, since there may be a rare case when it makes sense, such as in a table where the contents fit on one page.
- Modified transient columns are reset. This is normal and expected behavior.
- Not supported on the Select column, the optional first column of a table that contains a checkbox or radio button.
- Disabled when the Select column is checked for a row in a table. See Implementing Table Selection and Hide/Show for more information.
- Not supported on table rows created in the middle-tier.
- Sorting is allowed on a composite column whose leaf items are grouped under a classic table-supported region layout style such as a message component layout, flow layout or hide/show. However, if the item is added under multiple hierarchy levels of supported layout styles, then the sort property must be set on the top-most region layout.

Adding Rows

OA Framework displays an **Add Another Row** button to the table footer if a table is insertable (for Release 12.1.3 and earlier).

Note: As of Release 12.2.3, instead of an **Add Another Row** button appearing at the table footer, an **Add Another Row** icon now appears on the table's control bar if you set the **Oracle Applications Look and Feel / APPS_LOOK_AND_FEEL** profile to the Release 12.2.3 Look and Feel, as shown in the figure below. The button label that you can customize for the previous Look and Feel now appears as a tool tip for the icon.



Note: Also in Release 12.2.3, if you implement the table to add N rows, an **Add Multiple Rows** icon appears in place of the **Add Another Row** icon in the control bar.

When you add another row, the row is added as the last row in the current range. The existing last row in the current range is pushed into the next range.

Note: When you select the **Add Another Row** button/icon, the cursor is placed on the first updateable field of the new row that was added. This type of functionality is *not* supported for the **Add *n* Rows** button/icon.

You can also add new rows to a detail table that is associated to some master row via a detail view instance. In the past, you had to `setAutoInsertion(false)` on such a table, and handle row insertions yourself. Now, you can add rows automatically to detail tables because foreign keys are automatically populated for a table associated with a detail view instance of some master row. Of course, you still have to `setAutoInsertion(false)` if there is custom SQL in the view link, or if you have to perform some custom population of detail row data.

Declarative Implementation

There is currently no declarative implementation available for adding another row to a table.

To allow new rows to be added to a detail table (as in the case of a Table-in-Table, Detail Disclosure drilldown, or a master-detail template), then in addition to specifying `setAutoInsertion(true)` in your controller code, you must also attach the master and detail tables together by specifying an identical value in the `View Link Name` property of both the master and detail tables.

Note: Auto-insertion in detail tables only work for cases where the view link has no custom SQL.

Runtime Control

OA Framework renders an optional **Add Another Row** button (or icon, as of Release 12.2.3), if the table is programmatically marked as insertable by calling `setInsertable(true)` in your controller before calling the `prepareForRendering` method. When a user selects the **Add Another Row** button, OA Framework calls `processFormRequest` and adds a new row to the view instance associated with the table, at the bottom of the visible range.

You can set the OATableUtils.NEW_ROW_INITIALIZED property on the OATableBean in your controller to ensure that the state of the new rows added by OA Framework is STATUS_INITIALIZED. This ensures that rows not changed by the user are not posted to the database on a commit.

```
  //Set the NEW_ROW_INITIALIZED property
  tableBean.setAttributeValue(OATableUtils.NEW_ROW_INITIALIZED, Boolean.TRUE);
```

If the table region is programmatically marked as insertable, OA Framework creates an `OAAddTableRowBean` and designates it as the `columnFooter` named child of the `OATableBean`. If the `OATableBean` is also configured to total column values, then the total bean becomes an indexed child of the `OAAddTableRowBean`. See the Javadoc for `oracle.apps.fnd.framework.webui.beans.table.OATotalRowBean`.

You can make various modifications to the **Add Another Row** button/icon.

Example 1

Change the label on the **Add Another Row** button by getting a handle to the button and modifying its prompt.

```
processRequest(...)
{
  ...
  // Prepare the table before accessing column footers
  tableBean.prepareForRendering(pageContext);

  // Get a handle to "Add Another Row" button
  OAAddTableRowBean addRowBean = (OAAddTableRowBean)tableBean.getColumnFooter();

  if (addRowBean != null)
  {
    addRowBean.setText("<newCustomText>");
  }
  ...
}
```

Example 2

The other change you can make is to suppress the default **Add Another Row** button behavior of adding a single row to the view instance associated with the table, so that you can handle the event with your own code.

The following code example illustrates how to accomplish this:

```
processRequest
{
  ...
  // Enabled add row and turn off the default "Add Another Row" table event

  // The add row event has to be auto-handled by developer in processFormRequest
  tableBean.setInsertable(true);
  tableBean.setAutoInsertion(false);
  ...
}

processFormRequest
{
  ...
  if ((tableBean.getName()Equals(pageContext.getParameter(SOURCE_PARAM))) &&
    (ADD_ROWS_EVENT.equals(pageContext.getParameter(EVENT_PARAM))))
  {
    OAApplicationModule am = pageContext.getApplicationModule(tableBean);
    am.invokeMethod("handleInsertRow", null);
```

```
  }
  ...
}

// The ***AMImpl.java in which method "handlInsertRow" has been defined
public void handleInsertRow()
{
  OAViewObject vo = findViewObject("voName");
  vo.invokeMethod("handleInsertRow");
}

// The ***VOImpl.java which is associated with the table; and in which the
// handleInsertRow is defined
public void handleInsertRow()
{
  Row row = createRow();

  // Set any default attributes
  row.setAttribute(...);
  ...
  // Insert the row into the VO
  insertRow(row);
}
```

Example 3

The following code example shows how to change the default behavior of the **Add Another Row** button, so that it instead displays **Add 5 Rows** (or, in the case of Release 12.2.3 and above, displays an Add Multiple Rows icon) and adds five new rows instead of one:

```
processRequest
{
  ...
  tableBean.setInsertable(true);
  tableBean.setAutoInsertion(false);

  tableBean.prepareForRendering(pageContext);

  OAAddTableRowBean addRow = (OAAddTableRowBean)tableBean.getColumnFooter();

  // In general, rather than use hard-coded strings, use messages from
  // the message dictionary to set UI text.
  addRow.setText("Add 5 Rows");
  // As of Release 12.2.3, use the following API to specify the number of
  // rows to add.  In this example, it is set to 5.  If you set this to > 1, the
  // "Add Multiple Rows" icon renders, otherwise the "Add Another Row" icon renders.
  addRow.setRows(5);
  ...
}

processFormRequest
{
  ...
  if ((table.Bean.getName().equals(source)) && (ADD_ROWS_EVENT.equals(event)))
  {
    // invoke method on view instance that will add 5 rows
    ...
  }
}
```

Note: If you add multiple rows to a table, the new rows are shown at the bottom of the current range, pushing any existing rows into the top of the subsequent range. Only add, at most, a number of rows that is less than or equal to the number of rows displayed in your table.

Add Another Row and View Object Execution

If the view object associated with the table is *not* executed and you select **Add Another Row**, you get a state loss error on the page. For any successful table event, the table view object query *must* be executed before a table event occurs. In the case where you do not want to display any rows when the page renders, yet want to let the user select **Add Another Row** to add rows into the table, then before you add rows to the table, you must properly initialize the view object as described in the Initialization Guidelines.

Empty Rows In A Table

Multiple empty rows can be added into a table when any of the following occur:

- Your code inserts rows into the view object before the table renders, so the table displays new blank rows.
- A user selects the **Add Another Row** button multiple times.
- A user selects the **Add n Rows** button, which you have programmatically implemented to add 'n' rows into your view object.

In any of these cases, if the user does not enter values into some of the rows, you *must* include logic to recognize those empty rows (if any), and remove them from the view object. For example, if you prepopulate ten new empty rows in an expense reports table, but the user enters only three expense lines, you *must* provide code to recognize and remove the remaining seven empty rows from the view object.

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

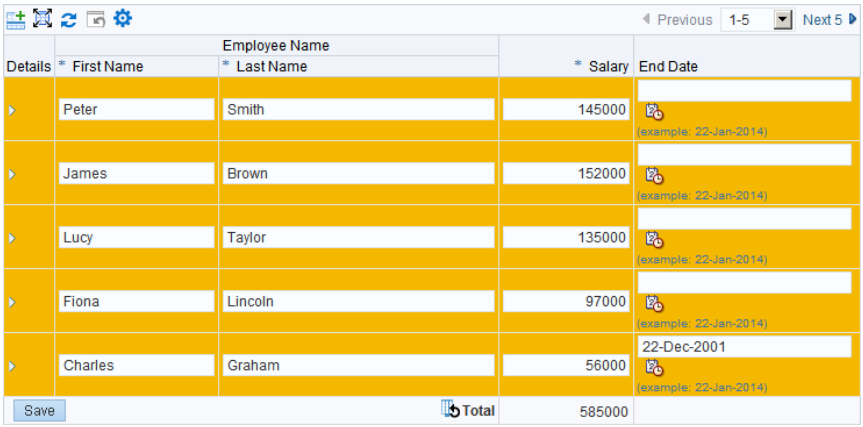
Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Totaling

OA Framework supports the Totaling of numeric values in a table column. If Totaling is enabled in a table, the table footer displays a column total and a **Recalculate** button, as shown below. The total displays a double precision summation of all visible rows in the table. The total reflects only the current visible records and not all the records queried and that the content of the totaled column is automatically right aligned.

Note: As part of the Release 12.2.4 Look and Feel, the **Recalculate** button appears as an icon next to the **Total** text. The button label that you can customize for the Swan Look and Feel now appears as a tool tip for the icon.



Declarative Implementation

You can tabulate a total for a table column if the column data type is a number and it is *not* the first column in the table. To create a total for a column, set the `Total Value` property in OA Extension to **True** for the corresponding region item.

If the region item associated with the totaled column is updateable, that is, **Read Only** is **False**, a **Recalculate** button also appears to the left of the Total. OA Framework can support totals for one or more columns.

Runtime Control

If the column data displays currency, you can programmatically set the currency code to tabulate a currency total. Use the `setCurrencyCode(String)` method on `OAWebBeanDataAttribute` to do this. The `oracle.apps.fnd.framework.webui.OAWebBeanConstants` public constant `CURRENCY_CODE`, which is defined on the web beans, formats the contents of the Total cell according to the precision and format of the specified currency.

The `OATotalRowBean` implements totals in the column footer. If the `Total Value` property in OA Extension is set on the table region, OA Framework creates an `OATotalRowBean` and specifies it as the `columnFooter` named child of the `OATableBean`. If `OATableBean` is also configured to insert rows (so it has an Add Another Row button), then `OATotalRowBean` becomes an indexed child of the add table row bean. See `OAAddTableRowBean`, which OA Framework in turn designates as the table's columnFooter object.

Controlling Tabular Values

To implement a total on a table column programmatically, call the method `setTabularFunctionCode(TABULAR_FUNCTION_SUM)` on the web bean. This method places the total value column footer.

To bypass OA Framework's mechanism to control the value generation for a table column footer, such as the total, so you can provide your own value, (such as a percentage), set the `OAWebBeanConstants TABULAR_FUNCTION_VALUE_ATTR` on the appropriate table column. This overrides OA Framework's internal behavior of footer value generation. Like all web bean attributes, you can data bound this attribute if required.

Sample usage:

```
OATableBean tableBean = ...;
OAMessageStyledTextBean salaryBean = tableBean.findChildRecursive(...);

String formattedTotal = ...;
// compute the total
salaryBean.setAttributeValue(TABULAR_FUNCTION_VALUE_ATTR,formattedTotal);
```

or:

```
salaryBean.setAttributeValue(TABULAR_FUNCTION_VALUE_ATTR,
  new OADataBoundValueViewObject(...));
```

Updating the Recalculate Button Label

To change the prompt for the **Recalculate** button when there is no **Add Another Row** button present in the column footer :

```
processRequest(...)
{
  ...
  // Prepare the table before accessing column footers
  tableBean.prepareForRendering(pageContext);

  // Get a handle to total row bean. NOTE: This code sample assumes that
  // there is NO "Add Another Row" button. If present, see next example.

  OATotalRowBean totalRowBean = (OATotalRowBean)tableBean.getColumnFooter();
  if (totalRowBean != null)
  {
    totalRowBean.setText("<newCustomText>");
  }
  ...
}
```

To change the prompt for the **Recalculate** button when there is an **Add Another Row** button present, use the following code example, which gets a handle for both the add another row and total row beans:

```
processRequest(...)
{
  ...
  // Prepare the table before accessing column footers
  tableBean.prepareForRendering(pageContext);

  // The column footer will be the "Add Another Row" button
  OAAddTableRowBean addRowBean = (OAAddTableRowBean)tableBean.getColumnFooter();
  if (addRowBean != null)
```

```
  {
    addRowBean.setText("<newCustomText>");

    // The total row bean will be an indexed child of the add row bean
    int childCount = addRowBean.getIndexedChildCount();

    for (int i = 0; i < childCount; i++)
    {
      OAWebBean webBean = (OAWebBean)addRowBean.getIndexedChild(i);
      if (webBean instanceof OATotalRowBean)
      {
        OATotalRowBean totalRowBean = (OATotalRowBean)webBean;
        totalRowBean.setText("<newCustomText>");
        break;
      }
    }
  }
  ...
}
```

Personalization Considerations

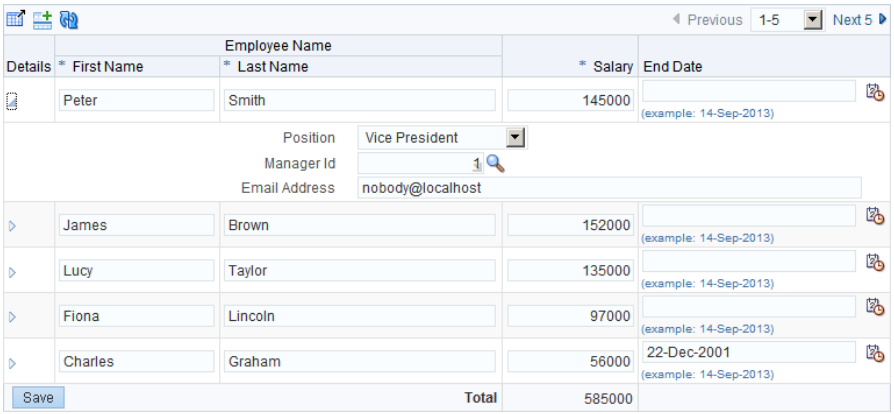
- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Totaling can be enabled for any column except for the first column in a table.
- You cannot total columns in the inner table of a Table-in-Table.
- Totaling is supported *only* on items immediately under a classic table. It is *not* supported on regions, such as a messageComponentLayout region, under a classic table.

Detail Disclosure

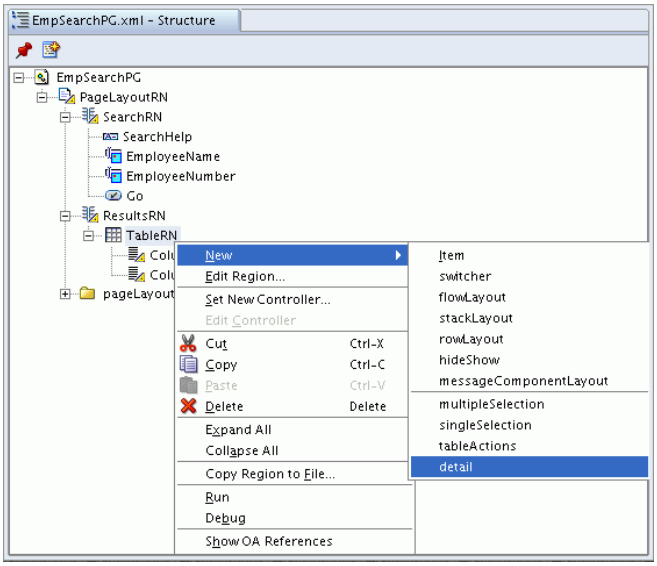
Detail disclosure is also known as a row-level Hide/Show in a table, as described in the BLAF Guidelines for Tables.

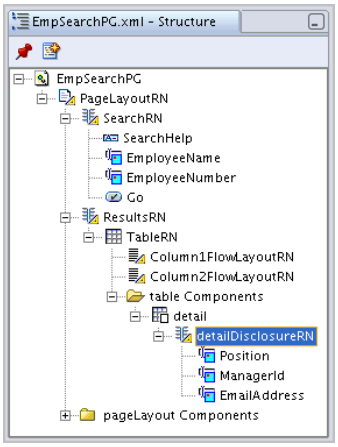


Declarative Implementation

Define the detail disclosure in a table as follows:

Step 1: Select the table region in the Structure pane of OA Extension. Under **New** in the context menu, select **Detail**. This creates a detail named child that you can use to show additional information for any row in the table. The recommended UI layout for the region is labeledFieldLayout. The default region style of the detail region is **header**, but you can change the style, if required, and add children to the region.





Step 2: For the table region, set the `Detail View Attribute` property to the name of the Boolean/String ("Y" or "N") attribute (of the same view object as that of the other table columns) that determines the shown or hidden state of the detail child of the rows.

Note: Specifying the named child detail region alone is sufficient to enable the addition of a row-level Hide/Show. You do not need to explicitly add a hideShow as the detail region.

Attention: The leaf items in the detail disclosure named child region *must* be associated with the same view object as the leaf items in the other columns of the table. The only exception to this is when your detail disclosure named child region contains another table (Table-in-Table), in which case the outer and inner tables *must* be associated with master and detail view objects connected via a view link.

Runtime Control

You can set the properties discussed in the Declarative Implementation section above in your controller:

```
import oracle.apps.fnd.framework.webui.beans.table.OATableBean;
...
OATableBean tableBean =
  (OATableBean)webBean.findIndexedChildRecursive("<tableBeanName>");

// Set the detail named child
OAWebBean detailNode = (OAWebBean)...;

// Get a handle to the detail node
tableBean.setDetail(detailNode);

// Set the detail view attribute name
tableBean.setDetailViewAttributeName("<detailViewAttributeName>");

// Turn on the "Hide All Details | Show All Details" links
tableBean.setAllDetailsEnabled(true);

...
```

Refer to Table-in-Table for information on nested tables or table inside detail disclosure.

Personalization Considerations

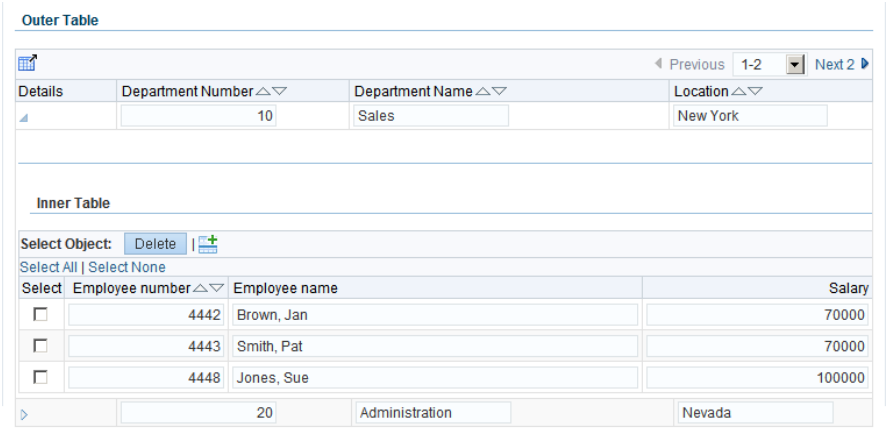
- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Do *not* create a hideShow or a hideShowHeader region under the detail region of a table because the BLAF Hide/Show guidelines for Tables does not allow embedding secondary or children objects in the details of a table row. See HGrids for information on how to display hierarchical information in a table. Similarly, do *not* create a hideShow region under an inner table of a Table-in-Table or implement detail disclosure inside the inner table of a Table-in-Table.
- In accordance with BLAF guidelines, the **Detail** column header text for the detail disclosure column and the **Show/Hide** link texts within the cells of the detail disclosure column can not be modified.
- The `All Details Enabled` property for the table renders the **Hide All Details|Show All Detail** link above the table. Selecting **Show All Details** expands all the detail disclosure regions, and selecting **Hide All Details** collapses all the detail disclosure regions. However, Performance team guidelines strongly discourage the use of this feature. As a result, you can not set this property declaratively in OA Extension for Table regions. If you want to enable this property, you *must* do so using the `setAllDetailsEnabled` method. If you need to use this feature, please review your functionality and UI design with the performance team first. Refer to the Runtime Control section for an example.
- The **Hide All Details|Show All Detail** link text cannot be changed.

Table-in-Table

If you have number of records that are inter-related, you can display them in tabular form using a table bean. This is an intuitive way to display data that is independent of other entities. Table-in-Table allows all inter-related information, typically master-detail relationships, to be seen and updated on one page by embedding an inner table within an outer table. This is possible because you can add a table bean as a named child of another table. The information in the inner table is based on the corresponding master row in the outer table. The inner table associated with each row is opened/closed using the **Hide/Show** links that appear in the Details column as shown in the sample table-in-table user interface below.



The inner table bears a master-detail relation with the outer table. For each master row, corresponding detail rows are displayed in an inner table. Users can control the rows in the inner table by defining additional search criteria to the criteria already defined in the view link, which relates the outer view object with the inner view object.

Note: A table-in-table user interface can be costly in terms of performance because multiple view objects and the bean hierarchy are created and replicated at runtime. Performance cost is reduced when the inner table is read-only because the same view object can be reused.

The creation of the view object attached to a detailed section is done during rendering time. If you never open the detailed section of a row, the view object is not created. So if you have ten master rows in the outer table and you open the detail section of only two rows, the performance cost is only for those two rows.

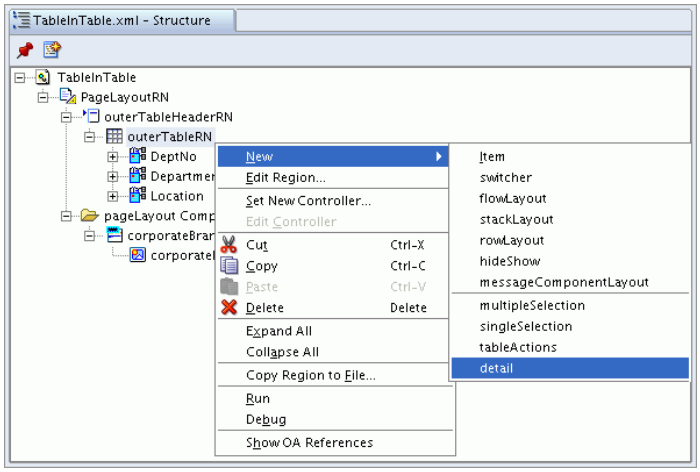
You can make the inner table editable by adding form-elements in the inner table so the data is pushed to the view object just like a regular table. The inner table also supports navigation, sorting, adding another row, single and multiple selection, row-level and column-level banding and exporting to a Microsoft Excel spreadsheet.

Declarative Implementation

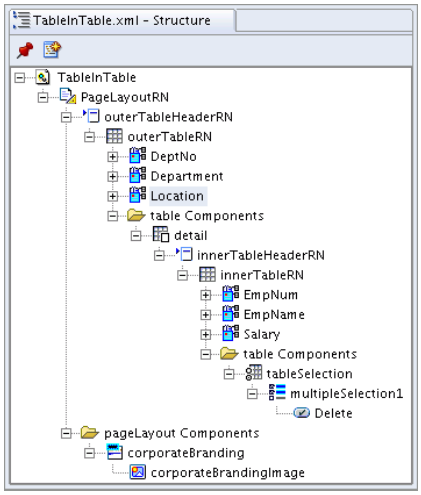
To create a table like the one shown above, you *must* define the region definition in xml using OA Extension, and add code in your controller.

Xml Structure

Step 1: Select the (outer) table region in the Structure pane of OA Extension. Under **New** in the context menu, select **Detail**. This creates a detail named child that you can use to show additional information for any row in the table. The default style of the detail region created is **header**. The detail region becomes the header for your inner table.



Step 2: Under the inner table header region, create a region with a style of **table**, to become your inner table.



Step 3: Follow the standard instructions for defining a table to create your inner table.

Step 4: For the outer table region, set the `Detail View Attribute` property to the name of the Boolean/String ("Y" or "N") attribute (of the same view object as that of the other table columns) that determines the shown or hidden state of the detail child of each row.

Runtime Control

After creating the XML page structure, you need to initialize some properties on the beans in your code.

As a example, suppose your BC4J setup is as follows:

- View object 1 = DeptVO.
- Primary Key of DeptVO = Deptno.
- View object 2 = EmpVO.
- DeptVO and EmpVO are connected by a view link called DeptEmpVL.
- DeptVO and EmpVO are connected by the query *WHERE DEPTNO = :1*.
- Application module contains the two view objects and the view link.

Note: The view link is used in the master-detail relationship to display the correct detail rowset.

To achieve the table-in-table display, add these lines in your controller:

```
public void processRequest(...)
{
  OAWebBean outerTable = (OAWebBean)webBean.findChildRecursive("outerTable");
  OAWebBean innerTable = (OAWebBean)webBean.findChildRecursive("innerTable");
  if (outerTable != null)
  {
    outerTable.setAttributeValue(CHILD_VIEW_ATTRIBUTE_NAME,"Deptno");
    outerTable.setAttributeValue(VIEW_LINK_NAME,"DeptEmpVL");
  }
  if (innerTable != null)
  {
    innerTable.setAttributeValue(CHILD_VIEW_ATTRIBUTE_NAME,"Deptno");
    innerTable.setAttributeValue(VIEW_LINK_NAME,"DeptEmpVL");
  }
  ...
}
```

Warning: The two properties that are set in the example above are required for the proper functioning of the bean ID generation and master-detail relationship. If you omit any of the above, you may end up with a JavaScript error or see incorrect records in the inner table.

RowSets

The table-in-table detail template (inner table) uses the RowSets interface to interact with the OA Framework Model. When you query data using a view object, the results of the query are stored in a RowSet. Each RowSet can have multiple iterators which allows scrolling through the set of rows and each view object can have multiple RowSets.

As a result of the RowSet implementation, to handle rows manually in a table-in-table, you need to go about it differently than when you handle rows in a regular table. Generally, with a table, you can get a handle to the table view object and manipulate it, such as by adding or deleting a row, and seeing the change reflected on the UI. This is not the case for the inner table of a table-in-table. Any changes that you make to the detail view object is never even considered.

To insert a new row or update the state of an existing row in an inner table, you *must* create an `oracle.apps.fnd.framework.webui.OAInnerDataObjectEnumerator` and use it to get a handle to the RowSet attached to the inner table. Once you have the RowSet, you can manipulate it in any manner and the changes are reflected in the UI. In previous versions of OA Framework, `OAInnerDataObjectEnumerator` used to scroll through the different view objects associated with each inner table. Now since a RowSet is associated with each inner table, the behavior of this enumerator has been updated. Following is an example of how to use the enumerator to go through the rows of a RowSet attached to an inner table:

```
// get a handle to inner table
OATableBean innerTable =
  (OATableBean)webBean.findChildRecursive("InnerTableBean");

// create an enumerator
OAInnerDataObjectEnumerator enum =
  new OAInnerDataObjectEnumerator(pageContext, innerTable);

while (enum.hasMoreElements())
{
  RowSet innerRowSet = (RowSet) enum.nextElement();

  // get all rows
  Row []rowsInRange = innerRowSet.getAllRowsInRange();

  for (int i = 0; i < rowsInRange.length; i++)
  {
    Row nextRow = (Row) rowsInRange[i];

    if ("Y".equals(nextRow.getAttribute("DeleteFlag")))
    {
      // delete the marked row
      nextRow.remove();
    }
  }

  // In case you want to add new rows in this RowSet, you can do the same
  OARow newRow = (OARow) innerRowSet.createRow();

  // initialize value for some attribute and insert the row
  newRow.setAttribute("SomeAttr", "SomeValue");
  innerRowSet.insertRow(newRow);

}

// In case you want to change the WhereClause of the containing view object
OAViewObject innerViewObject = (OAViewObject) innerRowSet.getViewObject();

String newWhereClause = "DEPT.LOC = :1";
innerViewObject.setWhereClause(newWhereClause);

// Each RowSet can now bind parameter specific to each inner web bean
```

The association of a RowSet instead of a view object to the inner table improves the performance of a table-in-table. The amount of memory and other JDBC resources required is limited because all RowSets of an inner table are part of the same view object, and therefore share the same query. Also, the implementation of a table-in-table is now consistent with other web beans of similar hierarchical nature. The master table is always in synchronization with the inner table because any change in the master table triggers changes in the inner table via a view link.

Note: If you use the `OAInnerDataObjectEnumerator` in the `processRequest` method, you may not be able to scroll through the RowSets because they may not yet be created. A RowSet for an inner table is created *only* when a user actually displays the inner table in the UI.

If you want to iterate through all fetched inner rows instead of the rows in the current range, create OAInnerDataObjectEnumerator in the following way:

```
OAInnerDataObjectEnumerator enum =
      new OAInnerDataObjectEnumerator(pageContext, innerTable, true);
```

The third argument indicates that the enumerator should go through all fetched rows. The default behavior is to fetch the rows in the current range.

Note: If a transient VO is used in a table in table scenario and the inner table bean's cache is cleared, then OAF does an execute of detail VO to obtain the detail row sets. The resulting executeQuery clears the transient VO. If you want to prevent an execution of the detail VO and subsequent clearing of the transient VO ,the property if "HOLD_DETAIL_EXECUTE" should be set to `true` on the VO. Then the detail VO will not be executed if the beans' cache is cleared. Looking at the VO.xml, you should see the following property

```
<Properties>
<Property Name ="HOLD_DETAIL_EXECUTE" Value ="true" />
</Properties>
```

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- The inner table does *not* support Totaling. The Totaling feature in the inner table is part of the advanced tables infrastructure only.
- Do *not* create a hideShow or hideShowHeader region under the detail region of a table because the BLAF Hide/Show guidelines for Tables do *not* allow embedding secondary or children objects in the details of a table row. See HGrids for information on how to display hierarchical information in a table. Similarly, you should not create a hideShow region under an inner table of a Table-in-Table or implement detail disclosure inside the inner table of a Table-in-Table.
- The inner table does not support LOVs. You must use an alternative UI.

Formatting a Table

There are several ways to format a table, both declaratively and programmatically. They are as follows:

- Full Table Formatting - affects the table as a whole.
- Column Formatting - affects only the columns of the table.
- Row Formatting - affects only the rows of the table.
- Column Header/Row Header Formatting - affects only the column headers or row headers of the table.

Refer to the section of interest for additional information.

Full Table Formatting

There are two types of formatting that you can apply which affect the table as a whole. The first is banding. In a table, no banding, that is, no alternating colors, is rendered in the columns or rows of a table by default. The figure below illustrates a table with horizontal banding and vertical and horizontal grids (which appear to the left of each column, and above each row).

Column 1	Column 2	Column 3
cell	cell	cell
cell	cell	cell
cell	cell	cell
cell	cell	cell
cell	cell	cell

The second type of formatting is setting the width of the table.

Declarative Implementation

The easiest way to change the appearance of a table is to change its width. This is done by changing the `Width` property on the table region in OA Extension. Although the width can be specified by pixels or percentages of the width of the parent element, percentages are commonly preferred.

Note: No matter what value you specify for the table width, if the width is not large enough to accommodate the table content, the value is overridden at display time to take up the minimum amount of necessary space.

Runtime Control

The UIX `TableBean tableFormat` object determines banding in a table. If a table format `DataObject` is set on the table, it is queried for format information applicable across the entire table rendering. Currently, the table format `DataObject` is queried for the type of banding to render, using the key **tableBanding** (`UIConstants.TABLE_BANDING_KEY`).

If the table format `DataObject` returns the UIConstant, `ROW_BANDING`, the table data rows alternate colors. If it returns the UIConstant, `COLUMN_BANDING`, the table columns alternate colors. Otherwise, no banding is rendered (the default).

If the table format indicates that banding is desired, the table format `DataObject` is also queried with the `BANDING_INTERVAL_KEY`. If this key returns an Integer greater than 1, that interval is used as the number of rows or columns to group in a band. For instance, if a row-banded table returns "2" as the banding interval, the table alternates two dark rows with two light rows over the course of the rendering of the table. The default is the integer 1.

The following controller code example illustrates how you can set row banding or column banding on a table:

```
import oracle.cabo.ui.data.DictionaryData;
...

processRequest(...)
{
  ...
  OATableBean tableBean = ...;
  DictionaryData tableFormat = (DictionaryData)tableBean.getTableFormat();

  if (tableFormat == null)
```

```
  tableFormat = new DictionaryData();

  // Set either column banding (COLUMN_BANDING) or row banding (ROW_BANDING)
  tableFormat.put(TABLE_BANDING_KEY, COLUMN_BANDING);

  // Set the table format
  tableBean.setTableFormat(tableFormat);
  ...
}
```

The UIX `oracle.cabo.ui.beans.table.TableStyle` class also provides a convenient way to set most table, row, or column formatting options. The following example illustrates how you can enable row and column banding on a table using the UIX `TableStyle` class:

```
import oracle.cabo.ui.data.DictionaryData;
import oracle.cabo.ui.beans.table.TableStyle;
...

processRequest(...)
{
  ...
  OATableBean tableBean = ...;

  // Set both column banding (COLUMN_BANDING) and row banding (ROW_BANDING)
  TableStyle tableFormat = new TableStyle(TableStyle.ROW_BANDING | TableStyle.COLUMN_BANDING);

  // Set the table format
  tableBean.setTableFormat(tableFormat);
  ...
}
```

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Column Formatting

You can modify the formatting of a table's columns as follows:

- Alter the alignment of a column's content.
- Turn on/off the line wrap within the cells of a column.
- Alter the width of a column.
- Turn on/off the display of the grid line to the left of a column.

Only one type of column formatting is set automatically, and that is the alignment of a column's content. The alignment of a column's content is based on its content's data type (see special note):

- Start justified - for characters and dates.
- Right justified - previously, all numbers were Right justified. Now, only numbers that are totaled or have currency code set are Right justified. For all other numbers, the alignment is Start justified.
- Center justified - for icon images and button. Applies only to image items and is not suitable for any other components that contain image items.

Declarative Implementation

There is declarative support for the one column formatting option in OA Extension, and that is to enable or disable cell content wrapping for the columns of a table. Set the `No Wrap` property to **False** (default) for an item under a table region to enable the wrapping of a column's cell content. You can also modify this column format programmatically as discussed in the Runtime Control section.

Runtime Control

The `TableBean` columnFormats object determines the column format of a table.

To modify a column's formatting, you can use the `TableBean` `getColumnFormats` method. This method returns a `DataObjectList`. Each `DataObject` in the list corresponds to one column from left to right in the table format. You specify formats inside each DataObject to alter the formatting of each column accordingly. These formatting DataObjects store their values under the following known, publicized keys (UIConstants):

Note: If a DataObject returns a value for a key, the value is used to change the column format, if not, the table assumes the default format.

- `COLUMN_DATA_FORMAT_KEY` (`columnDataFormat`) - Although OA Framework automatically sets column content alignment based on the column content data type, you can programmatically override the default using this key. This property can return any one of three legal values depending on the object in the column and/or its data type. All three values are UIConstants:
  - `TEXT_FORMAT` (`textFormat`)- column content is Start justified.
  - `NUMBER_FORMAT` (`numberFormat`)- column content is Right justified.
  - `ICON_BUTTON_FORMAT` (`iconButtonFormat`)- column content is center justified.
- `CELL_NO_WRAP_FORMAT_KEY` (`cellNoWrapFormat`)- This property determines whether the cell should be rendered with line wrapping disabled. It is often used for LOV or date fields. To override the default, return `Boolean.TRUE` to render the cell in the column without wrapping the contents. The default behavior is to wrap the cell contents because disallowing wrapping can cause tables to become too wide, and is therefore discouraged.
- `WIDTH_KEY` (`width`)- This property returns a string that represents your recommended column width. It can be used to indicate what percentage of extra width in the table this particular column should take. If a table takes up more space than its content needs, the table apportions that space among its columns. However, some columns (like buttons) don't need any more space than the minimum they are given. Other columns, like those containing text, should take extra space if it allows them to reduce the number of lines needed to display their content. The width can be in pixels or percentages to indicate how much of the extra space that column should receive.The total among all the columns should add to 100%. If no column formats in the table request a specific width, the space is divided evenly among the data columns as much as possible.
- `DISPLAY_GRID_KEY` (`displayGrid`)- This property determines whether to display or suppress the vertical grid line to the left of your column. By default, a grid line is shown to the left of every column. But in some cases you may want to use vertical grid lines more sparingly to emphasize the relationship between some columns and de-emphasize it between others. Return Boolean.FALSE to override the default by suppressing the grid lines.
- `BANDING_SHADE_KEY` (`bandingShade`)- This property determines the banding shade of a column, allowing you to control the banding patterns of your columns. When you explicitly band a column, you override the use of banding specified by the table format DataObject. This key can return one of two UIConstants values:
  - `BANDING_SHADE_LIGHT` (`light`)- a light shaded column
  - `BANDING_SHADE_DARK` (`dark`)- a dark shaded column

The following code example illustrates how you can override the default column format by forcing a column's content to not wrap.

```
processRequest
```

```
{
  ...
  tableBean.prepareForRendering(pageContext);

  // now get a handle on the array of Column Format Dictionaries
  DataObjectList myColumnFormats = tableBean.getColumnFormats();

  // get a handle on the specific Column Format Dictionary you
  // are interested in
  oracle.cabo.ui.data.DictionaryData myIconColumnFormat = (oracle.cabo.ui.data.DictionaryData)
    myColumnFormats.getItem(pageContext.findChildIndex(tableBean,<yourChildItemName>));

  // set the CELL_NO_WRAP_FORMAT_KEY property to TRUE
  myIconColumnFormat.put(CELL_NO_WRAP_FORMAT_KEY, Boolean.TRUE);
  ...
}
```

Note: The `setWrapEnabled(true/false)` API may be used to set the wrap behavior prior to calling `prepareForRendering`.

Attention: If your table includes a column that contains a messageTextInput item and you change the column formatting for that column immediately after your `myColumnFormat.put()` call, you should call the `setDataAlignment(String dataAlignmentFormat)` method of `oracle.apps.fnd.framework.webui.beans.message.OAMessageTextInputBean` on the messageTextInput item under that column. `dataAlignmentFormat` may only be `TEXT_FORMAT` or `NUMBER_FORMAT`, but should be set to match the column format. If you do *not* call the `setDataAlignment` method, the messageTextInput field itself will be aligned with the column header, but the data within the field will not be aligned.

Attention: If your table includes a column that contains a messageLovInput item with a `Data Type` of NUMBER, the data is always right-aligned. However, for the column containing the messageLovInput item as its child, only the column header and not the column data can be left-aligned. In Release 12, you can now use the `oracle.apps.fnd.framework.webui.beans.message.OAMessageLovInputBean` API `setDataAlignment(String dataAlignFormat)` to specify the data alignment for a messageLovInput item. The valid values for the `dataAlignFormat` parameter are `NUMBER_FORMAT` and `TEXT_FORMAT`.

For example, to left-align the data in a messageLovInput item with data type NUMBER, you would include code such as the following in your controller:

```
OAMessageLovInputBean lovBean =
  (OAMessageLovInputBean)webBean.findChildRecursive(...);
lovBean.setDataAlignment(TEXT_FORMAT);
```

To right-align the data in a messageLovInput item with data type NUMBER, you would include this code in your controller (Note: The default alignment is right-align, so normally you wouldn't need to include this code, but it is shown here to simply illustrate a point):

```
OAMessageLovInputBean lovBean =
  (OAMessageLovInputBean)webBean.findChildRecursive(...);
lovBean.setDataAlignment(NUMBER_FORMAT);
```

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Row Formatting

OA Framework does not set any default row formatting, except to render a horizontal grid line above each row.

Declarative Implementation

There is currently no declarative support for formatting the rows in a table.

Runtime Control

The UIX `TableBean rowFormats` object determines the row format of a table. To modify a row's formatting, use the `TableBean getRowFormats` method, which returns a `DataObjectList`. Each `DataObject` in the list corresponds to one row from top to bottom in the table format. Specify a format inside each DataObject to alter the formatting of each row accordingly. These formatting DataObjects store their values under the known, publicized key (UIConstants) `DISPLAY_GRID_KEY`. UIX queries each of the DataObjects for the `DISPLAY_GRID_KEY` property to determine if it returns `BOOLEAN.TRUE` or `BOOLEAN.FALSE`. Each one that returns `BOOLEAN.FALSE` from this query suppresses a grid line above that row.

Note: If a DataObject returns a value for a key, the value is used to change the row format. If not, the table assumes the default format.

The following code example illustrates how you can override the default row format by suppressing a row's grid line.

```
processRequest
{
  ...
  tableBean.prepareForRendering(pageContext);

  // now get a handle on the array of Row Format Dictionaries
  DataObjectList myRowFormats = tableBean.getRowFormats();
  if (myRowFormats = null)
  {
    DictionaryData[] myRowFormats = new DictionaryData[size];
  }

  // set the DISPLAY_GRID_KEY property to FALSE for the second row
  // which corresponds to index 1 in the DictionaryData array
  rowFormats[1] = new DictionaryData(DISPLAY_GRID_KEY, Boolean.FALSE);

  //Modify any other row formats
  ...

  tableBean.setRowFormats(rowFormats);
  ...
}
```

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Column Header/Row Header Formatting

The only time you need to format a column header or a row header is when you need to control its ability to wrap its content. The default is to allow wrapping as not doing so can cause the table to grow too wide.

Declarative Implementation

There is currently no declarative support for formatting the column headers and row headers in a table.

Runtime Control

To modify a column header or row header formatting, you can use the `TableBean getColumnHeaderFormats` or `getRowHeaderFormats` method, respectively. These methods returns a `DataObjectList`. Each `DataObject` in the list corresponds to one column header from left to right or one row header from top to bottom, in the table format. You specify formats inside each `DataObject` to alter the formatting of each column header or row header accordingly. These formatting `DataObjects` store their values under the known, publicized key (UIConstants) `CELL_NO_WRAP_FORMAT_KEY`. This property determines whether the column header or row header should be rendered with line wrapping disabled. To override the default, return `Boolean.TRUE` to render the column header or row header without wrapping the contents.

Note: If a DataObject returns a value for a key, the value is used to change the row format. If not, the table assumes the default format.

The following code example illustrates how you can modify wrap settings for a column:

```
// To modify wrap settings for the column named <columnName>
if (tableBean.getColumnHeaderFormats() == null)
{
  DictionaryData columnHeaderFormats[] = new
    DictionaryData[tableBean.getIndexedChildCount(null)];
  int columnNumber = pageContext.findChildIndex(tableBean, "<columnName>");

  columnHeaderFormats[columnNumber] = new DictionaryData();
  columnHeaderFormats[columnNumber].put(CELL_NO_WRAP_FORMAT_KEY,
    Boolean.TRUE);
  tableBean.setColumnHeaderFormats(new ArrayDataSet(columnHeaderFormats));
}
```

The following code example illustrates how to use the default wrapping information defined declaratively using the `No Wrap` property on the column (cell) for the column header:

```
int childCount = tableBean.getIndexedChildCount(null);
DictionaryData columnHeaderFormats[] = new DictionaryData[childCount];
for (int i = 0; i < childCount; i++)
{
  columnHeaderFormats[i] = new DictionaryData();

  // Get column wrapping info
  Boolean isWrapEnabled = ((OAWebBean)tableBean.getIndexedChild(i)).isWrapEnabled();
  Boolean cellNoWrap = isWrapEnabled ? Boolean.FALSE : Boolean.TRUE;

  // Set header wrapping info
  columnHeaderFormats[i].put(CELL_NO_WRAP_FORMAT_KEY, cellNoWrap); }
```

Personalization Considerations

- See a summary of Classic Tables personalization considerations in the Oracle Application Framework Personalization Guide.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Table Row Dynamic Poplist

OA Framework provides programmatic support for the implementation of a choice list (poplist or pulldown) in an updatable multi-row table, such that its poplist values can vary on a row-by-row basis.

Declarative Implementation

This feature can not be declaratively implemented.

Runtime Control

Refer to Dynamic Poplists in Standard Web Widgets for programmatic instructions.

Personalization Considerations

See a summary of Standard Web Widgets personalization considerations in the Oracle Application Framework Personalization Guide if you plan to implement a dynamic poplist in an advanced table.

Usage Restrictions

- Currently, there are no known usage restrictions for this feature.

Table Performance Issues

There are a number of situations in BC4J that can trigger all the rows to be brought in. Observe the following:

- Do *not* issue `getRowCount()/last()/setFetchMode(FETCH_ALL)` on your table view object if you want to use incremental fetch logic.
- Avoid `setRangeSize(-1)`. If your view object already contains some rows and you issue `setRangeSize`, BC4J tries to fill up the range by bringing in rows from the database if the number of rows in the VO is less than the range size that you try to adjust to. `setRangeSize(-1)` would cause all the rows to be brought in in this case.
- If you need to use the table view object to iterate the rows before rendering the table, use it with caution.

  Note There may be cases where you would like to set the range start to 0 and the range size to be equal to the total row count to iterate all the rows in the view object. After the iteration,

reset the range start and size to the original values.

- When you set the range size to a size that is potentially larger than the current size, set the range start first to 0, then set the range size. If you set the range size first and then range start afterwards -- and your current range start is greater than 0, BC4J faults in some rows from the database to fill up the range when it sets the new range size. BC4J then resets the range start to 0 as well as part of the `setRangeSize` operation if the new range size is different from the current range size.
- When you set the range size to a size that is potentially smaller than the current size (usually when you restore the original range size and start), then reverse the order, set the range size first and then the range start (since you may want to set the range start to a value greater than 0).

These rules may sound a little odd and confusing to you, because BC4J performs some implicit fill up logic, but these rules are based on some heuristics and experiments. In general you should be careful when you use the `setRangeSize` and `setRangeStart` methods on a table view object.

- The table rendering logic makes the following three assumptions to render a table without any extra logic that would degrade performance (to avoid unnecessarily re-setting and adjusting the range size multiple times):

1. For the table to render properly, make sure the table view object range start is the default value 0 before the table bean is created and rendered for the first time. If your code performs row navigation through methods like the view object `next` or `last` methods before the table bean is rendered, these methods move the range start to a value greater than 0. (The view object methods `first` or `previous` moves up the range.) Please perform the following in this case:

   Option 1:

   ```
   int fetchedRowCount = vo.getFetchedRowCount();
   if (fetchedRowCount > 0)
   {
     // Save the range start and size.
     int savedRangeStart = vo.getRangeStart();
     int savedRangeSize = vo.getRangeSize();

     // When you set the range size to a size that
     // is potentially larger than the current size,you should
     // set the range start first to 0 and then set the range size.
     vo.setRangeStart(0);
     vo.setRangeSize(fetchedRowCount);
   }
   ```

   Try:

   ```
   {
     for (int i = 0; i < fetchedRowCount; i++)
     {
       //Or you can use getAllRowsInRange() outside the for loop.
       Row row = vo.getRowAtRangeIndex(i);
       ...
       // Perform row operation.
     }
   }<</pre>

   Finally:

   {
     // Restore the original range - do in finally clause,
     // in case action on row throws an exception.
     // When you set the range size to a size that is
     // potentially smaller than the current size
     // (usually when you restore the original range size and start),
     // then reverse the order
     // -- that is, set the range size first and then the range start.
     vo.setRangeSize(savedRangeSize);
     vo.setRangeStart(savedRangeStart);
   }
   ```

   Option 2:

   ```
   Alternatively, use a secondary iterator if you do not want to deal with
           messing up the default row set range or row currency:


   RowSetIterator deleteIter = vo.createRowSetIterator("deleteIter");
   int fetchedRowCount = vo.getFetchedRowCount();
   if (fetchedRowCount > 0)
   {
     deleteIter.setRangeStart(0);
     deleteIter.setRangeSize(fetchedRowCount);
     for (int i = 0; i < fetchedRowCount; i++)
     {
      Row row = deleteIter.getRowAtRangeIndex(i);
      ...
     }
   }


   finally:


   {
   if (deleteIter != null)
     deleteIter.closeRowSetIterator();
   }


   Note: You can also use the following convenience methods to find rows matching some criteria: OAViewObject.getFilteredRows
           and OAViewObject.getFilteredRowsInRange.
   ```

2. Exception to assumption 1: If you redirect to current page with `retainAM=Y` in the URL, the range start can be greater than 0 if the user is already in some range other than the first range. To preserve the current table range upon redirect, place the following check on the `executeQuery` call:

   ```
   if (!vo.isPreparedForExecution())
     vo.executeQuery();
   ```

This is because `executeQuery` resets the range start to 0, which may not be what you want when you redirect with `retainAM=Y`.

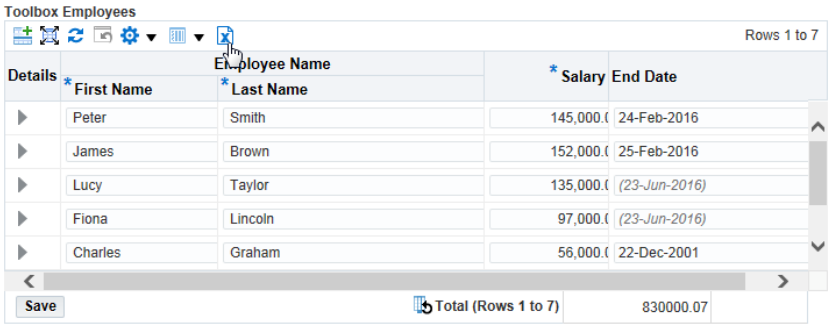
3. If you programmatically set the number of rows displayed for the table, it takes effect *only* if the table view object range start is initially 0.

Row-Level Error Prefix

You can now override the default row prefix that OA Framework displays for row and attribute error messages in a table. See the Overriding the Row-Level Error Prefix in the Error Handling document for additional information.

Export Table Data

As of Release 12.2.6, you can enable the export of table data from tables by simply setting the profile option FND: Enable BNE Table Integration. When you enable this feature and select the export table data icon in the table control bar, the data in the view object of the associated table bean exports to a Microsoft Excel spreadsheet that you can download to your machine. This feature invokes Oracle Web Applications Desktop Integrator (Oracle WebADI) to create the Excel spreadsheet. For more information about Oracle WebADI, refer to the "Accessing Oracle Web Applications Desktop Integrator" topic in the "Introduction to Oracle Web Applications Desktop Integrator" chapter of the *Oracle Web Applications Desktop Integrator Implementation and Administration Guide*.



The following configurations do not support the export table data feature:

1. Tables rendered in the email facet.
2. Tables rendered in printable pages.
3. Inner Tables (table-in-table).
4. Tables in pages running in embedded mode, such as tables inside a JTT skin.
5. Tables in pages running in portlet mode.
6. List of values table web beans.
7. Attachment tables.
8. Operating systems other than Windows.

Exporting Tables With Switcher Columns

Be aware of the following important requirements when exporting tables that have switcher columns.

Scenario	Expected Result
One of the columns and the child 0 under column is a switcher. In all such cases, children are non web bean data attributes such as image beans.	The column does not appear in Excel.
One of the columns is a switcher and one case is a non web bean data attribute (image). Another case is a web bean data attribute (link or message styled text).	The column appears in Excel. For rows that evaluate to the case non web bean data attribute, you must observe blank cells. For rows that evaluate to the case web bean data attribute, you must observe the data in Excel.
One of the columns is a switcher and all the cases are web bean data attributes (link or message styled text). We assume case 1 has the view attribute va1, and case 2 has the view attribute va2.	The column appears in Excel. For rows that evaluate to the case va1, you must observe data that is obtained from the row getAttribute(va1). For rows that evaluate to case va2, you must observe data that is obtained from row.getAttribute(va2).
One of the columns is a switcher, and all the cases are web bean data attributes (link or choice). We assume that case 1 has the view attribute va1, and case 2 has the view attritbute va2 if the second case is a choice.	The column appears in Excel. For rows that evaluate to the case va1, you must observe data that is obtained from row.getAttribute(va1). For rows that evaluate to the case va2, you must observe data that is obtained from row.getAttribute(va2). As the second case is a choice, you must see the name (Pick List Display Name) and not the value (Pick List Value attr) of the option.

Rich Table Interactions

To enable the rich interactions introduced in Release 12.2.3, set the profile **FND: Enable Rich Table Interactions** to **True**.

Note: If you set the profile **FND: Enable Rich Table Interactions** to **True**, OA Framework creates a user view of any rich table modifications you make. The user view always takes highest precedence when OA Framework displays the table. As a result, if you reorder a table using the admin Personalization Framework UI, the saved admin personalization will not take effect. If you wish to display the table columns as ordered according to your admin personalization, you can set the **Enable Column Reorder** property to false for those tables for which you feel admin personalization should take precedence. If you set the profile **FND: Enable Rich Table Interactions** to **False**, you disable all rich interactions across all products and tables, as well as disable mobile gestures.

Known Issues

- See a summary of key table issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
  - Table Overview
  - Table Layout
  - Table Subcomponents
  - Manipulation of Table Display
  - Common Table Actions
  - Table Navigation/Action Methods
  - Personalization of Table Views
- Javadoc File(s)
  - `oracle.apps.fnd.framework.webui.beans.table.OATableBean`

    - oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean
    - oracle.apps.fnd.framework.webui.beans.OAWebBean
    - oracle.apps.fnd.framework.webui.beans.OAWebBeanDataAttribute
    - oracle.apps.fnd.framework.webui.OAWebBeanConstants
    - oracle.apps.fnd.framework.webui.beans.OAWebBeanTable
    - oracle.apps.fnd.framework.webui.beans.table.OAAddTableRowBean
    - oracle.apps.fnd.framework.webui.beans.table.OATotalRowBean
    - oracle.apps.fnd.framework.webui.beans.message.OAMessageLovInputBean
    - oracle.apps.fnd.framework.webui.OAInnerDataObjectEnumerator
    - oracle.cabo.ui.beans.table.AddTableRowBean
    - oracle.cabo.ui.beans.table.TableBean
    - oracle.cabo.ui.beans.table.TableStyle
- Lesson(s)
- Sample Code
- FAQs
    - Tables and Advanced Tables

Color Picker

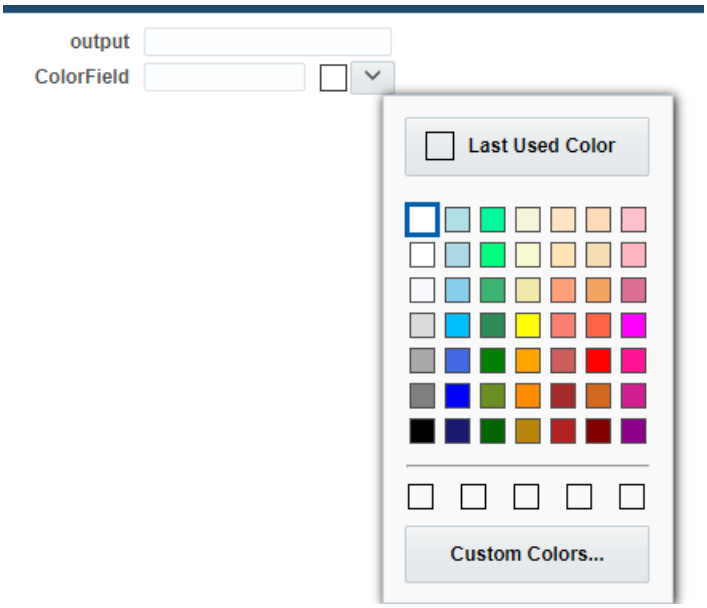
Overview

Oracle E-Business Suite users can enter a color for a color field using a Color Picker. This feature comprises a popup that shows a color palette - a set of fixed colors.

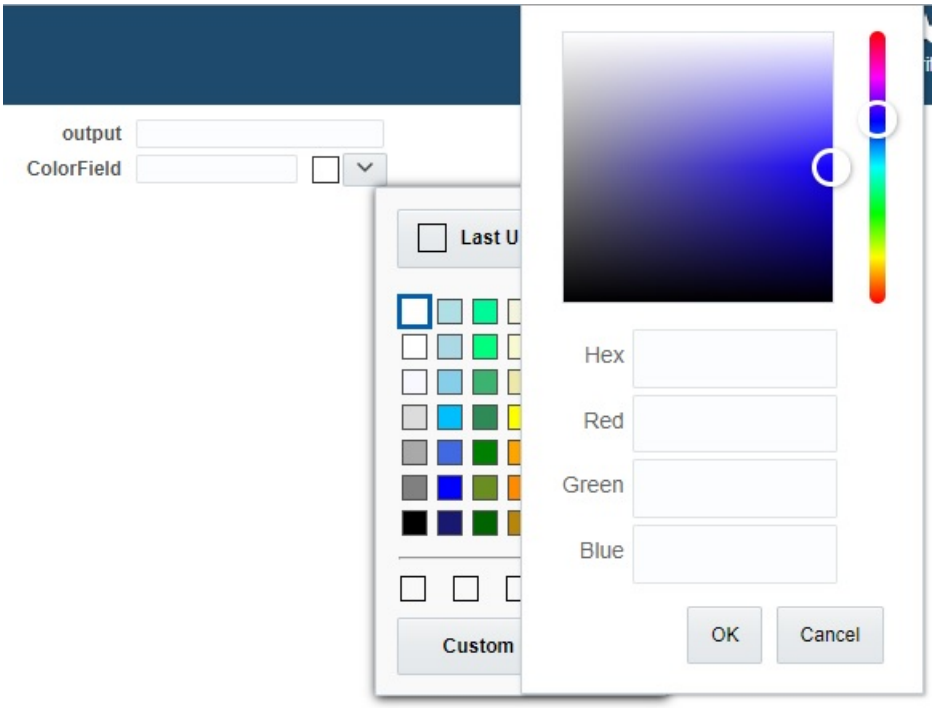
A second-level popup can be opened from the color palette popup. This popup displays a color spectrum, where a color can be picked from a distribution of colors rather than a fixed set.

The Color Picker feature is used in the Theme Editor.

Color Palette Popup



Color Spectrum Popup



Runtime Control

Color Picker needs to be added programmatically to the desired page, as follows:

```
OAMessageColorFieldBean colorBean = (OAMessageColorFieldBean) createWebBean(pageContext,
MESSAGE_COLOR_FIELD_BEAN, null, "USER_DEFINED");
colorBean.setName("USER_DEFINED_NAME");

colorBean.setViewAttributeName("FirstName");
colorBean.setPrompt("USER_DEFINED_PROMPT");

colorBean.setDataType("VARCHAR2");

OAMessageComponentLayoutBean colorRn = (OAMessageComponentLayoutBean)
webBean.findChildRecursive("region1");
colorRn.addIndexedChild(colorBean);
```

Technical Design

- The launch button has an ID of `<colorpickerprompt>$swbw` (originally <`colorpickerprompt>$sw`).
- The color palette popup is an OAF embedded popup, and all its contents are from JET: namely, Last Used Color, Color Palette, Recent Color Buttons, Custom Color Buttons.
- The color spectrum popup and all its contents are also from JET.
- For a color field present in an embedded popup, a popup in popup is used. However, since the color spectrum popup is from JET, the case of popup >popup >(JET) popup is valid.

Known Issues

- Color Picker does not work properly if the color field is added to a parameter popup.
- In Firefox, for a color field in a a popup (embedded popup with no Close button), on selecting a value from the color spectrum the focus goes to the base page instead of the color field popup as it should.
- For a color spectrum popup, the values for Hex, R, G, B are not validated.

User Interface

Concurrent Processing: Request Submission and Monitoring

Overview

The Concurrent Processing Request Submission and Monitoring user interfaces are available as OA Framework-based pages. The Request Submission flow includes multiple pages that steps the user through submitting a specific program or allowed programs (based on their responsibility) to run. The Request Monitoring (Viewing) flow includes pages that allow users to search for and view their concurrent requests.

Oracle Application Object Library provides a seeded menu that contains the standard functions for Request Submission and Monitoring. The internal name for this menu is **FND_REQUESTS** (user menu name is "**Requests**"), and contains the standard seeded functions **FNDCPSRSSSWA** and **FNDCPVIEWREQUEST**. You can add this menu to your application if you want to utilize these functions. Alternatively, as this document describes, you can create your own functions to launch Request Submission and Monitoring from your application.

Note: This document assumes you are familiar with the Concurrent Processing feature. Refer to the *Oracle E-Business Suite User's Guide* or *Oracle E-Business Suite Setup Guide* if you need further information.

Contents

- Adding Request Submission to Your Product
  - Declarative Implementation
    - Configuring Request Submission with Optional URL Parameters
- Adding Request Monitoring to Your Product
  - Declarative Implementation
    - Configuring Request Monitoring with Optional URL Parameters
  - Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

Adding Request Submission to Your Product

The Standard Concurrent Request Submission flow takes place across the following six pages with a train implementation:

- **Program Name** - allows the user to select the program to submit. It contains a required field called Program Name.
- **Parameters** - if the program has required parameters, this page is displayed, otherwise, it is skipped.
- **Schedule** - allows the user to schedule when the program runs. It contains the default scheduling of "As Soon As Possible".
- **Notifications** - allows the user to optionally send notifications to designated users when the request completes.
- **Printer** - allows the user to optionally specify printing options with the configured request.
- **Summary** - displays a summary of submitted requests.

With Oracle E-Business Suite, you can restrict some of the train steps that a user must navigate through to submit a concurrent request. You can configure request submission within your application such that some of the information required for a request is pre-entered. End-users need only enter a limited amount of data specific to a current request. As a result, you can hide some of the pages described above from the end-user, because they are no longer necessary.

Declarative Implementation

To call the first page of Request Submission from your application, registering your function in the **Form Functions** form with the following URL parameters in the **HTML Call** field:

```
akRegionCode=FNDCPPROGRAMPAGE

akRegionApplicationId=0
```

Note: The FND seeded function name that contains these parameters is **FNDCPSRSSSWA**.

Configuring Request Submission with Optional URL Parameters

If you wish to restrict some of the pages in the Request Submission flow, you can do so by specifying the following parameters in the URL that you specify in the **HTML Call** field of the **Form Functions** form:

- **requestURL = <*URL value*>**

  URL Value is the page to which you want the user to be redirected to when the user selects Cancel or when the request is submitted. If this parameter is not specified, the user is redirected to the **View Requests** page.

- **programApplName =<*program application short name*>& programName=<*program short name*>**

  Use these parameters to allow your users to submit a specific program from your application without having to enter a program name. If these parameters are specified by the calling application, the Request Submission **Program Name** page will render without the **Program Name LOV**. The **Program Name field** will instead display as a text-only item showing the program specified.

- **programDesc=<*program description*>**

  Use this parameter to provide a description to your request. The description appears as **Request Name** on the **Request Submission Program Name** page. You can later search for requests based on the supplied request name (description).

- **programRegion=Hide**

  Use this parameter to hide the **Program Name** page during request submission. Note that if you specify this parameter, you must also indicate the program name to submit by coding the calling application to provide the **programApplName** and **programName** parameters.

- **parameterRegion=Hide**

  If the program you are configuring to submit does not require any parameters, you can specify this parameter with the value 'Hide'. This parameter hides the **Parameters** page during request submission.

- **scheduleRegion=Hide**

  If you do not want to allow users the option of scheduling the request submission, specify this parameter with the value 'Hide' to hide the **Schedule** page. By default, if the **Schedule** page is hidden, the request is submitted with the schedule, "As Soon As Possible".

- **notifyRegion=Hide**

  Specify this parameter with the value 'Hide' to hide the **Notification** page during request submission.

- **printRegion=Hide**

  Specify this parameter with the value 'Hide' to hide the **Printer** page during request submission.

- **pageTitle=<*page title*>**

  Specify this parameter to set the page title for the Request Submission flow.

Examples

1. To launch the full Request Submission flow:

```
HashMap parameters = new HashMap();
String url = "OA.jsp";
parameters.put("akRegionApplicationId", "0");
parameters.put("akRegionCode", "FNDCPPROGRAMPAGE");
```

2. To launch the Request Submission flow specifically for the Active Users program (program short name=FNDSCURS):

```
HashMap parameters = new HashMap();
String url = "OA.jsp";
parameters.put("akRegionApplicationId", "0");
parameters.put("programApplName", "FND");
parameters.put("programName", "FNDSCURS");
parameters.put("programDesc", "Some Description");
```

3. To launch the Request Submission flow specifically for the Active Users program (FNDSCURS), such that the program is submitted with default values only (that is, you do not want the user to provide any information in the request submission):

```
HashMap parameters = new HashMap();
String url = "OA.jsp";
parameters.put("akRegionApplicationId", "0");
parameters.put("akRegionCode", "FNDCPPROGRAMPAGE");
parameters.put("programApplName", "FND");
parameters.put("programName", "FNDSCURS");
parameters.put("programDesc", "Some Description");
parameters.put("programRegion", "Hide");
parameters.put("parameterRegion", "Hide");
parameters.put("scheduleRegion", "Hide");
```

```
    parameters.put("notifyRegion", "Hide");
    parameters.put("printRegion", "Hide");

parameters.put("requestMode", "DEFERRED");
parameters.put("requestId", <req id>);

if (pageContext.getParameter("uploadCat") != null)
  {
    try
      {
        // get the JDBC connection
        Connection conn = (Connection)
          ((RequestAMImpl)pageContext.
           getApplicationModule(webBean)).getDBConnection();
        ConcurrentRequest cr = new ConcurrentRequest();
        // set it as deferred request mode.
        cr.setDeferred();
        // call submit request
        int reqId = cr.submitRequest(progAppl, progName, desc,
          scRSTime, false, para);

        dbTran.commit();

        // redirect page to Request Scheduling page

        com.sun.java.util.collections.HashMap parameters = new HashMap();

        String url = "OA.jsp";
        parameters.put("akRegionApplicationId", "0");
        parameters.put("akRegionCode", "FNDCPPROGRAMPAGE");
        String id = "" + requestId + "";
        parameters.put("requestMode", "DEFERRED");
        parameters.put("requestId", id);

        pageContext.setForwardURL(url,
                                  null,
                                  OAWebBeanConstants.KEEP_MENU_CONTEXT,
                                  null,
                                  parameters,
                                  true,
                                  OAWebBeanConstants.ADD_BREAD_CRUMB_NO,
                                  OAWebBeanConstants.IGNORE_MESSAGES);
      }
      catch { ...}
  }
```

Adding Request Monitoring to Your Product

The Request Monitoring user interface provides the ability to search for a current user's requests based on three categories: all requests, completed requests and running requests. It also allows a user to specify criteria to search for a request based on a specific request ID, requests for a specific program or a range of scheduled requests.

Using the search results list, a user can select a request to see the details of that request or view its output file. From the Details page for a request, a user can place a hold on a pending request, cancel a pending request or view the request's log file.

Declarative Implementation

To call the first page of Request Monitoring from your application, registering your function in the **Form Functions** form with the following URL parameters in the **HTML Call** field:

```
akRegionCode=FNDCPREQUESTVIEWPAGE

akRegionApplicationId=0
```

Note: The FND seeded function name that contains these parameters is **FNDCPVIEWREQUEST**.

Example

To launch the View Requests page:

```
OA.jsp?akRegionCode=FNDCPREQUESTVIEWREGION&akRegionApplicationId=0
```

Configuring Request Monitoring with Optional URL Parameters

If you wish to restrict some of the pages in the Request Monitoring flow, you can do so by specifying the following parameters in the URL that you specify in the **HTML Call** field of the **Form Functions** form:

- **printRegion = Hide**

  Specify this parameter with the value 'Hide' to hide the **Print** region in the Request Details page, so that users do not see the print information for a request.

- **requestID =<*request id*>**

  Specify this parameter, to search for a specific request based on the supplied *request id*.

- **requestDesc =<*request description*>**

  Specify this parameter, to search for request(s) based on the supplied *request description*.

- **startDate =<*startdate*>**

  Specify this parameter, to search for request(s) based on the supplied *startdate*.

- **endDate =<*enddate*>**

  Specify this parameter, to search for request(s) based on the supplied *enddate*.

- **progApplShortName =<*program application short name*>&progShortName=<*program short name*>**

  Use these parameters to search for requests for a specific program.

Runtime Control

There are no runtime control steps necessary to add request monitoring to your product.

Personalization Considerations

- See a summary of Concurrent Processing personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None.

Related Information

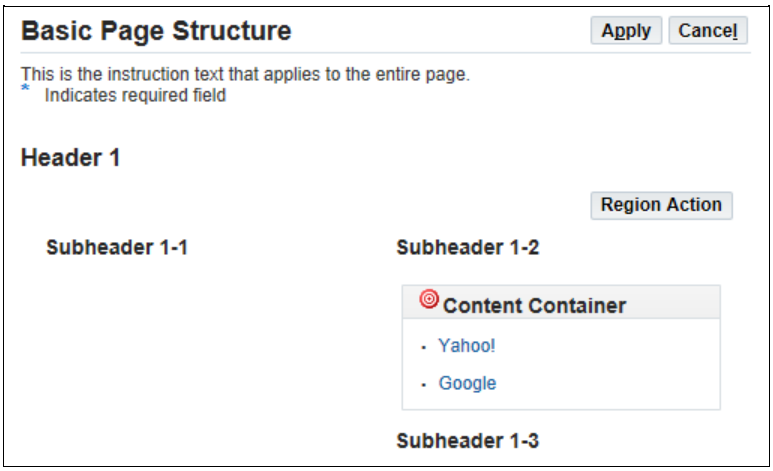
- BLAF UI Guideline(s)
- Javadoc File(s)
- Lesson(s)
- Sample Code

Content Containers

Overview

Per the Oracle Browser Look-and-Feel (BLAF) UI Guideline Content Containers in Page, content containers are regions with special graphical characteristics (border, shape and background color) that help set off the information they contain from the rest of the page. They are often used to hold related information or shortcuts as shown in Figure 1.

Figure 1: Example of content containers used for related information and shortcuts.



Implementation

Content containers can be added to the side of the page in a dedicated column as shown in Figure 1 (so the page content renders beside them), or within the page content as just another region. The implementation instructions differ somewhat in each case.

Within Page Content

To add a content container to a region within your page contents:

Step 1: In the JDeveloper structure pane, select the region to which you want to add your content container, right-click and select **New > Region**.

Step 2: Specify an `ID` in accordance with the OA Framework Naming Standards and set the `Region Style` to **contentContainer**.

Step 3: Set the `Text` property to the title that you want to display in the content container, and optionally set the `Image URI` to the name of the BLAF image you want to display (for example: `bullseyeicon_cctitle.gif`).

Tip: See Images in Your Pages for information on using the BLAF icon repository.

Step 4 (optional): By default, content containers render with a light background as shown above. If you want a different color scheme, set the `Background Shade` accordingly (valid options are presented in a list).

Step 5: Set the `Width` to **100%** to ensure that the content container renders fully within the space allowed.

Step 6: Add regions and items to the **contentContainer** region as you normally would. For example, to create the Shortcuts content container shown in Figure 1, you would first add a **bulletedList** region to the **contentContainer**, and then add **link** items to the **bulletedList**.

At runtime, the OA Framework instantiates an `oracle.apps.fnd.framework.webui.beans.layout.OAContentContainerBean`.

Beside Page Content

To add a content container to the side of your page:

Step 1: Create a shared region for your content container (see the Shared Regions instructions in Implementing the View if you need help).

Step 2: Configure your content container region as described above.

Step 3: Instantiate your content container programmatically and add it to the page in the named "end" area as shown:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{

  // Always call this first.

  super.processRequest(pageContext, webBean);
```

```
  // Instantiate the content container that you defined declaratively.

  OAContentContainerBean content =
    (OAContentContainerBean)createWebBean(pageContext,
                                          "/oracle/apps/fnd/framework/toolbox/samplelib/webui/ContentContainRN",
                          null, // ignore item for regions
                          true); // using OAExtension
  // Get the page layout bean, and specify the content container as the
  // end content.

  OAPageLayoutBean pageLayoutBean = pageContext.getPageLayoutBean();
  pageLayoutBean.setEnd(content);

}
```

Note: For information about the "start" and "end" areas in a page (including ensuring that you have the correct amount of horizontal space between your primary page content and anything that you render in the start and end areas), see Page Layout (How to Place Content).

Tip: If you want to add multiple content containers to the "end" area, see the Home Page example in the ToolBox Tutorial / Sample Library. Even though the content containers are fully constructed programmatically in the example, it will show you how to place them on a stack layout with vertical space between them. You would follow the same example even if you instantiated declaratively defined content containers as shown above.

Personalization Considerations

- None

Known Issues

- None

Related Information

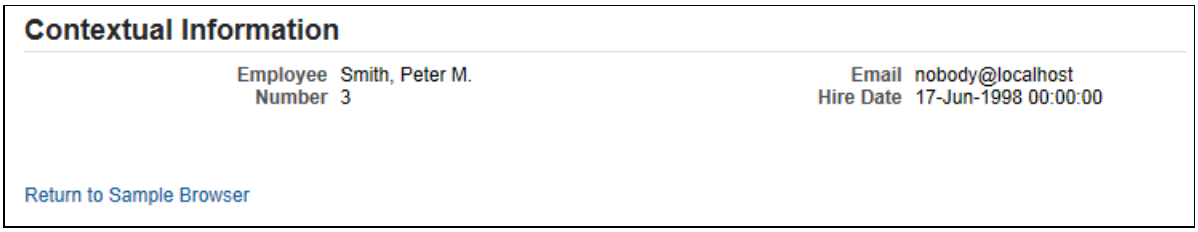
- BLAF UI Guidelines
  - Content Containers in a Page
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAContentContainerBean`
- Developer's Guide
  - Bulleted List
  - Related Links / Shortcuts
  - Images in Your Pages
  - Page Layout (How to Place Content)
- OA Framework ToolBox Tutorial / Sample Library
  - Home Page Lab
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.BasicStructPG.xml`
  - `oracle.apps.fnd.framework.toolbox.tutorial.webui.HomePageCO.java`

Contextual Information

Overview

In a multipage task flow, it is often helpful to remind users of selection(s) made in previous pages by displaying static, contextual information at the top of the page. As described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Contextual Information this standard layout appears as follows:

Figure 1: Double Column Layout Contextual Information Example



Note that, if page-level action/navigation buttons are present, the contextual information should render *in parallel* with the top-level buttons (not below them). Furthermore, if page-level instruction text and/or keys are also present, they should render below the end of the contextual information.

Declarative Implementation

To add a contextual information region to your page:

Note: All package, region and item names must comply with the OA Framework Package / File / Directory standards.

Step 1: Create your **pageLayout** region as you normally would; be sure to specify the `Title` property (represented by the "Page Title" text value in Figure 1).

Step 2: Select your **pageLayout** region in the **Structure** pane, right-click and select **New > pageStatus**.

- JDeveloper automatically creates a **pageStatus** node and adds a **flowLayout** default region beneath it. Change this region's `Style` to **messageComponentLayout**.

Step 3: Set the **messageComponentLayout** region's `Rows` and `Columns` properties as described in the Page Layout (How to Place Content) document.

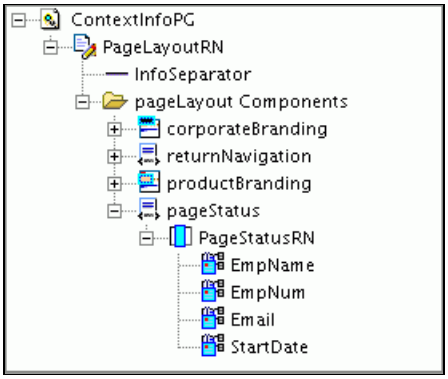
- For example, for the double column layout shown in Figure 1 above, set both the `Rows` and `Columns` property values to **2**.
- If you want to display two items next to one another, set the `Rows` property value to **1** and the `Columns` property value to **2**.

Step 4: Add the items that you want to display in the appropriate sequence. For example, to achieve the layout shown in Figure 1 above, you would need to add the corresponding items as shown in Figure 2 below.

Select the **messageComponentLayout** region, right-click and select **New > messageStyledText**. For each **messageStyledText** item:

- Specify an `Attribute Set` in accordance with the attribute set usage guidelines (see Implementing the View for general information about attribute sets, and the OA Framework View Coding Standards). This should set the `Prompt` and `Data Type` values correctly; verify that they are correct.
- Set the `CSS Class` to **OraDataText** (for display-only data that should render in bold).
- Set the `View Instance` and `View Attribute` names for the underlying data source. Note that you can use a view object that you explicitly query when this page renders, or you might bind to a cached row if this page shares a retained application module with a previous page.

Figure 2: Contextual Information Item Sequence for the Layout in Figure 1



Step 5: Select your **pageLayout** region again, right-click and select **New > Item**. Set this `Item Style` to **separator.**

Step 6: Add remaining items and regions to the **pageLayout** region as needed for your design.

Runtime Control

There are no particular runtime actions associated with these standard components used in this particular context, however, remember to execute the underlying query in a `processRequest()` method if these fields bind to a task-specific view object (see Implementing the Model and Implementing the Controller for examples of this).

Also see the Standard Web Widgets documentation for additional information about working with OAMessageStyledTextBeans, and Page Layout (How to Place Content) for information about using different layout components.

Related Information

- BLAF UI Guidelines:
  - Contextual Information
- Developer's Guide:
  - Separator
  - Standard Web Widgets
  - Page Layout (How to Place Content)
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAMessageComponentLayoutBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageStyledTextBean`
- OA Framework ToolBox Tutorial / Sample Library
  - See the `/oracle/apps/fnd/framework/toolbox/samplelib/webui/ContextInfoPG` in the `SampleLibrary.jpr`.

Custom HTML

Overview

In general, you should avoid using custom HTML and design pages using OA Framework components if you can, as they meet the NLS, accessibility, security, and Oracle Browser Look-and-Feel (BLAF) UI guidelines and standards. If you need to write custom HTML, UIX provides two alternatives that you should use instead of the OARawText bean:

- `oracle.apps.fnd.framework.webui.beans.OAHTMLWebBean` - helps you create HTML tags to output text. Although `HTMLWebBean` creates only one tag, and you would have to create a hierarchy of `OAHTMLWebBeans` to duplicate the HTML achievable with `oracle.apps.fnd.framework.webui.beans.OARawTextBean`, `OAHTMLWebBean` does provide the following added features:
  - Automatic escaping
  - Pretty printing
  - XHTML syntax support
  - Debugging assistance
- `oracle.apps.fnd.framework.webui.beans.OAFormattedTextBean` - helps you format text using the generic HTML formatting tags. Note that `OAFormattedTextBean` intentionally cannot accomplish everything that `OARawTextBean` does because it is designed to block cross-site scripting attacks and to provide only tags for formatting (and links).

Note: Be sure to set a unique ID for each component in the page via `setAttributeValue ID_ATTR` or `"id"`, otherwise these web beans by default will take the name of the component as the ID, which may result in cases of duplicate IDs. For example:

```
OAHTMLWebBean td0 = (OAHTMLWebBean)createWebBean(pageContext, HTML_WEB_BEAN, null, "td0");
td0.setAttributeValue("id", unique_id);
```

Displaying PDF Content Inline on Pages

Tip: If you need to display some pdf content that's generated dynamically, we generally recommend that you use `OAMessageDownloadBean` instead of displaying the pdf content inline on a page.

The following sample code, which you should add to your Controller, shows how to display pdf content inline on a page using `OAHTMLWebBean`.

Note: You need to cache the generated pdf content in the session in advance. In `DisplayPdf.jsp`, retrieve the pdf content from the session, stream it out to the response, and then remove the cached pdf content from the session.

```
 OAHTMLWebBean pdfElement =
  (OAHTMLWebBean)createWebBean(pageContext, HTML_WEB_BEAN, null, "IFRAME");

//use IFRAME or EMBED
pdfElement.setHTMLAttributeValue("src", "/OA_HTML/DisplayPdf.jsp");

//pdfBlob
// p_output
pdfElement.setHTMLAttributeValue("width", "100%");
pdfElement.setHTMLAttributeValue("height", "500");
```

Using HTML Tags in Messages

You can insert HTML tags in messages displayed in a message box, inline tip, and dialog page, and in messages associated with tip region items (`oracle.apps.fnd.framework.webui.beans.OATipBean`). You can also create your own styled text region item with HTML tags in the message text using the **formattedText** region item style (`oracle.apps.fnd.framework.webui.beans.OAFormattedTextBean`).

Internally, all these messages utilize the UIX `oracle.cabo.ui.beans.FormattedTextBean` to incorporate the HTML tags.

To see which HTML tags are allowed in `FormattedTextBean`, refer to the UIX `FormattedTextBean` Javadoc. Some notes regarding the `FormattedTextBean`:

- The `FormattedTextBean` also allows HTML links through the "`<a href ...>`" HTML tag.

  Note: If the HTML link references an OA Framework page, you need to first process its URL to retain the session values associated with the current transaction so that breadcrumbs can be retained. For example:

  ```
  String Url = "OA.jsp?..";
  OAUrl url = new OAUrl(Url);
  Url = OAUrl.createURL(renderingContext);
  MessageToken [] tokens = {new MessageToken("TOKEN_NAME", Url)};
  formattedTextBean.setText(pageContext.getMessage("FND", msgName, tokens );
  ```

- For messages in a message box, inline tip, and dialog page, and for messages associated with tip region items, UIX takes care of rendering the message text with the proper CSS style. However, if you create a formattedText region item yourself, you also need to specify the CSS style yourself. You can do so declaratively by setting the CSS Class property in OA Extension or programmatically through the `OAFormattedTextBean setCSSClass` method.

For Messages in a Message Box or Inline Tip:

When you seed your message in the `FND_NEW_MESSAGES` table, enclose your text in `<html>...</html>`.

Example:

```
<html>User authentication failed.<br>Cause: Invalid password.</html>
```

Note Inline messages in a rendered table may not reflect HTML tags properly. These are known limitations coming from the browser and operating system. UIX is generating the correct syntax in the HTML output.

For Messages in a Dialog Page:

Seed your message in the `FND_NEW_MESSAGES` table with `<html>...</html>` to enable HTML tags in the dialog page message.

The dialog page uses `OAFormattedTextBean` for the description and instruction messages to enable HTML tags as long as the message text is enclosed in `<html>...</html>`. The description message still appears in bold text and the instruction message still appears in plain text even when HTML tags are enabled. To have better control over what appears as bold, you can set a null value for the description message and just use the instruction message.

Note The `OAFormattedTextBean` text itself does not require `<html></html>` enclosing tags, but the HTML-enabled messages in the message box and dialog page do. UIX automatically drops the extra `<html></html>` upon rendering the formatted text.

Personalization Considerations

- See a summary of Custom HTML personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guidelines
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.OARawTextBean`
  - `oracle.apps.fnd.framework.webui.beans.OAHTMLWebBean`
  - `oracle.apps.fnd.framework.webui.beans.OAFormattedTextBean`

Data Export

Overview

As described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Export/Import Page Templates, you can implement an Export button that exports data displayed in one or more regions of an OA Framework page to a file. Export saves the data to a .csv (comma separated values) file or a .txt (tab separated values) file that you can view in Microsoft Excel. When you select the Export button, Microsoft Windows opens a dialog box that lets you view the file either by opening it directly in Microsoft Excel, or by saving it to a designated directory to open later.

The export file format and encoding is determined by these three profile options:

- Export MIME type
- FND: NATIVE CLIENT ENCODING
- FND: Unicode in Export

Note: The program that is launched while saving exported data is dependent on each user's PC's setting. To launch a .csv file in Microsoft Excel, the .csv file type needs to be mapped to open with Microsoft Excel. If you try to open a .csv file that is not mapped to any program, Microsoft Windows will ask you to choose a program with which to open the file. A user can easily determine the file to program mapping on their PC by first selecting a .csv file in Windows Explorer, and then choosing **Properties** from the right mouse button context menu for that file. In the **General** tab of the Properties window, Opens With: lists the program that can open the selected file.

Only web beans that are rendered on the page are ever exported. If you export data from multiple regions in a page, each region appears in the export file in a tabular format, with a row of column names for that region, followed by rows of corresponding values.

Note: If a date field has a time component, Export converts the date/time field to the client browser's time zone. If a date field does not have a time component, Export does not convert that date field's time zone because no meaningful conversion is possible for a date without a time. The date is entered with respect to one time zone and must always be viewed as a day in that time zone, regardless of the location in which it is being viewed. Oracle E-Business Suite generally uses the corporate time zone for these day definitions. Dates without a time component represent the day with respect to the corporate headquarters (corporate days).

Note: Data Import functionality is currently not supported.

Note: Beginning in Release 12.2.6, rather than explicitly enabling an Export button for each table that requires a data download option, you can set the profile option FND: Enable BNE Table Integration to enable an export table icon across all classic and advanced tables that allow users to export table data to a Microsoft Excel spreadsheet.

Contents

- Exporting Data From All Regions On A Page
- Exporting Data From a Specific Region On A Page
- Personalization Considerations
- Known Issues
- Export to Excel
- Export to PDF
- Related Information

Exporting Data From All Regions On A Page

Declarative Implementation

You can declaratively enable data export for all regions in a page by creating a page-level Export button. Page-level action buttons, such as Export, render both below the page title and the page contents. If a page title is not specified for the pageLayout region, the Export button only renders below the page contents. For more information, see the Page-Level Buttons discussion in the topic for implementing Buttons (Action/Navigation).

Step 1: Select your **pageLayout** region and create a region with the style **pageButtonBar** beneath it.

Step 2: Add an item of item style **exportButton** to the pageButtonBar region to automatically expose an Export button on the page. If you have more than one button to add to the pageButtonBar, be sure to add all the buttons in ascending sequence as you want them to appear from left to right.

When a user selects the Export button, OA Framework exports the data from all the rendered regions on that page to a .csv file.

Step 3: Generally, when you export data from a page, OA Framework exports the data from the view object attribute associated with each item in each region. In some cases, when you do not want to export the data from the view attribute associated with an item, you can specify a different view attribute to export from, by setting the `Export View Attribute` property for that item. If you

don't set or set the `Export View Attribute` property to null, OA Framework exports data from the view attribute name associated with the item.

An example of using the `Export View Attribute` is when you have a composite column in a table. You may have a composite column that consist of a flowLayout region that displays an image and some text. If you do not want to export the image file name from the composite column, you can set the `Export Attribute Name` property for that image item to some other view attribute from which you want to export data.

Note: When exporting data from a table or advanced table, the data exported may be different from the data that is displayed in the leaf item of the table column. This may occur if the leaf item's `Export View Attribute` property is declaratively set to a different view attribute than the attribute whose data is displayed.

Note: OA Framework provides limited support for exporting data from a switcher column under a table or advanced table. If the direct named children of the switcher web bean are leaf nodes that contain data, then OA Framework can export the data from the appropriate leaf node. If, however, the direct named children under the switcher web bean are containers, you must use the `Export View Attribute` property and add a transient view attribute to your view object as described in the Runtime Control section in order to export the data from the appropriate child container.

Attention: If you intend to support the Export feature on a Gantt chart or HGrid, you cannot use different viewAttributeNames at different levels in the hierarchy column. All levels of the hierarchy column (that is, all nodeDefs) should have the same viewAttributeName. This is analogous to the definition of all other columns of a HGrid or Gantt. This restriction does not apply if the Export feature is not being used.

Note: When exporting data from a page, data from hidden fields (FormValue items) or from the undisplayed child of a switcher region, cannot be exported.

Step 4: Set the `Export All Rows` property on the exportButton item to **True** to export all rows from a table's view object regardless of the view object's setMaxFetchSize value. If the `Export All Rows` property is set to **False**, the total number of rows that can be exported from a table is defined by its view object's setMaxFetchsize. The default value for this property is False for existing Export buttons and True for any new Export button.

Runtime Control

There are no general runtime control steps necessary to enable the export of data from a page.

Exporting Data from a Switcher Column in a Table

OA Framework can successfully export data from a switcher column in a table *only if* the direct named children of the switcher column are leaf nodes (nodes that consist of data).

If the named children of the switcher column are container web beans (for example, flowLayout web bean), then you must also perform the following steps to export the data from the child container:

Step 1: Create a transient view attribute in the view object used for the switcher region. For example, `ExportSwitcherViewAttr` in `MyVO`.

Step 2: In Jdeveloper, set the `Export View Attribute` property on the switcher region to the transient view attribute you created in Step 1.

Continuing with the example in Step 1, suppose you have a switcher column in a table that switches between containers `child1` and `child2` and you would like to export data from the `CustomerName` or `CustomerID` view attributes under these containers, respectively. The general XML structure would look similar to the following, with the `Export View Attribute` property set to `ExportSwitcherViewAttr`:

```
- table region
  - switcher region
      (View Instance = "MyVO1", View Attribute Name = "SwitcherViewAttr",
       Export View Attribute ="ExportSwitcherViewAttr")
      - header region (case "child1")
        - messageStyledText item
            (View Instance = "MyVO1", View Attribute Name = "CustomerName")
      - flowLayout region (case "child2")
        - messageStyledText item
            (View Instance = "MyVO1", View Attribute Name = "CustomerId")
  - etc.
```

Step 3: Add code in the getter of your view object row accessor to return the data from the leaf items of the switcher column's child containers, based on the value of the switcher view attribute. Following the example in Steps 1 and 2, you would add the following code in the getter of `MyVORowImpl.java`:

```
public String getExportSwitcherViewAttr()
{
```

```
  if ("child1".equals(getSwitcherViewAttr()))
    return getCustomerName();
  else if ( "child2".equals(getSwitcherViewAttr()) )
    return getCustomerId();
  return null;
}
```

Exporting Data From a Specific Region On A Page

Declarative Implementation

To export data from a specific region on a page:

Step 1: Refer to the Oracle BLAF UI Guideline: Export/Import Page Templates for the placement of the Export button appropriate for the region for which you want to enable export. For example, if you are enabling export for a table region, the Export button should render as a right-justified global table button, above the table control bar, but beneath the table header.

Step 2: In the OA Extension xml Structure pane, determine where you want to position the Export button for your region and create a new region item of item style **exportButton.**

Step 3: Set the `View Instance` name for the exportButton region item to the same view object associated with the region for which you are enabling export. This ties the Export button to the correct data source.

Attention: Make sure you do not associate the same view instance to multiple regions within the page that you want to export. If you do, you get a "NoDefException" if the region you try to export is an HGrid region. For all other region types within the page, data displayed across all the regions that share the same view instance will be exported rather than just the data displayed in the intended region for which the "Export" button is selected. To work around this issue, you can assign different view instances of the same view to the different regions of the page.

Step 4: Generally, when you export data from a region, OA Framework exports the data from the view object attribute associated with each item in that region. In some cases, when you do not want to export the data from the view attribute associated with an item, you can specify a different view attribute to export from, by setting the `Export View Attribute` property for that item. If you don't set or set the `Export View Attribute` property to null, OA Framework exports data from the view attribute name associated with the item.

Note: When exporting data from a table or advanced table, the data that is exported from the leaf item of the table column may be from a different view attribute than the data displayed. This occurs when the declarative definition of the table column's leaf item has a value specified for its `Export View Attribute` property.

Note: OA Framework provides limited support for exporting data from a switcher column under a table or advanced table. If the direct named children of the switcher web bean are leaf nodes that contain data, then OA Framework can export the data from the appropriate leaf node. If, however, the direct named children under the switcher web bean are containers, you must use the `Export View Attribute` property and add a transient view attribute to your view object as described in the Runtime Control section in order to export the data from the appropriate child container.

Attention: If you intend to support the Export feature on a Gantt chart or HGrid, you cannot use different viewAttributeNames at different levels in the hierarchy column. All levels of the hierarchy column (that is, all nodeDefs) should have the same viewAttributeName. This is analogous to the definition of all other columns of a HGrid or Gantt. This restriction does not apply if the Export feature is not being used.

Note: When exporting data from a page, data from hidden fields (FormValue items) or from the undisplayed child of a switcher region, cannot be exported.

Step 5: Set the `Export All Rows` property on the exportButton item to **True** to export all rows from a table's view object regardless of the view object's setMaxFetchSize value. If the `Export All Rows` property is set to **False**, the total number of rows that can be exported from a table is defined by its view object's setMaxFetchsize. The default value for this property is False for existing Export buttons and True for any new Export button.

Runtime Control

In the general case, there are no runtime control steps necessary to enable the export of data from a region. However, if you wish to export data from a Master/Detail region or a switcher column within a table, there are specific programmatic steps you need to consider.

Exporting Data From Master/Detail Regions

If you have a page that contains both a Master region and a corresponding Detail region and you want to export all the data from these two regions regardless of which Master row is selected, you must do so programmatically. There is no declarative support for this.

First refer to the Coordinating Master/Detail Tables discussion for details on how to create such a region. Then add code, as shown in

the following example, to the `processRequest` method of your controller:

```
OAExportBean expBean =
  (OAExportBean)webBean.findChildRecursive("Export");
// set the exportDetails for this Master/Detail region.
expBean.setExportDetails(oaPageContext,"masterRegionName",
  "detailRegionName", "masterVuName", " detailVuName");
```

Note: This type of Master/Detail export is supported only when you implement a region-level export and not a page-level export (where all the regions on a page are exported).

Exporting Data from a Switcher Column in a Table

OA Framework can successfully export data from a switcher column in a table *only if* the direct named children of the switcher column are leaf nodes (nodes that consist of data).

If the named children of the switcher column are container web beans (for example, flowLayout web bean), then you must also perform the following steps to export the data from the child container:

Step 1: Create a transient view attribute in the view object used for the switcher region. For example, `ExportSwitcherViewAttr` in `MyVO`.

Step 2: In Jdeveloper, set the `Export View Attribute` property on the switcher region to the transient view attribute you created in Step 1.

Continuing with the example in Step 1, suppose you have a switcher column in a table that switches between containers `child1` and `child2` and you would like to export data from the `CustomerName` or `CustomerID` view attributes under these containers, respectively. The general XML structure would look similar to the following, with the `Export View Attribute` property set to `ExportSwitcherViewAttr`:

```
- table region
  - switcher region
      (View Instance = "MyVO1", View Attribute Name = "SwitcherViewAttr",
       Export View Attribute ="ExportSwitcherViewAttr")
      - header region (case "child1")
        - messageStyledText item
           (View Instance = "MyVO1", View Attribute Name = "CustomerName")
      - flowLayout region (case "child2")
        - messageStyledText item
            (View Instance = "MyVO1", View Attribute Name = "CustomerId")
  - etc.
```

Step 3: Add code in the getter of your view object row accessor to return the data from the leaf items of the switcher column's child containers, based on the value of the switcher view attribute. Following the example in Steps 1 and 2, you would add the following code in the getter of `MyVORowImpl.java`:

```
public String getExportSwitcherViewAttr()
{
  if ("child1".equals(getSwitcherViewAttr()))
    return getCustomerName();
  else if ( "child2".equals(getSwitcherViewAttr()) )
    return getCustomerId();
  return null;
}
```

Personalization Considerations

There are no personalization considerations.

Known Issues

- See a summary of key data export/import issues with suggested workarounds if applicable.

Export to Excel

This feature:

- Provides the capability to export an OA Framework table data from a supported browser to Excel.
- Honors table settings, thereby exporting only those columns that are displayed / turned on by the user through table settings.

Site Level Control

The feature can be controlled at site level through the profile option 'FND: Enable BNE Table Integration' / FND_ENABLE_BNE_TABLE_INTEGRATION. The profile options can be set at all levels, and by default is set to 'False'. The same profile option also controls export to PDF.

Metadata Control

Table (Classic/Advanced) Metadata Property	Default Value	Allowed Values	Property Meaning
Enable Excel Export	Null	True, False	Enables export to Excel menu option in table actions

Programmatic Control

There is no programmatic control for this feature.

Unsupported Tables for Export to Excel

- Tables rendered in the email facet.
- Tables rendered in printable pages.
- Inner Tables (table-in-table).
- Tables in pages running in embedded mode, such as tables inside a JTT skin.
- Tables in pages running in portlet mode.
- List of values table web beans.
- Attachment tables.
- Tables whose view object is dirty any time. Dirty View Objects indicate ongoing transactions, hence export will be disabled.
- Tables whose view object does not have any where clause or where clause params. These tables fall under blind query category (see Footnote)
- Tables whose view object binding style is Oracle Named and not Oracle Positional. As per the OA Framework Model Coding guidelines, the binding style for the table view object must be set to Oracle Positional.
- Tables on operating systems other than Windows. Excel export is based on Oracle Web Applications Desktop Integrator, which is not supported on Mac.
- For guest users, we recommend turning off the profile option FND_ENABLE_BNE_TABLE_INTEGRATION.

Footnote: There are a few cases when the view object associated with the table will not have a where clause or any where clause parameters. The database tables may be populated at runtime. In certain cases, even with a blind query from the UI, the number of rows the user sees does not exceed a certain reasonable limit. To enable OA Framework to skip the blind query check in such cases, the developer can set the property Enable PDF Export to 'True', which causes OA Framework to skip the the blind query check.

Export Data Limit

OA Framework will export records up to the limit controlled by the profile option 'FND: View Object Max Fetch Size' / VO_MAX_FETCH_SIZE. There is currently no option to export all rows as such.

Export to PDF

This feature:

- Provides the capability to export an OA Framework table data from a supported browser to PDF.
- Honors table settings, thereby exporting only those columns that are displayed / turned on by the user through table settings.

Site Level Control

The feature can be controlled at site level through the profile option 'FND: Enable BNE Table Integration' / FND_ENABLE_BNE_TABLE_INTEGRATION. The profile options can be set at all levels, and by default is set to 'False'. The same profile option also controls export to Excel.

Metadata Control

Table (Classic/Advanced) Metadata Property	Default Value	Allowed Values	Property Meaning
Enable PDF Export	Null	True, False	Enables export to PDF menu option in table actions

Programmatic Control

There is no programmatic control for this feature.

Unsupported Tables for Export (PDF and Excel)

- Tables rendered in the email facet.
- Tables rendered in printable pages.
- Inner Tables (table-in-table).
- Tables in pages running in embedded mode, such as tables inside a JTT skin.
- Tables in pages running in portlet mode.
- List of values table web beans.
- Attachment tables.
- Tables whose view object is dirty any time. Dirty View Objects indicate on going transactions hence export will disabled.
- Tables whose view object does not have any where clause or where clause params. These tables fall under blind query category (see Footnote)
- Tables whose view object binding style is Oracle Named and not Oracle Positional. As per the OA Framework Model Coding guidelines, the binding style for the table view object must be set to Oracle Positional.
- For guest users, we recommend turning off the profile option FND_ENABLE_BNE_TABLE_INTEGRATION.

Footnote: There are a few cases when the view object associated with the table will not have a where clause or any where clause parameters. The database tables may be populated at runtime. In certain cases, even with a blind query from the UI, the number of rows the user sees does not exceed a certain reasonable limit. To enable OA Framework to skip the blind query check in such cases, the developer can set the property Enable PDF Export to 'True', which causes OA Framework to skip the the blind query check.

Export Data Limit

OA Framework will export records up to the limit controlled by the profile option 'FND: View Object Max Fetch Size' / VO_MAX_FETCH_SIZE. There is currently no option to export all rows as such.

Personalization Considerations

Branding Image

Every exported PDF document should carry the same branding image in the base page. Since the default Oracle Theme in the base page is dark blue, it requires lighter corporate branding. However, the exported PDF document comes with a lighter background. If you would like to apply a darker corporate branding, you can set the profile option 'FND: Export to PDF Corporate Branding' image location / FND_EXPORT_TO_PDF_CORP_BRANDING_IMG_LOC. This should point to the file name of the image under the $OA_MEDIA directory.

End User Options

The end user can choose the font size used to render PDF content. By default, OA Framework sets it to 10.

Related Information

- BLAF UI Guideline(s)
  - Export/Import Page Templates
  - Export/Import Flows
- Javadoc File(s)
  - `oracle.apps.fnd.framework.webui.beans.form.OAExportBean`
- Lesson(s)

Sample Code

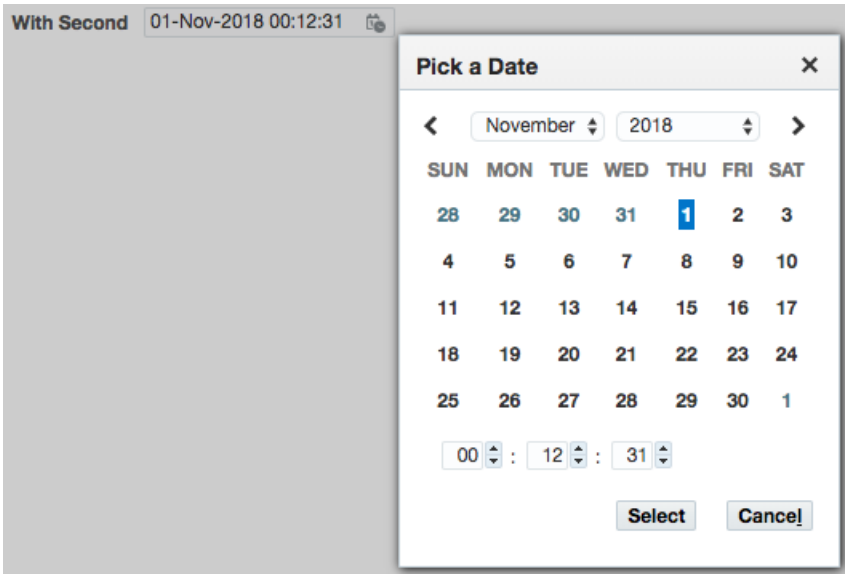
Date Picker

Overview

Oracle E-Business Suite users can enter a date for a date field by using a Date Picker. A Date Picker, as described further in the Oracle Browser Look-and-Feel (BLAF) UI Guidelines: Date Picker, is a feature that displays a graphical calendar from which users can select a date to populate a date field on a page. Including a Date Picker in a page means that not only can users easily select a date graphically, but clearly visible restrictions of available dates can be enforced for specific application contexts. A Date Picker can be implemented with or without an associated Time Picker.

If the time component is enabled, the popup displays with a Select button, clicking on which will close the popup.

Figure 1: Example of a Date Picker with the time component enabled.



If the time component is not enabled, this includes the situations of a Date Picker without the time component, and a Date Time Picker without the time component. In both these cases, the popup displays without a Select button: selecting a date or clicking on the side of the Date Picker will close the popup.

Figure 2: Example of a Date Picker without the time component enabled.

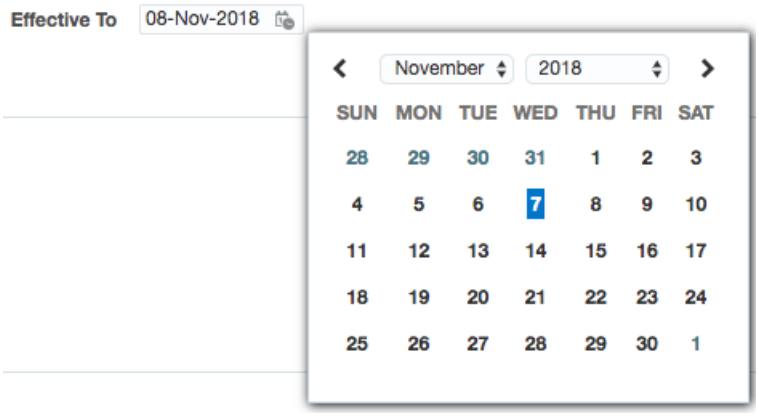
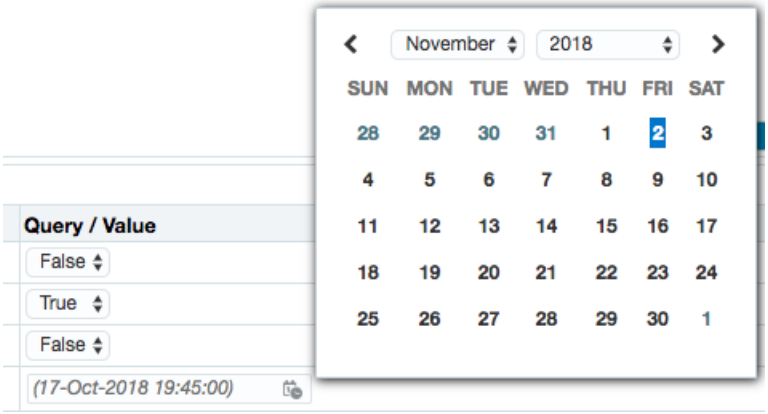


Figure 3: Example of a Date Time picker without the time component enabled.



Note: As of Oracle E-Business Suite Release 12.2.9, Date Pickers open in a OAPopupBean instead of the modal window that was used in previous releases.

Icon Access to Secondary Window

OA Framework automatically renders a Date Picker icon when you define a date field in a page.

Note OA Framework only supports the month and year as separate pulldown lists in the Date Picker.

Time Picker

As of Release 12.2.4, the Date Picker supports a time component that allows a user to specify a time while choosing a date. This time picking functionality is available for all supported Date Picker calendars.

The time component renders its additional time fields as spin boxes that enforce valid time choices in the 24 hour format, as shown in Figure 4. A user may choose any value between 00:00:00 to 23:59:59.

If a date field has no value for time, the launched date picker displays the default current system time (existing behavior) in the time fields. If a date field has a non-null value for time, the Date Picker's displays that time in its time fields.

Note: OA Framework does not support the time component for inline Date Pickers.

Declarative Implementation

To implement a Date Picker on your page so that it displays in a secondary window, simply create a **messageTextInput** item on your page and set its **Data Type** property to **Date**.

Note: The time component of the Date Picker may only be implemented programmatically.

Runtime Control

While there are no programmatic steps necessary to implement a Date Picker in a secondary window for a date field, if you wish to implement a Date Picker with a time component, you must programmatically invoke the `setTimePickerEnabled(true)` API in the class `oracle.apps.fnd.framework.webui.beans.message.OAMessageDateFieldBean` as shown in the code example below. Similarly if you wish to restrict the Date Picker to a range of dates, include code in the `processRequest` method of your controller as shown:

```
OAMessageDateFieldBean dateField = (OAMessageDateFieldBean)
  webBean.findIndexedChildRecursive("Date1");
  dateField.setMinValue(minDate);
  dateField.setMaxValue(maxDate);
  \\To enable the time component for Date Picker
  dateField.setTimePickerEnabled(true);
```

Personalization Considerations

- See a summary of Date Picker personalization considerations in the Oracle Application Framework Personalization Guide.

Inline Date Picker

When one or more date fields appear on a page, an Inline Date Picker can be associated with those date fields programmatically, allowing users to quickly select dates for those fields. An Inline Date Picker is displayed inline in the page contents. A user populates a date field by setting the focus on the desired date field and selecting a date from the Inline Date Picker.

Note There should be ONLY one Inline Date Picker on a page even if multiple date fields exist on the page.

Note OA Framework only supports the month and year as separate pulldown lists in the Date Picker.

Declarative Implementation

There is currently no declarative support for implementing an Inline Date Picker, however, you must define a date field on your page first by creating a **messageTextInput** item on your page and setting its **Data Type** property to **Date**.

Runtime Control

Creating an Inline Date Picker

Once you declaratively define one or more date fields on your page, you can programmatically create an Inline Date Picker and associate the Inline Date Picker ID with these date fields. You can also programmatically determine the placement of the Inline Date Picker on the page. To create an Inline Date Picker, include the following code in the processRequest method of your controller.

```
import oracle.apps.fnd.framework.webui.beans.form.OAInlineDatePickerBean;
import oracle.apps.fnd.framework.webui.beans.message.OAMessageDateFieldBean;


public void processRequest(OAPageContext pageContext, OAWebBean webBean)


{
   ...
   OAInlineDatePickerBean inlineDatePicker = (OAInlineDatePickerBean) createWebBean
     (pageContext, INLINE_DATEPICKER_BEAN, null, "DatePicker");
   inlineDatePicker.setID("DatePicker");
   OAMessageDateFieldBean dateField =
     (OAMessageDateFieldBean)webBean.findIndexedChildRecursive("Date1");
   dateField.setPickerId("DatePicker");
   // Set the same inlineDatePicker to another date field.
   dateField = (OAMessageDateFieldBean)webBean.findIndexedChildRecursive("Date2");
   dateField.setPickerId("DatePicker");

   webBean.addIndexedChild(inlineDatePicker);
}
```

Creating an Inline Date Picker with Max and Min Values

To display an Inline Date Picker in a page with a restricted range of dates, include the following code in the processRequest method of your controller.

```
import oracle.apps.fnd.framework.webui.beans.form.OAInlineDatePickerBean;
import oracle.apps.fnd.framework.webui.beans.message.OAMessageDateFieldBean
import java.util.Date;


public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   ...
   Date minDate = new Date(100, 06, 04); // 4th July 2000
   Date maxDate = new Date(104, 11, 17); // 17th December 2004


   OAInlineDatePickerBean inlineDatePicker = (OAInlineDatePickerBean) createWebBean
     (pageContext, INLINE_DATEPICKER_BEAN, null, "DatePicker");
   inlineDatePicker.setID("DatePicker");
   inlineDatePicker.setMinValue(minDate);
   inlineDatePicker.setMaxValue(maxDate);
   OAMessageDateFieldBean dateField =
     (OAMessageDateFieldBean)webBean.findIndexedChildRecursive("Date1");
   dateField.setPickerId("DatePicker");
   // Should be set on the date field also.
   dateField.setMinValue(minDate);
   dateField.setMaxValue(maxDate);

   webBean.addIndexedChild(inlineDatePicker);
}
```

Personalization Considerations

- See a summary of Date Picker personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guideline
  - Date Picker
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.form.OAInlineDatePickerBean`
  - `oracle.apps.fnd.framework.webui.beans.form.OADateFieldBean`

Declarative Page Flow

Overview

With the OA Framework you can declaratively define navigation relationships between pages using the following techniques. You can either:

- Specify the personalizable Destination Function property for select UI components in a page or
- Use Oracle Workflow to define a workflow that describes your navigation rules for a page flow, and then leverage this workflow at runtime to dynamically ascertain the target page for each navigation action

With the "Destination Function" technique, you can associate a target navigation page with a UI component at design time. So, for example, if the user selects a particular link or button, you can navigate to the page whose UI function is associated with that link or button. Since the `Destination Function` UI property is personalizable, this approach lets customers easily change the target page for a given action. This technique is best used for simple, point-to-point navigation that can be statically defined.

With the Workflow technique, you can model a complex page flow including forward/back navigation, looping and conditional branching. Each time the user tries to perform a navigation action, the workflow determines what the target page should be. Generally, the workflow technique is best suited to multistep transactions. In this case, customers can't simply change the logic that interrogates the workflow to obtain the destination page, but they can change the workflow definition itself if necessary.

Destination Function Page Flow

To define a navigation action that can be personalized by administrators, you simply need to set the `Destination Function` property value to the name of a UI function whose `Web HTML` call points to the target page (see the Tabs/Navigation document for information about creating functions). Customers can then create a function-based personalization for this page, for example, and substitute their new function for the one that you seed.

The `Destination Function` property is supported for a select list of components. The corresponding behavior differs slightly based on whether the component renders as a link or a form submit item.

Note: The `Destination Function` property is not yet supported on the **navigationBar** (`oracle.apps.fnd.framework.webui.beans.nav.OANavigationBarBean`) so it cannot be used with this component for step-by-step transaction flows.

GET Implementation

If you set the `Destination Function` property on a bean that normally renders as a link, the `Destination URI` property is ignored (if specified). When the user selects the link, a GET request is issued and the page associated with the target function displays. The list of beans in this category include:

Note that this functionality existed previously for some beans.

- button
- image
- link
- nodeDef (in a tree)
- staticStyledText
- messageStyledText

Tip: If you want to add parameters to the URL generated for the GET request, you can use the `setInvokeFunctionParams(String functionParam)` method on your bean. For example, you could add the following `processRequest()` logic to add the request parameter `foo=bar` to the URL when a button with a `Destination Function` is selected:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);


  OAButtonBean button =
     (OAButtonBean)webBean.findIndexedChildRecursive("<myButton>");

  button.setInvokeFunctionParams("foo=bar");

}
```

POST Implementation

If you set the `Destination Function` property on a bean that normally submits the form when selected (a **submitButton**, for example, or another component configured to submit the form with a fireAction), setting this property does not change this behavior. You must handle the form submission event as you normally would with the code shown below.

The list of beans in this category include:

- button
- image
- link
- nodeDef (in a tree)
- staticStyledText
- messageStyledText
- singleSelection
- messageRadioGroup
- messageCheckBox
- messageTextInput
- messageChoice
- messageLovChoice
- messageRadioButton
- submitButton

If you set the `Destination Function` for one of these beans, you must code `processFormRequest()` logic to obtain the function name so you can pass it to a `setForward*URL()` method call. This is illustrated in the following example that handles the `fireAction` event for a **messageChoice** bean.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean  webBean)
{
 super.processFormRequest(pageContext, webBean);
 String event = pageContext.getParameter(EVENT_PARAM);
 String source = pageContext.getParameter(SOURCE_PARAM);

 if ( "<myEventName>".equals(event) && "<myPoplist>".equals(source) )
 {
    OAMessageChoiceBean poplistBean  =
     (OAMessageChoiceBean)webBean.findIndexedChildRecursive("<myPoplist>");

    String invokeFunc = poplistBean.getInvokeFunctionName();
    pageContext.setForwardURL(invokeFunc,...);
  }
}
```

Personalization Considerations

- See a summary of Page Flow personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

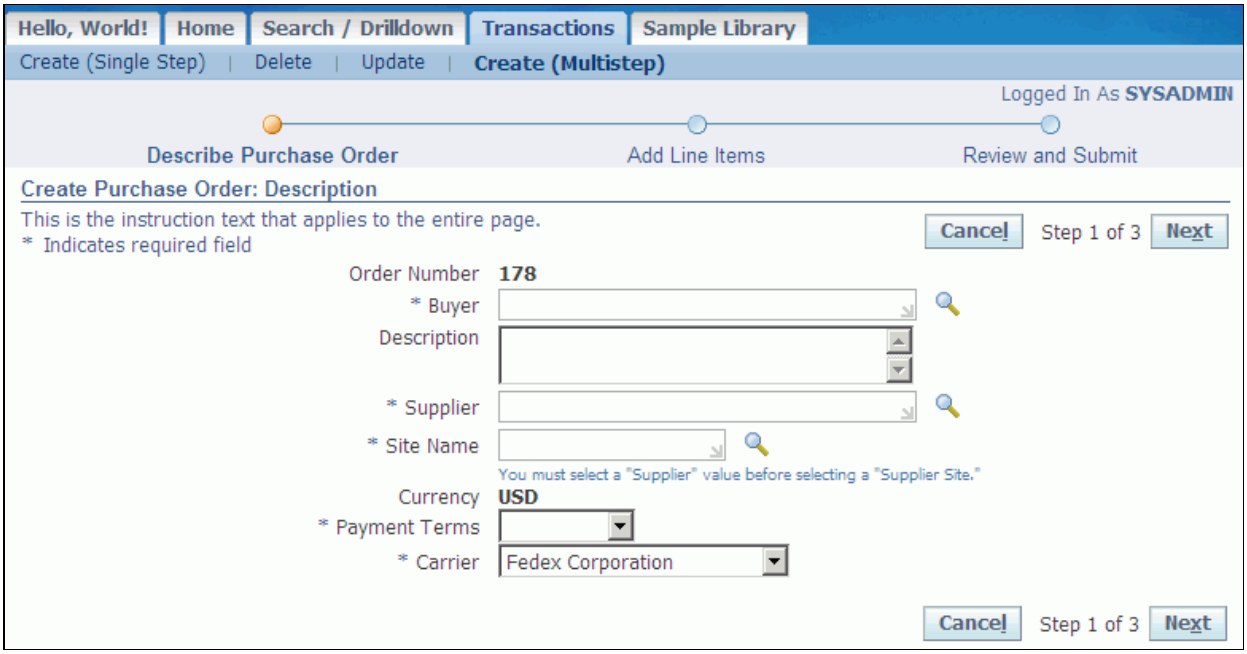
- None

Workflow Page Flow

Instead of adding complex, static logic to your page controllers to handle different navigation scenarios, this technique lets you leverage Oracle Workflow to handle all the conditional navigation rules. This section describes how to create an appropriate workflow definition for this purpose, and how to interact with it as the user navigates through the transaction.

For additional information about the Workflow product, see the *Oracle Workflow Developer's Guide*, the *Oracle Workflow Administrator's Guide*, the *Oracle Workflow User's Guide* and the *Oracle Workflow API Reference*. These instructions assume that you understand how to create, test and deploy Workflow process definitions; it focuses exclusively on the characteristics of a workflow that are important to this particular use case.

Figure 1: Example of the first page in a multistep flow (this is the Multistep Create flow in the OA Framework ToolBox Tutorial which uses Workflow).



Implementation

To create a multistep page flow using Workflow, follow these steps:

Step 1: Create Your Transaction Pages

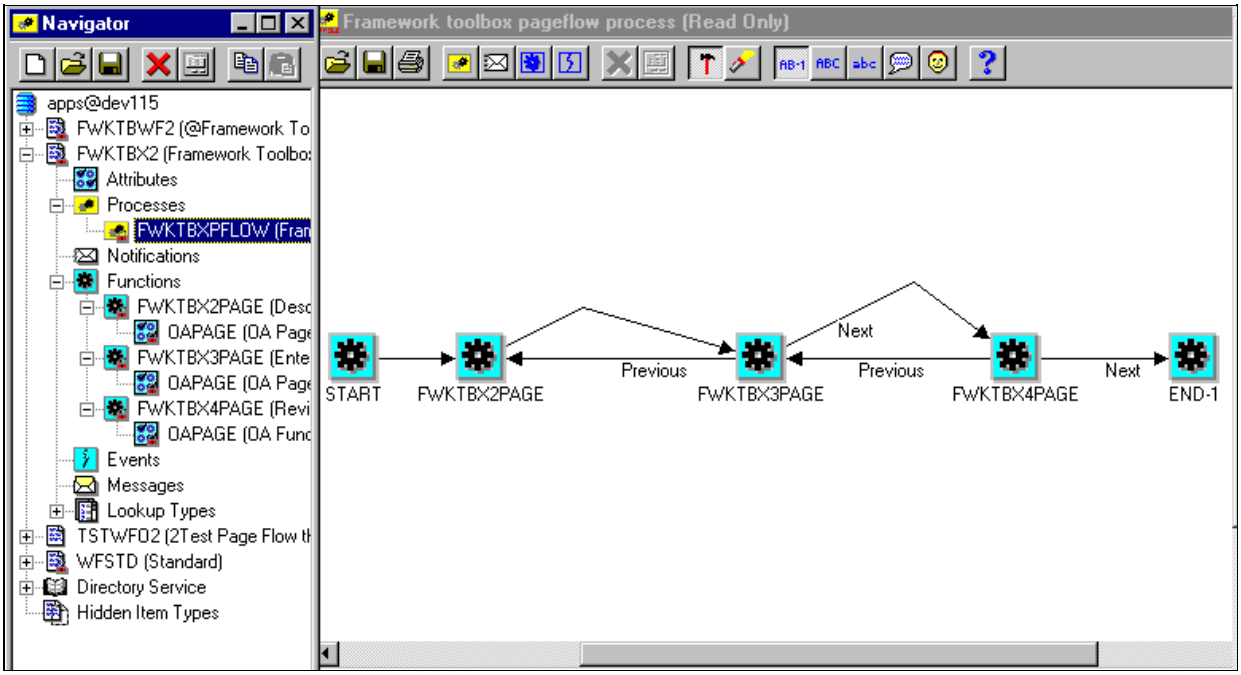
Create all the pages that need to participate in a multipage transaction, as you would create any OA Framework page.

Step 2: Define the Page Flow Workflow

Use Oracle's Workflow Builder to define your page flow in accordance with the rules and guidelines described below.

Note that this document assumes that you know how to use the Workflow Builder. For general information on defining workflow processes refer to the *Oracle Workflow Developer's Guide*.

Figure 2: Example of a Page Flow in Oracle Workflow Builder.



To configure a workflow as a page flow:

1. Add a function activity for each page in the transaction. For each activity, decide whether to:

- Associate a result with this activity (if appropriate for conditional navigation), or
- Explicitly mark the activity as blocked by associating the `wf_standard.block` PL/SQL function with it.

  When the Workflow Engine encounters a blocking activity it stops and waits for some sub-process or external entity to provide the information it needs to proceed. So, with the mapping between transaction pages and the blocking activities in the Workflow, you can query the activity data to find out what page ought to be rendered.

2. Add FORM attribute to each page-related blocking activity. You will use this for page information as shown in Figure 4 below.

Figure 3: Function definition in Oracle Workflow

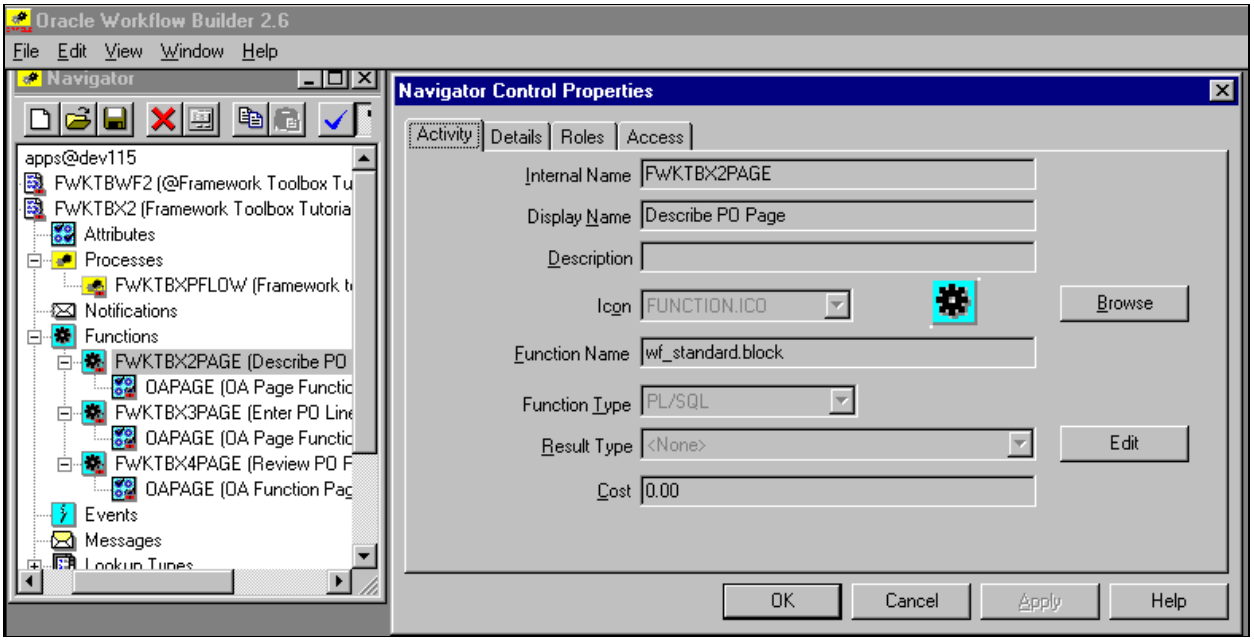
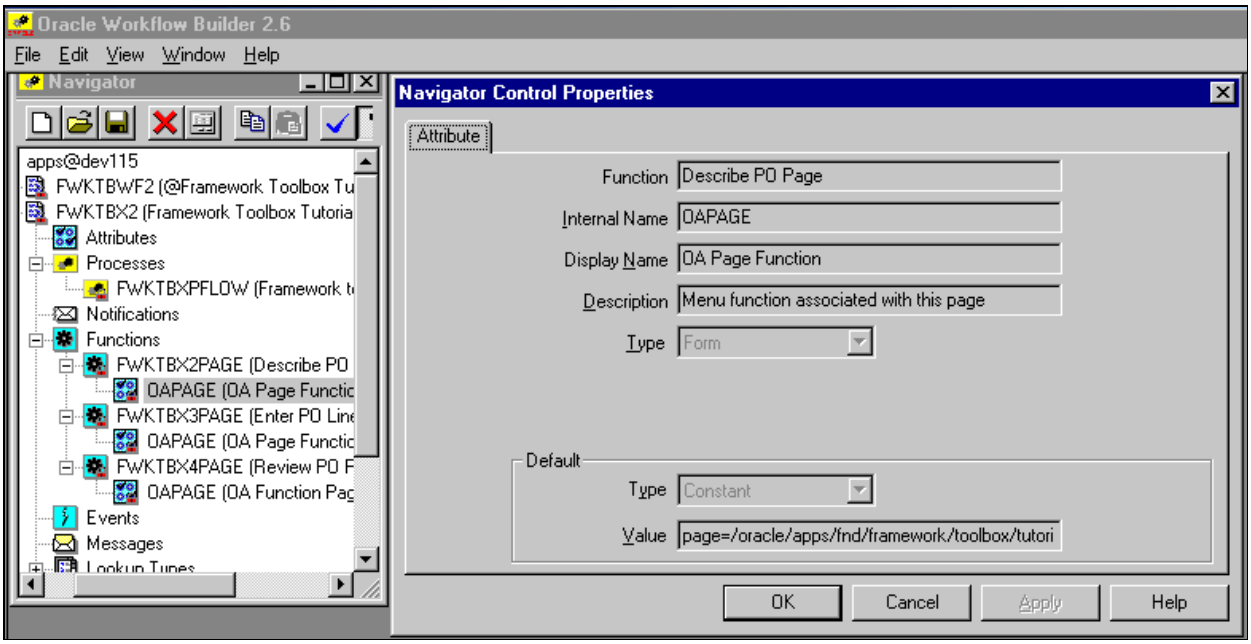


Figure 6: Function activity attribute with page information.



You can also add blocking activities that do not correspond to a page for the following use cases:

- You send a notification at the end of the transaction.

  For example, a Create Expense Report flow might send out a notification asking for manager approval. Once the manager responds, you want to send an FYI notification to the user who submitted the expense report. In this case, the blocking notification activity comes after the multiple page part of the transaction has concluded.

- The workflow is comprised of two discrete transactions as described in this user flow:

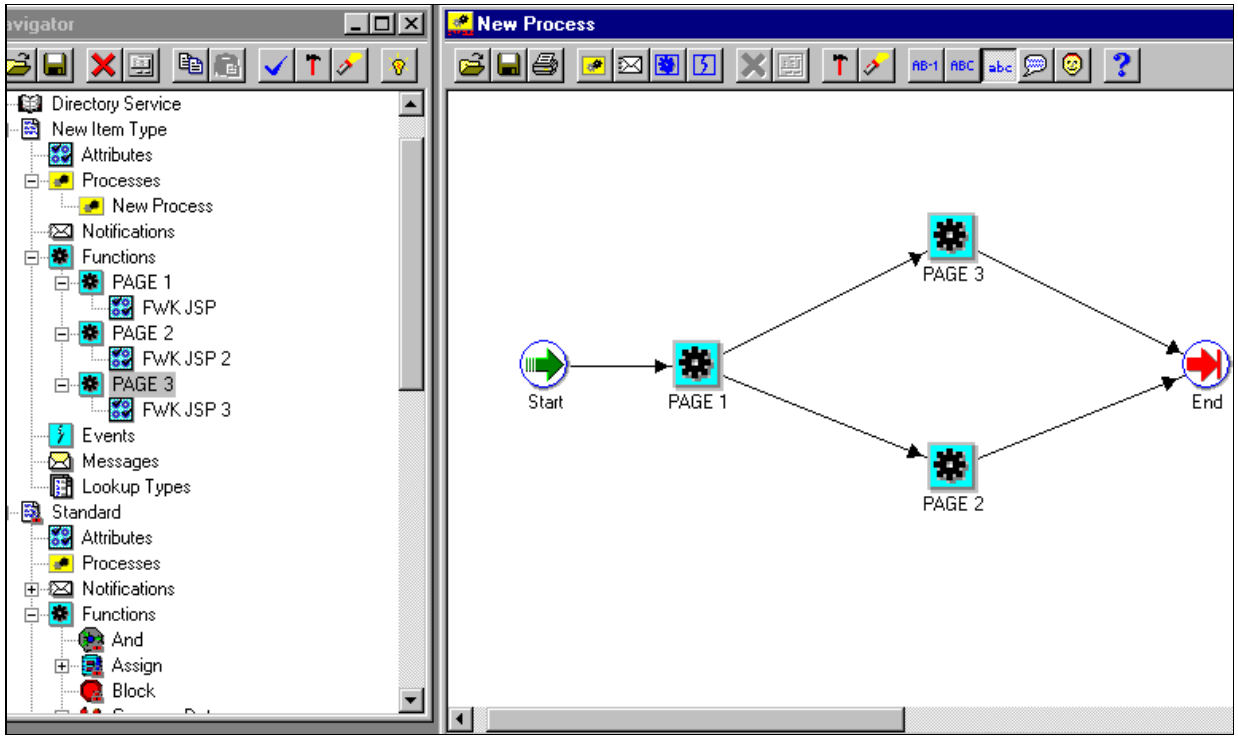
    1. User selects a "Create Expense Report" action.
    2. User fills out Page 1, Page 2, Page 3 to create and submit an expense report (*transaction #1*)
    3. Report submission causes a notification to the user's manager.
    4. The manager can approve or reject the expense request.
    5. If the manager rejects the expense, an FYI Notification is sent to the user asking him to update the expense report.
    6. User selects an "Update Expense Report" action.
    7. This time user steps through Page 4 and Page 5 to update and resubmit the expense report *(transaction #2*)

    In the corresponding workflow process definition, the two transactions are separated by a notification that requires a user response. The second part of the create expense report transaction is optional and is executed only if the expense report is rejected. The point here is that blocking activities that do not correspond to pages should only be inserted at appropriate points in the flow, otherwise the user won't able to finish the transaction.

    Note that in this second scenario your application has to do all the work to make sure that when the user returns to update the expense report, he is in the right context. Your application is also responsible for calling the correct OA Framework API to indicate that the page flow depends on a previously started workflow process

Workflow activities that denote pages in a flow should not have OR joins. The following scenario won't work. In this flow, Page 1 can go to Page 2 or Page 3. Since there is no result type associated with Page 1, the Workflow Engine will randomly decide between Page 2 or Page 3. You can change this into a valid flow by attaching a result type to Page 2.

Figure 7: Invalid workflow with an OR.



Step 3: Start the Page Flow

There are two ways you can start a Workflow process for a page flow based on whether your page flow is launched from an originating page or not (for example, the originating page might be a read-only summary with a "Create Object" button that launched the page flow).

In the scenario where you have a launching page, your `processFormRequest()` logic for the button that launches the page flow should appear as follows:

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
```

```
  ...

  if (OANavigation.isWorkflowInProgress(pageContext))
  {
    String oldKey = OANavigation.getItemKey(pageContext);

    if (!OANavigation.abortProcess(pageContext, "FWKTBX2","FWKTBXPFLOW", oldKey, null, null))
    {
       ...throw exception
     }
  }

  OAApplicationModule am = pageContext.getRootApplicationModule();

  String wfKey = null;
  String nextPage = null;

  // Initialize the Workflow pageflow.

  Serializable[] parameters = { "fwk_tbx_wf_seq" };
  wfKey = (String)am.invokeMethod("getWfKey", parameters);

  OANavigation.createProcess(pageContext, "FWKTBX2","FWKTBXPFLOW", wfKey);
  OANavigation.startProcess(pageContext, "FWKTBX2", "FWKTBXPFLOW",wfKey);

  // This is the correct signature for initializing a new workflow when you're going
  // to transition to the first activity.
  nextPage = OANavigation.getNextPage(pageContext, "FWKTBX2", wfKey,null, false);

  HashMap params = new HashMap(1);

  params.put("poStep", "1");

  pageContext.setForwardURL(pageContext.getApplicationJSP() + "?" + nextPage, // target page
       null,
       KEEP_MENU_CONTEXT,
       "", // No need to specify since we're keeping menu context
       params, // Page parameters
       true, // Be sure to retain the AM!
       ADD_BREAD_CRUMB_NO, // Do not display breadcrumbs
       OAException.ERROR); // Do not forward w/ errors

} // end processFormRequest()
```

If you don't want to display a generic page and instead would like to start with the first page defined in the Workflow, your page controller must instantiate the workflow, query it for page information, and then redirect the request to this page as shown:

```
public void processRequest(OAPageContext  pageContext, OAWebBean webBean)
{
   ...


   String wfKey = null;
   String nextPage = null;

   if (!OANavigation.isWorkflowInProgress(pageContext))
   {
     OAApplicationModule am = pageContext.getRootApplicationModule();

     // Initialize the Workflow pageflow.
     Serializable[] parameters = { "fwk_tbx_wf_seq" };
     wfKey = (String)am.invokeMethod("getWfKey", parameters);

     OANavigation.createProcess(pageContext, "FWKTBX2", "FWKTBXPFLOW", wfKey);
     OANavigation.startProcess(pageContext, "FWKTBX2", "FWKTBXPFLOW", wfKey);

   }

   nextPage = OANavigation.getNextPage(pageContext, "FWKTBX2", wfKey, null, false);

   HashMap params = new HashMap(1);

   params.put("poStep", "1");
```

```
    pageContext.setForwardURL(pageContext.getApplicationJSP() + "?" +    nextPage, // target page
       null,
       KEEP_MENU_CONTEXT,
      "", // No need to specify since we're keeping menu context
      params, // Page parameters
      true, // Be sure to retain the AM!
      ADD_BREAD_CRUMB_NO, // Do not display breadcrumbs
      OAException.ERROR); // Do not forward w/ errors

  }
}
```

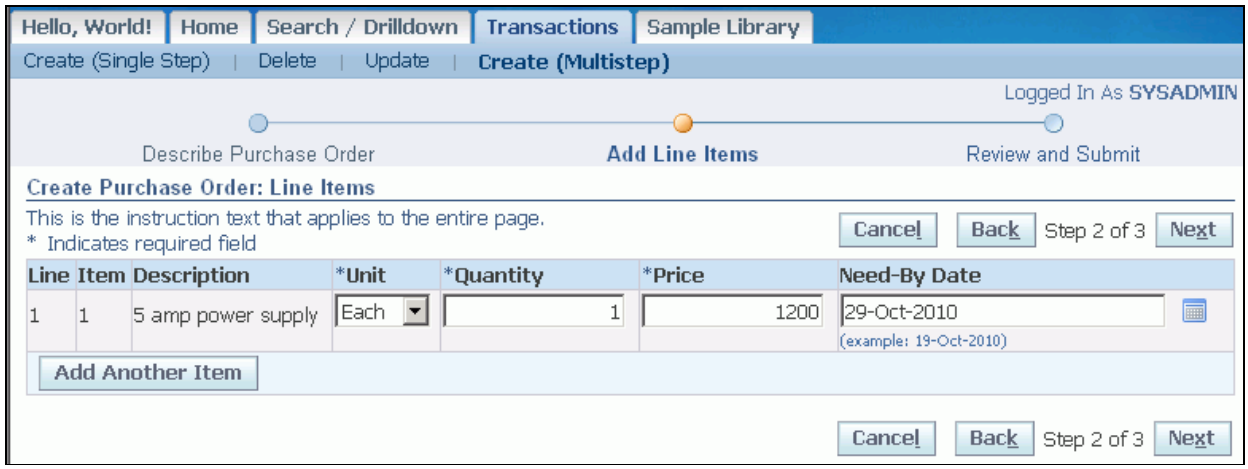
Step 4: Transition Between Pages

Note: Please see the Multistep Create flow in the OA Framework ToolBox Tutorial for a complete code example.

The `OANavigation` API provides three `getNextPage()` methods for transitioning the workflow.

- `getNextPage(OAPageContext pageContext)` - transitions to the next page of the workflow.
- `getNextPage(OAPageContext pageContext, String resultCode)` - transitions to the next page of the workflow based on the result type determined in your controller.
- `getNextPage(OAPageContext pageContext, String wfItemType, String wfItemKey, String wfProcess, boolean initialize)` - used to resume a saved transaction.

Figure 8: Second step of the purchase order.



To transition to the next page of an active transaction of the workflow, use `OANavigation.getNextPage(OAPageContext pageContext)` in your `proceessFormRequest()` method. The `getNextPage()` completes the blocking activity corresponding to the current page and advances the workflow until it encounters the next blocking activity. The workflow transitions to a waiting state and control is transferred back to the calling `getNextPage()`.

```
nextPage = OANavigation.getNextPage(pageContext);
```

The method then queries the blocking activity and retrieves the page information associated with the activity's FORM type attribute and returns it to the `processFormRequest()` of the calling controller. You should write a `setForwardURL` to redirect the request to next page in flow.

```
HashMap params = .....;
params.put(.....);
pageContext.setForwardURL(........);
```

To transition to the next page as per the result type returned by the OA Framework controller use `getNextPage(OAPageContext pageContext, String resultcode)`.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  ...
  String workflowResultCode = null;

  if (pageContext.getParameter("Next") != null)
  {
    workflowResultCode = "Next";
  }
```

```
  else if (pageContext.getParameter("Previous") != null)
  {
    workflowResultCode = "Previous";
  }

  nextPage = OANavigation.getNextPage(pageContext,workflowResultCode);
  HashMap params = ....
  params.put(....);
  pageContext.setForwardURL(.........);
}
```

Step 5: Clear the Workflow Context

Since the Workflow transaction holds an independent local JDBC connection, you must override the `OADBTransaction.beforePoolCheckin()` method in your root UI application module and call `getDBTransaction.clearWorkflowInfo()` to release the connection.

```
public void beforePoolCheckin()
{
   // Always call this first.
   super.beforePoolCheckin(...);
   getDBTransaction.clearWorkflowInfo();

}
```

Saving the Transaction and Continuing Later

OA Framework does not provide built-in support for "Save for Later" transactions. If you want to provide this feature, you must implement all the state management yourself. To resume a saved transaction, you must query a blocked workflow to retrieve the current page information by calling `getNextPage(OAPageContext pageContext, String wfItemType, String wfItemKey, String wfProcess, Boolean initialize)`.

Browser Back Button Support

Workflow-based page flows automatically support browser Back button navigation at the Workflow technology layer. Whenever user moves between the pages using browser Back button and resubmits, the workflow rewinds itself to the appropriate blocking activity so it stays in synch with the user's current navigation position.

See Supporting the Browser Back Button for additional information as it relates to the explicit Back button support required at the OA Framework layer.

Page Flows with Train and Navigation Bar

If the number of page flow steps is known, it is appropriate to use a train and associated navigation bar as shown in Figure 1 above (if the number of steps is indeterminate, this is not an appropriate UI design). For implementation instructions, see Locator Element: Train and Locator Element: Page Navigation in Chapter 4.

Error Handling

For the most part, you handle application errors in your Workflow-based page flows the same way that you do in any other context. That said, however, you need to consider the possibility that the user might fix mistakes and resubmit the form whenever you introduce workflow code.

Known Issues

- See a summary of key declarative page flow issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
  - Step-by-Step Page Flows
  - Step-by-Step (3+ Step) Page Template
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.nav.OANavigationBarBean`
- OA Framework Developer's Guide
  - Tabs / Navigation
  - Submitting the Form
  - Locator Element: Train
  - Locator Element: Page / Record Navigation
- OA Framework ToolBox Tutorial / Sample Library

    - See the Create (Multistep) example in the Tutorial.jpr (run the `test_fwktutorial.jsp` to try it)
- Oracle Workflow Documentation
    - Oracle Workflow Developer's Guide
    - Oracle Workflow Administrator's Guide
    - Oracle Workflow User's Guide
    - Oracle Workflow API Reference

Default Favorites

Overview

This feature enables administrators to offer favorite options for users from the *Default Favorites* page, which are then suggested to the user via a popup on the simplified Home Page. The popup also contains suggestions about popular functions used by other users, and it can be invoked from the Add Favorites icon on the simplified Home Page.

In other words, the Default Favorites feature offers a way to suggest a standard set of favorites out of the box, so that all users will see the same set of links on the simplified Home Page when they log in. These favorites can then be used as convenient shortcuts.

Using the Default Favorites Page

A new page, Favorites, has been added to the functional administrator responsibility under the Personalization menu. This page is similar to the Manage Favorites page, and loads with all available responsibilities..

Administrators can search for and select functions, and add them to the adjacent table by clicking on 'Add to Global List'. A column for count has been added to the results table, so that administrators can identify the popularly used functions in that responsibility. Administrators can also view the number of users who currently have that function in their Favorites list.

The selected functions are added to the lower 'Active Default Favorites' table. Clicking on 'Save' saves the changes to the database. Clicking on 'Apply' propagates and adds these functions to the relevant users' favorites lists if the agent listener is running.

Default Favorites Popup

The default favorites popup can be launched from the Add Favorites icon on the simplified Home Page. The first accordion contains the suggestions (if any) from the admin. The second accordion contains top bookmarked functions by other users as tiles. Springboard items are used in this case. The springboard item component has been enhanced to support a rating bar with a star to show selection. The springboard items are embedded into masonry tiles to use a masonry layout inside the accordion. This prevents scrolling inside the popup, and provides pagination support.

Select All/None can be used for selection. Users can select functions from both accordions and click on Add. Clicking on Cancel or the X on the popup will close the popup. The simplified Home Page reloads with the newly added favorites.

The popup can only be launched from the Add Favorites icon on the simplified Home Page. On reopening the popup, already added functions are removed from the suggestion list.

User can go to the Manage Favorites page by clicking on the Manage Favorites link at the bottom left of the popup.

If the user does not select anything and clicks on 'Add', this displays a confirmation message on dialog page. Clicking on 'No' reloads the popup. Clicking on 'Yes' closes the popup and reloads the Home Page.

UI Guidelines

This feature is used for to make favorites available for users by the administrator, as suggested to the users via a popup and preset in the case of the first logins by new users.

Associated Profile Options

No new or existing profile options are applicable to this feature.

Developer/System Administrator Procedures

Default Favorites feature (available to administrators)

1. Go to Functional Administrator responsibility, Personalization menu, Favorites tab. This is the Manage Default Favorites page.
2. The responsibility pop list loads with all active responsibilities. Choose the responsibility and click on Go. Select functions and add to the table below by clicking on 'Add to global List' button.For this purpose another column for count has been added to the result table so as to help the admin decide/select the popularly used functions in that responsibility. It gives us the no of users who currently have that function in their favorite list.
3. Click Save and Apply.

Favorite Suggestion Popup (available to all users from simplified home page only)

1. The suggestion popup can be launched from the 'Add Favorites' icon on the Home Page.
2. The popup launches with 2 sections. The first accordion contains the suggestions from the admin (if any) and the second accordion contains top bookmarked functions by other users. Users can select functions from both accordions and click on

Add. Select All/None can also be used for selection
3. The simplified homepage reloads with the newly added favorites.
4. On reopening the popup, already added functions are removed from the suggestion list.
5. Additional popup fetaures:
6. User can go to the manage favorite page by clicking on the Manage Favorite link at the bottom left side of the popup.
7. If the user does not select anything and clicks on 'Add' we see a confirmation message on dialog page. Clicking on No reloads the popup. Clicking on Yes closes the popup and reloads the homepage.
8. Clicking on Cancel and X icon closes the popup.

Administrator Options

Administrators should be able to browse through all active responsibilities in the instance and select the frequently used functions to be added to the suggestion list. This feature should be available to all users who have administrator responsibilities via the functional administrator responsibility.

Count shown beside every function

A new column in the table illustrates how popular a function is by showing the number of users bookmarking it. This is helpful to administrators in selecting the most used functions from a responsibility.

Creation of popup to show administrator suggested functions

A suggestion popup is available to all users to help them select the functions they feel are most appropriate to them. This gives them a choice rather than forcing use of the administrator-selected functions.

Key features of the popup include:

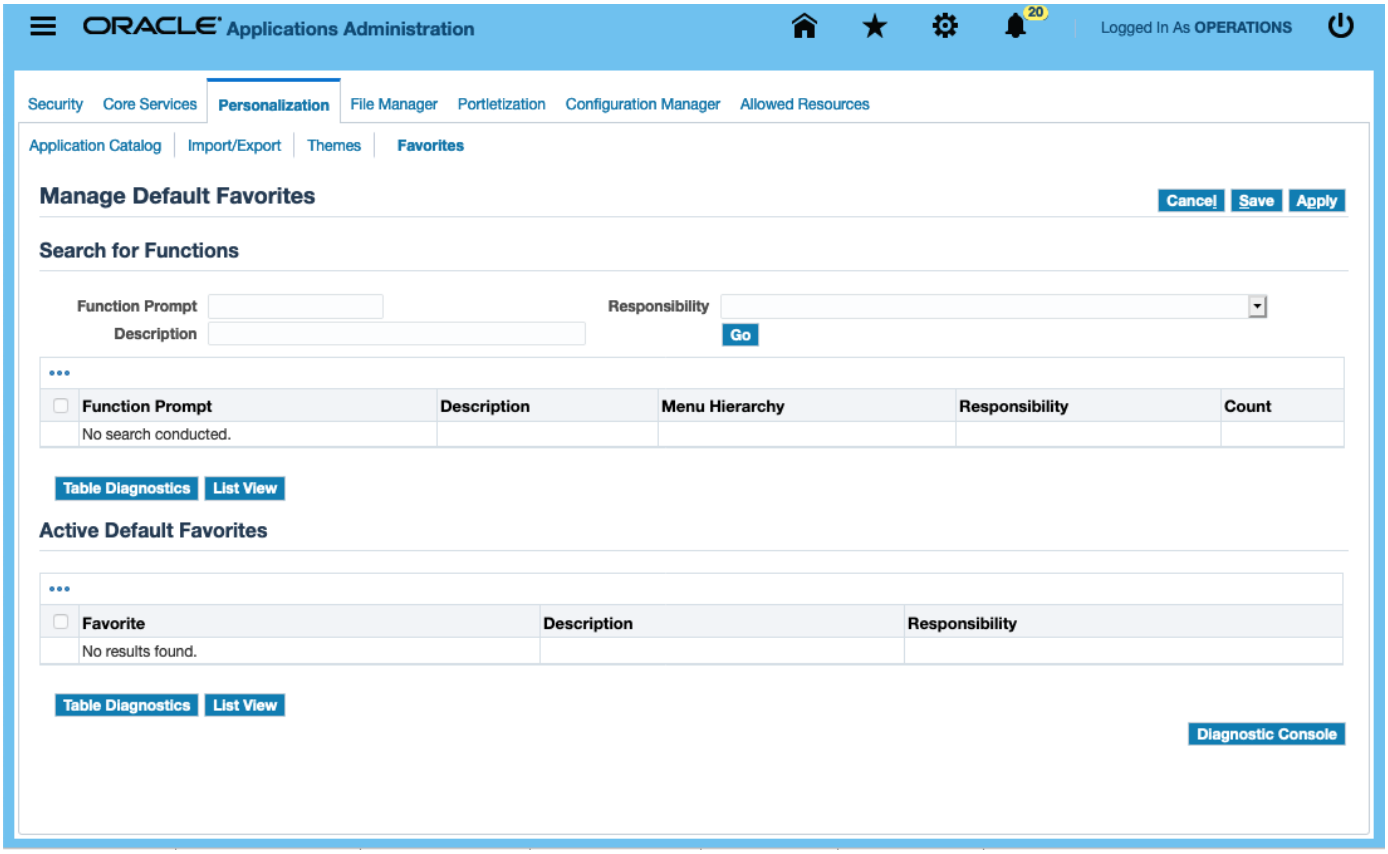
- The function list on the popup filters out any already-added functions in the user's list. On adding of the selected functions to the user's list, those functions are removed from the suggestion popup.
- This popup is launched from the 'Add Favorites' button on the simple Home Page when there are no favorites added for the user and the Home Page is empty.
- The popup has a link to take the user's Manage Favorites page. If there are no suggestions available, the user is taken directly to his Manage Favorites page.
- The popup also shows a list of all popular functions in decreasing order of popularity. Popularity is determined by the number of users having that function in their favorites list. Functions with zero users are not included.

Known Issues

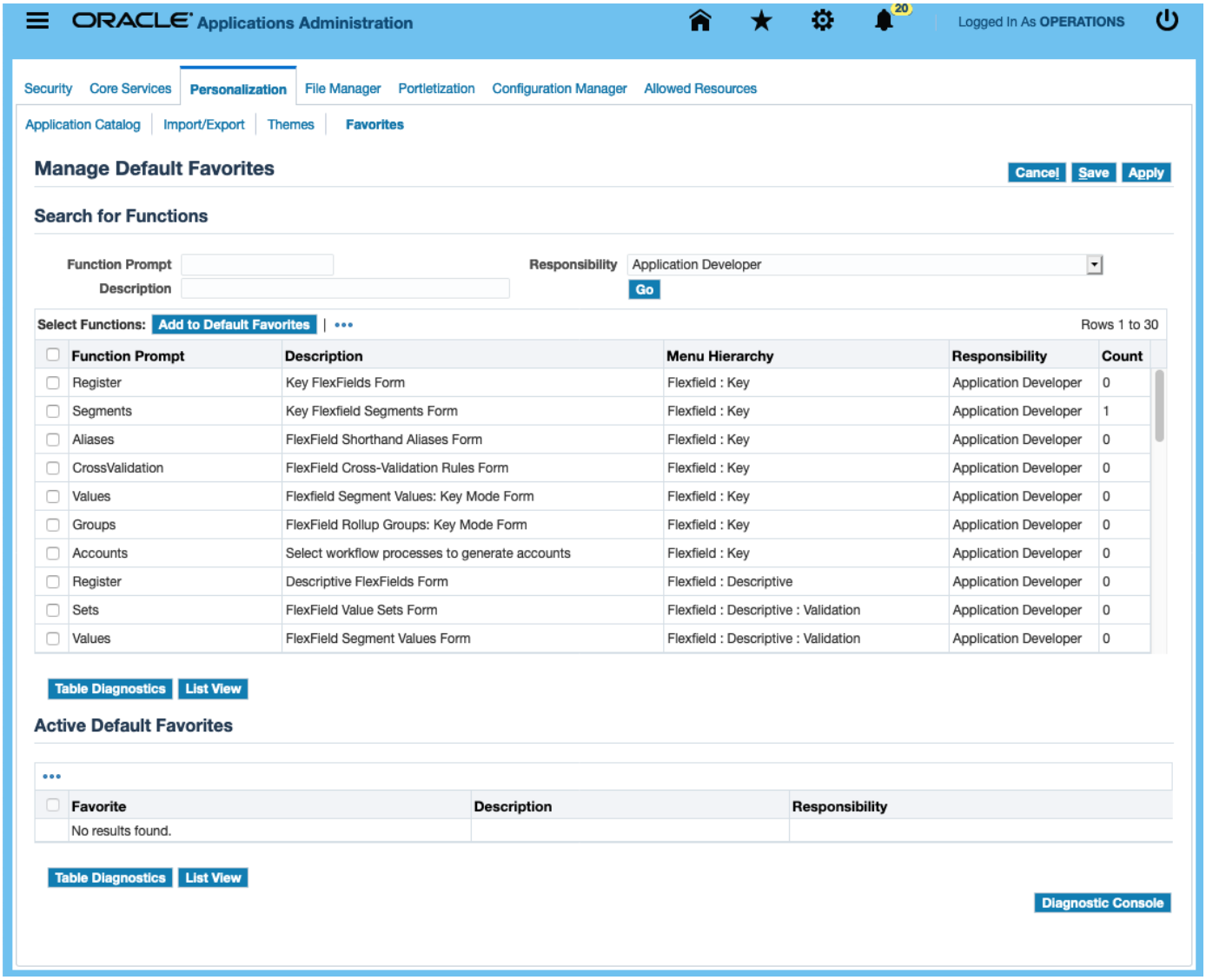
- The favorites suggestion popup can be launched only from the simplified Home Page.

Example Screens

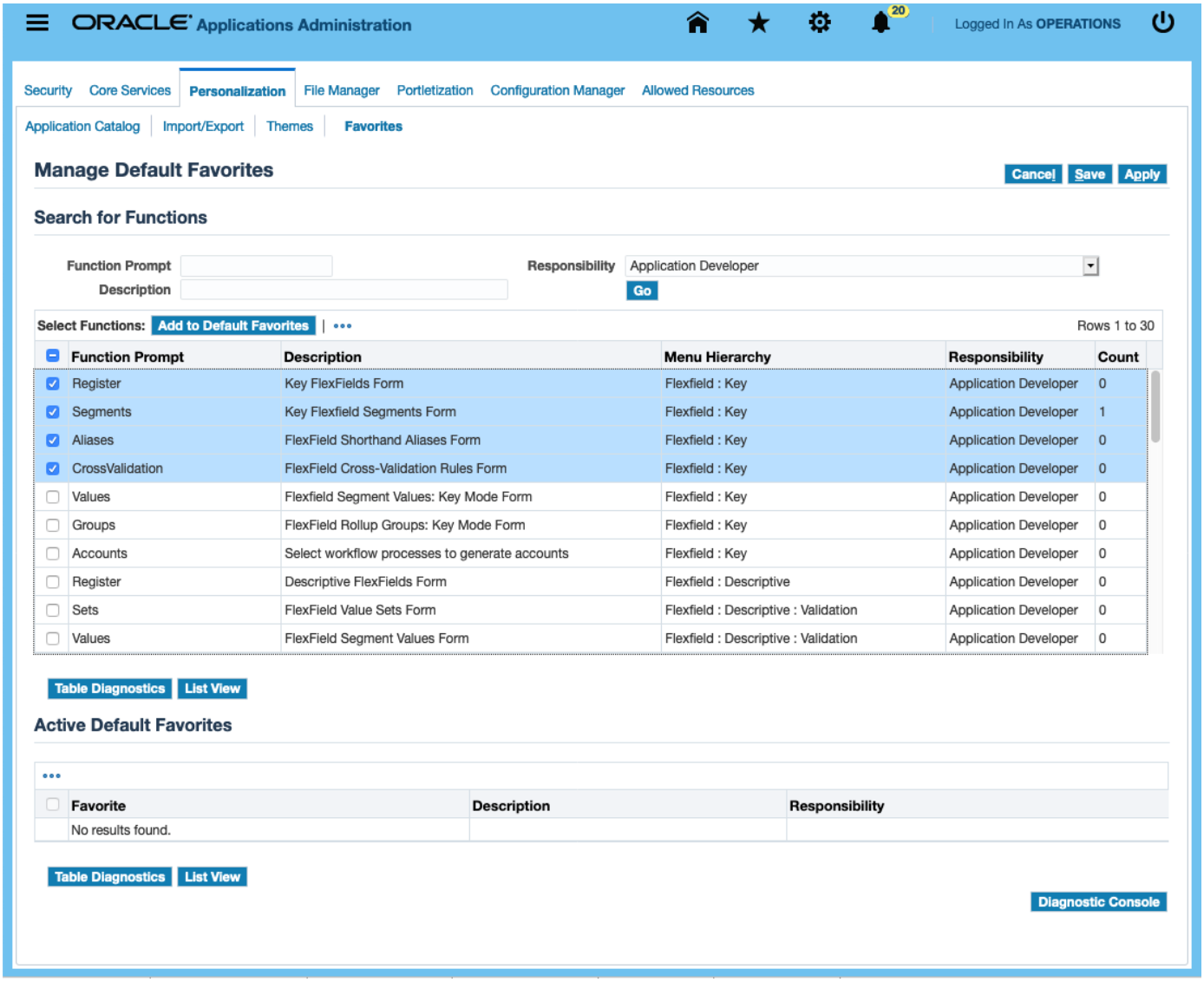
Personalization tab with Favorites subtab, and Manage Default Favorites selected:



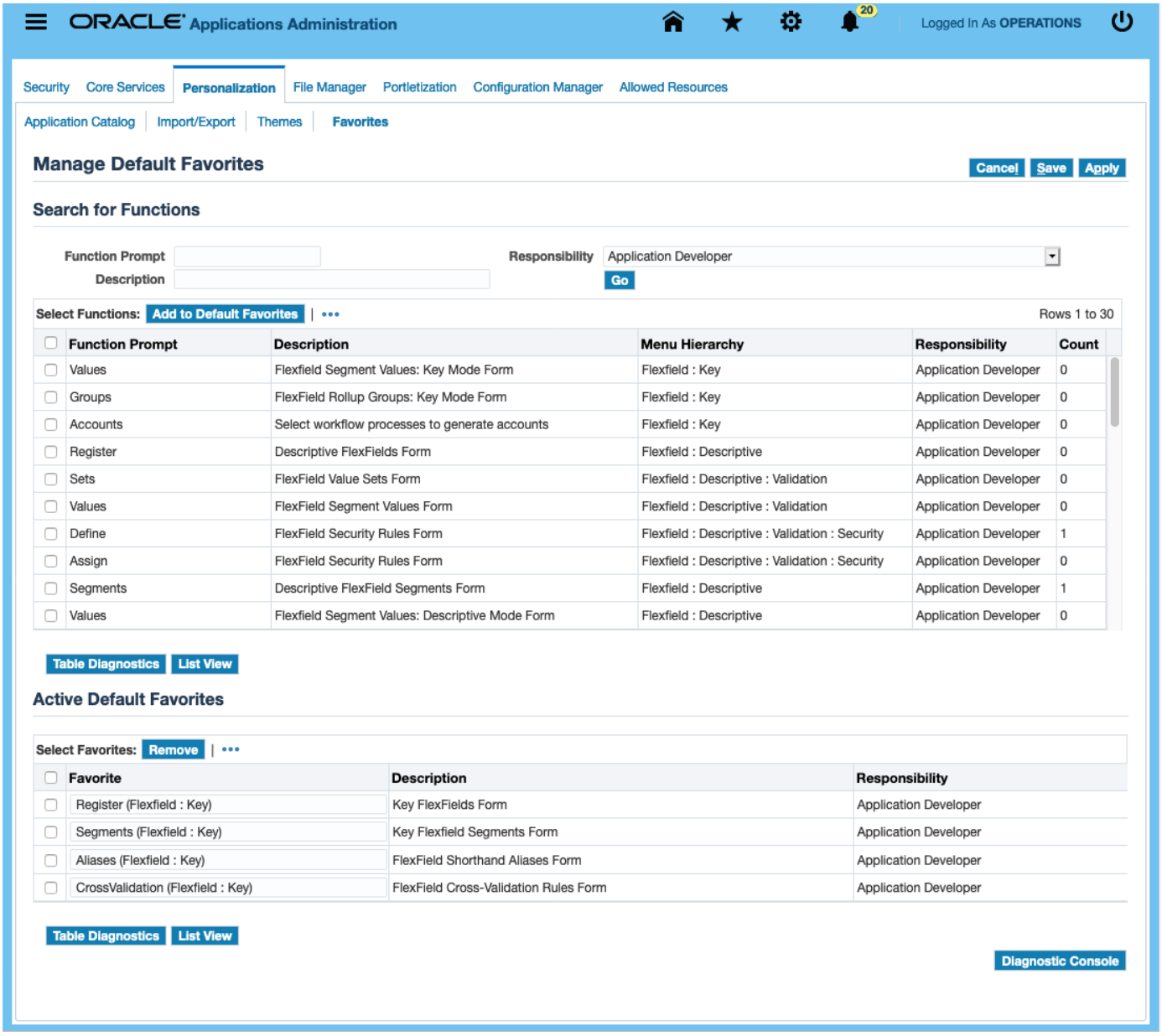
Example of choosing Application Developer responsibility in readiness for selecting Default Favorites:



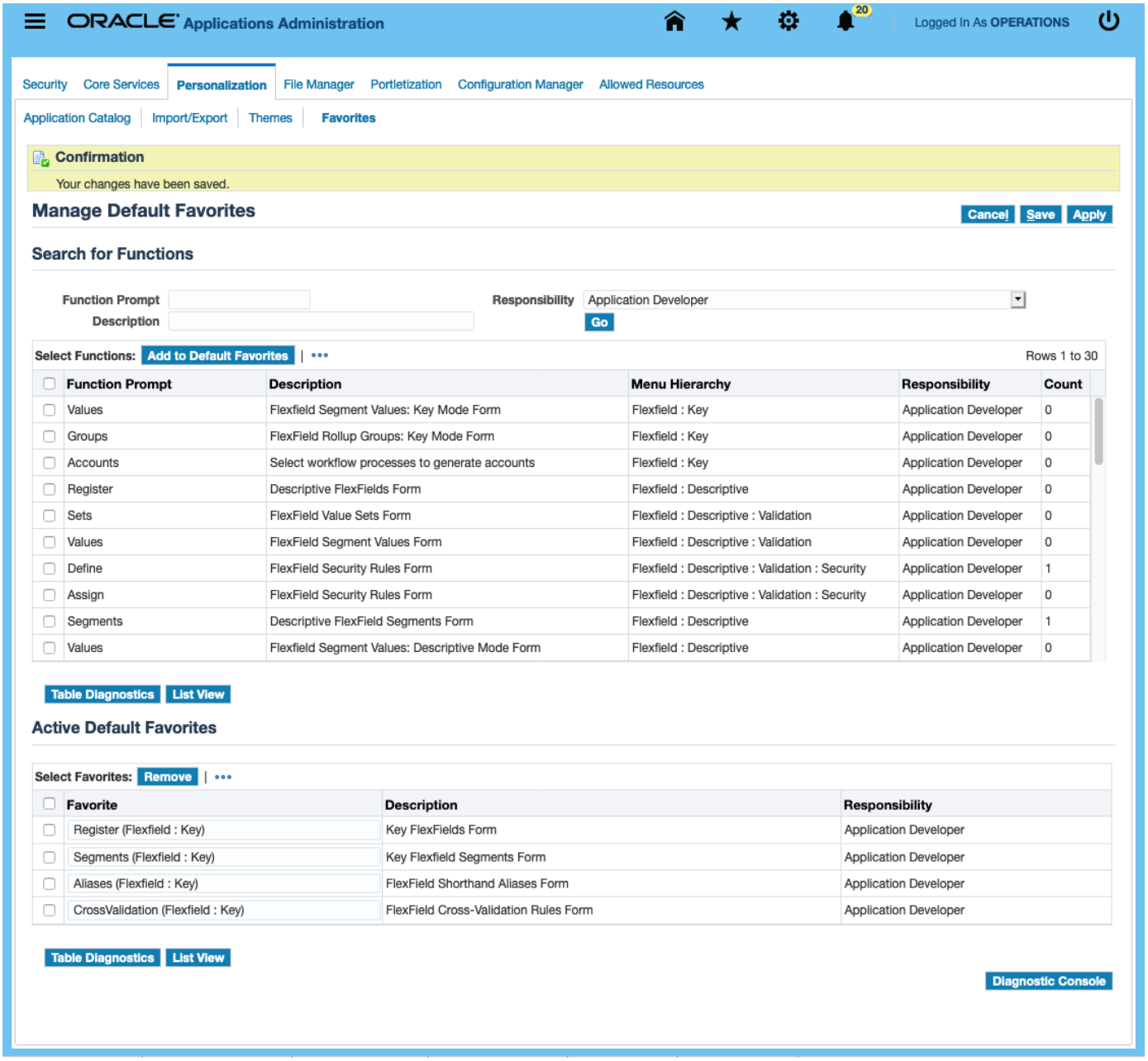
Example of selecting Default Favorites for Application Developer responsibility:



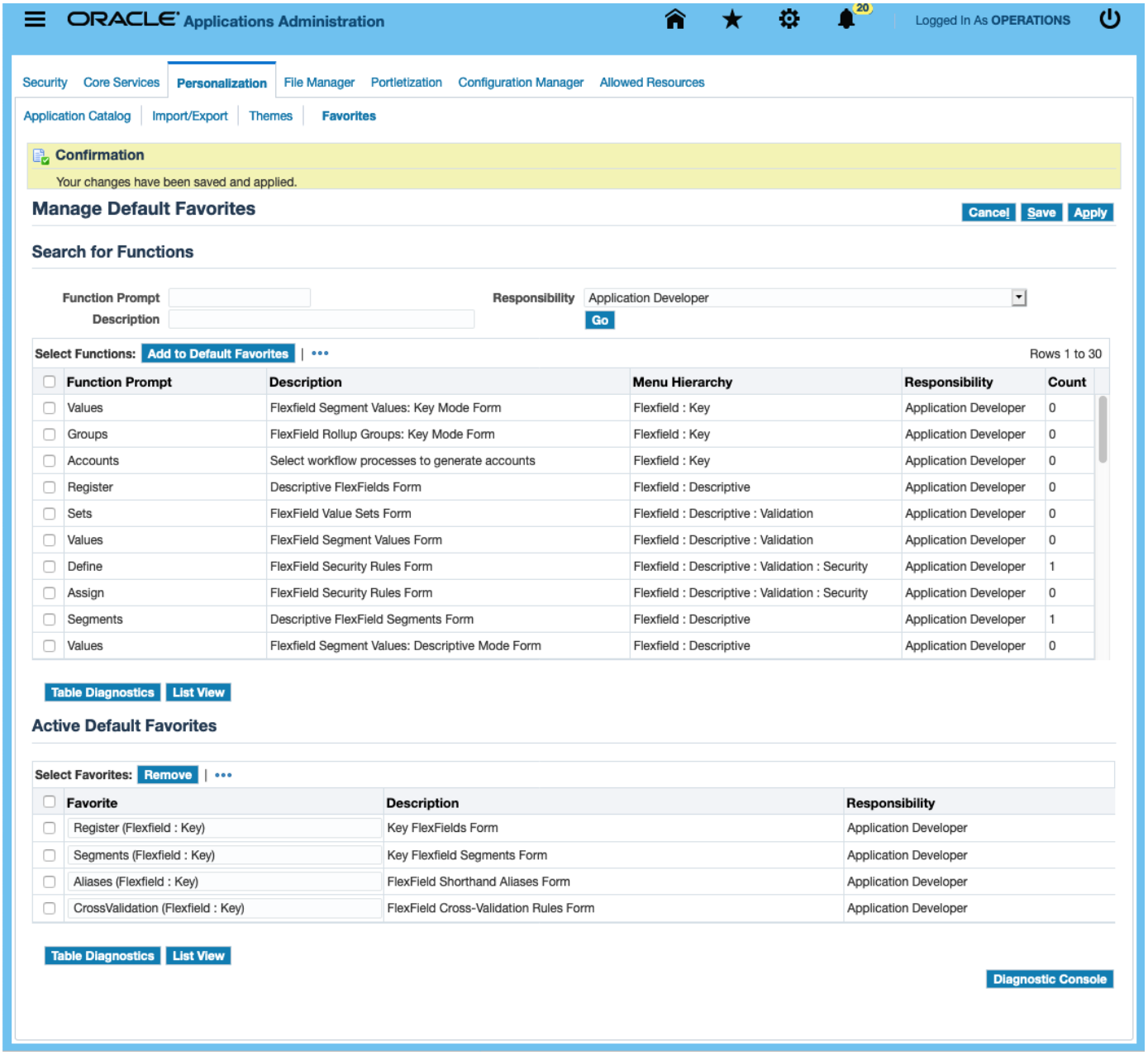
Example of Default Favorites having been added to Application Developer responsibility:



Example of saving Default Favorites that have been added to Application Developer responsibility:



Example of saving and applying Default Favorites that have been added to Application Developer responsibility:


ORACLE Applications Administration
20
Logged In As OPERATIONS
Security
Core Services
Personalization
File Manager
Portletization
Configuration Manager
Allowed Resources
Application Catalog
Import/Export
Themes
Favorites
Confirmation
Your changes have been saved and applied.
Manage Default Favorites
Cancel
Save
Apply
Search for Functions
Function Prompt
Description
Responsibility
Application Developer
Go
Select Functions:
Add to Default Favorites
Rows 1 to 30
Function Prompt
Description
Menu Hierarchy
Responsibility
Count
Values
Flexfield Segment Values: Key Mode Form
Flexfield : Key
Application Developer
0
Groups
FlexField Rollup Groups: Key Mode Form
Flexfield : Key
Application Developer
0
Accounts
Select workflow processes to generate accounts
Flexfield : Key
Application Developer
0
Register
Descriptive FlexFields Form
Flexfield : Descriptive
Application Developer
0
Sets
FlexField Value Sets Form
Flexfield : Descriptive : Validation
Application Developer
0
Values
FlexField Segment Values Form
Flexfield : Descriptive : Validation
Application Developer
0
Define
FlexField Security Rules Form
Flexfield : Descriptive : Validation : Security
Application Developer
1
Assign
FlexField Security Rules Form
Flexfield : Descriptive : Validation : Security
Application Developer
0
Segments
Descriptive FlexField Segments Form
Flexfield : Descriptive
Application Developer
1
Values
Flexfield Segment Values: Descriptive Mode Form
Flexfield : Descriptive
Application Developer
0
Table Diagnostics
List View
Active Default Favorites
Select Favorites:
Remove
Favorite
Description
Responsibility
Register (Flexfield : Key)
Key FlexFields Form
Application Developer
Segments (Flexfield : Key)
Key Flexfield Segments Form
Application Developer
Aliases (Flexfield : Key)
FlexField Shorthand Aliases Form
Application Developer
CrossValidation (Flexfield : Key)
FlexField Cross-Validation Rules Form
Application Developer
Table Diagnostics
List View
Diagnostic Console

Dialog Pages

Overview

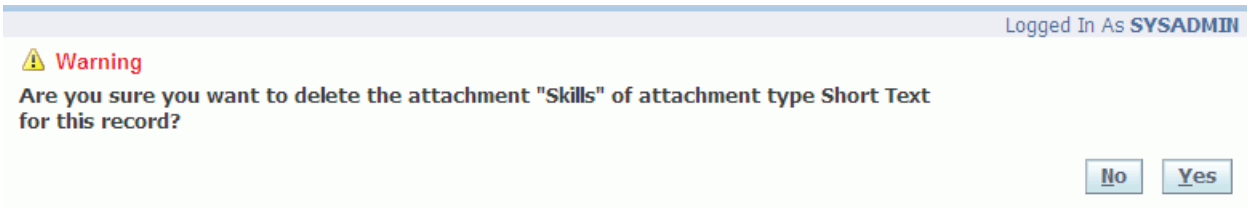
As described in the BLAF UI Guideline: Message Flows specification, messaging can be introduced into application flows when an Error, Information, Warning, Confirmation, or Processing Message needs to be displayed. There are two basic kinds of messaging flows:

- Inline Message - The message appears inline on a page around the region item that requires attention. The inline message is also repeated in the message box on top of the page.
- Dialog Page - The message appears on it's own dialog page in the flow.

  Note: As of Release 12.2.4, dialog pages render as modal pop-ups under certain conditions.

The inline message is described in detail in Chapter 4: Implementing Message Boxes. This document focuses on how to implement an Error, Information, Warning, or Confirmation message in a dialog page. The following figure shows an example of a Warning message displayed in a dialog page.

Figure 1: Warning dialog page.



Contents

- Declarative Implementation
- Runtime Control
  - Using HTML Tags in Messages
  - Handling Messages that Reference Web Beans Without Prompts
  - New Dialog Page APIs
  - Modal Dialog Pop-ups
- Personalization Considerations
- Known Issues
- Related Information

Declarative Implementation

Since a dialog page is displayed in the context of runtime events and circumstances, there is no corresponding declarative implementation.

Runtime Control

Note: As of Release 12.2.4, dialog pages render as modal pop-ups under certain conditions.

You can display an exception as a message in a dialog page using the APIs in the `oracle.apps.fnd.framework.webui.OADialogPage` class and `oracle.apps.fnd.framework.webui.OAPageContext` interface.

The OADialogPage class holds properties for the generic dialog page. To create a dialog page object, first use the constructors to instantiate the basic properties, then use the setter methods provided in the class to set additional properties. Please refer to the `OADialogPage` Javadoc for an explanation of basic usage and additional examples.

To navigate (redirect) to a dialog page, use the `OAPageContext.redirectToDialogPage` method. The `OAPageContext` interface contains the context and state information specific for a client request. The following `redirectToDialogPage` methods are provided in `OAPageContext`:

```
// Redirects to a dialog message page.
public void redirectToDialogPage(OADialogPage dialogPage);
// Convenience method to create and redirect to a dialog page with basic properties set.
Public void redirectToDialogPage(byte messageType,
OAException descriptionMessage,
OAException instructionMessage,
String okButtonUrl,
```

```
String noButtonUrl)
```

Refer to the `OAPageContext` Javadoc for further information on these methods.

Example: Redirect to a Basic Warning Page

You can include the following code example in your controller `processFormRequest` method to redirect to a basic warning page:

```
OAException descMesg = new OAException("FND", "FND_CANCEL_WARNING");
OAException instrMesg = new OAException("FND", "FND_CANCEL_ALERT");
String okUrl = APPS_HTML_DIRECTORY + "OA.jsp?OAFunc=FND_REQUISITIONS_PAGE";
String noUrl = APPS_HTML_DIRECTORY + "OA.jsp?OAFunc=FND_REQUISITIONS_PAGE&retainAM=Y";
pageContext.redirectToDialogPage(OAException.WARNING, descMesg, instrMesg, okUrl, noUrl);
```

Example: Make Dialog Page Action Buttons Submit Back to the Calling Page

Refer to the `setPostToCallingPage` method, documented in the `OADialogPage` Javadoc for a code example of how to make the dialog page action buttons submit back to the calling page. In the example, the OK button commits the changes on the dialog page and the NO button rolls back the changes.

Example: Display a Warning Page When a Delete Icon is Selected.

Refer to Task 4 of the Delete exercise in the Oracle Application Framework Toolbox Tutorial for another example of how to create a Warning dialog page. The specific example is implemented in controller class `EmployeeResultsCO` in LabSolutions.jpr of the Toolbox. This example displays a Warning page when a user selects the Delete icon from an Employees Search page. The Warning page displays Yes and No submit buttons, as shown in Figure 1.

Using HTML Tags in Messages

You may insert HTML tags in the messages displayed in a dialog page. For more information, refer to the Chapter 4 topic Custom HTML.

Handling Messages that Reference Web Beans Without Prompts

If your page contains a web bean that does not have a prompt value associated with it and a user enters an invalid value for the web bean, the error message that results will be malformed. For example, if you have a messageTextInput field with no prompt and you enter an invalid value, the error message may display as:

```
Value "A" in "" is not a number.
```

To avoid these malformed messages, use `oracle.apps.fnd.framework.webui.beans.message.OAMessagePromptBean`.

Create an `OAMessagePromptBean` and set:

- `Prompt` - to the alternate prompt that is going to be displayed for the web bean.
- `LabeledNodeId` - to those set of web bean ID's that you want associated with this alternate prompt. (These are the web beans without a current prompt associated with them).

You can associate multiple web bean ID's to the `LabeledNodeId` of this `OAMessagePromptBean`. As a result, all those web beans will be associated with the prompt of the `OAMessagePromptBean`.

The following code example illustrates this:

```
...
OAMessagePromptBean bean = new OAMessagePromptBean();
bean.setID("someID");
bean.setPrompt("Alternative");
bean.setLabeledNodeID("RequiredBeanID");
webBean.addIndexedChild(bean);
...
```

`RequiredBeanID` is the ID of the web bean with which this alternate Prompt is to be associated.

New Dialog Page APIs

The `setFooterNestedRegionCode` and `setFooterNestedRegionApplicationID` methods in the `OADialogPage` class have been deprecated.

If you use the standard **OK/NO** buttons provided on the dialog page, you do not need to make any changes.

However, if you are using the `OADialogPage.setFooterNestedRegionCode` and `OADialogPage.setFooterNestedRegionApplicationID` methods to render customized buttons under the footer region, you must update your code to use the new method `OADialogPage.setPageButtonBarRegionRefName` instead. This new method lets you set a reference to a page button bar region, built declaratively with OA Extension. You can add any number of custom buttons to the page button bar region. OA Framework renders the page button bar region under the footer. Adding your custom content to a page button bar region allows the page buttons to be rendered and positioned in the proper places specified by the latest UI standards. To render the **Return To** navigation link under the footer, use the existing `setReturnToLinkLabel` and `setReturnToLinkURL` methods in `OADialogPage`.

To render a nested region, created with OA Extension, under a dialog page header, use the new `OADialogPage.setHeaderNestedRegionRefName` method.

Modal Dialog Pop-ups

As of Release 12.2.4, a dialog page renders as a modal parameterized pop-up to eliminate the need for a user to navigate to a separate page to view a message. A modal dialog pop-up replaces the dialog page if the profile option FND: Enable Dialog Pages as Modal Popups is set to **Yes** and any one of the following conditions are met:

- The OK button URL (`OkButtonURL`) in the dialog page is defined using the `OADialogPage.setOkButtonUrl` method or as an argument in the `OAPageContext.redirectToDialogPage` API.
- The No button URL (`NoButtonUrl`) in the dialog page is defined using the `OADialogPage.setNoButtonUrl` method or as an argument in the `OAPageContext.redirectToDialogPage` API.
- The **Return To** link URL in the dialog page is defined using the `OADialogPage.setReturnToLinkURL` API.

  Note: You may need to restore the old dialog page for the following use cases:

  1. The dialog region controller has a dependency on the page layout that cannot be bypassed.
  2. The dialog page is redirected to from the `processRequest` method of the controller.

  To restore the old dialog page, use one of the following two approaches, which will override the value of the profile option **FND: Enable Dialog Pages as Modal Popups**:

  1. Use the new overloaded `OAPageContext.redirectToDialogPage` API, setting the `renderAsModalPopup` argument to `false`.
  2. Use `OADialogPage.setShowInPopup(false)`.

Personalization Considerations

- See a summary of Dialog Pages personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- Custom global buttons will not be shown in dialog pages, since product code does not take control of the custom global buttons in such pages.

Related Information

- BLAF UI Guidelines
  - Messaging Flows
  - Messaging Templates
- Javadoc File
  - `oracle.apps.fnd.framework.webui.OADialogPage`
  - `oracle.apps.fnd.framework.webui.OAPageContext`
  - `oracle.apps.fnd.framework.webui.beans.OAFormattedTextBean`
  - `oracle.apps.fnd.framework.webui.beans.OATipBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessagePromptBean`
  - `oracle.cabo.ui.beans.message.MessagePromptBean`
- Lesson
  - Delete exercise
- Sample Code
  - `oracle.apps.fnd.framework.toolbox.labsolutions.webui.EmployeeResultsCO` in LabSolutions.jpr of the Toolbox Tutorial

Dynamic User Interface

Overview

OA Framework allows you to implement dynamic pages using declarative techniques. This document describes the following in response to user actions, security grants and data state.

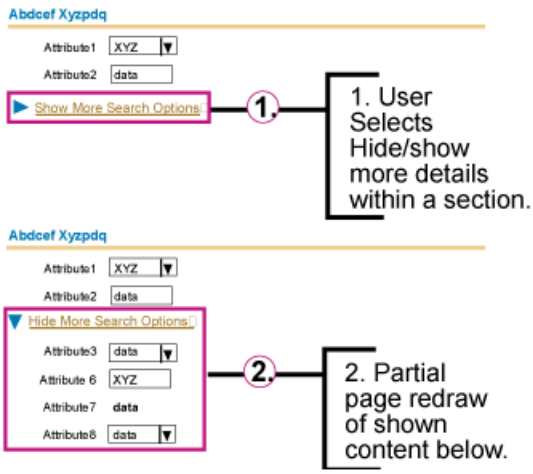
- Partial Page Rendering (PPR)
- Component-Based Function Security
- Table Content Switchers

Partial Page Rendering

As described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Partial Page Rendering (PPR), Partial Page Rendering (PPR) is a means by which designated parts of a page -- as opposed to the whole page -- are refreshed when the user performs certain actions. When PPR is enabled, UIX issues requests including the list of any partial page refresh targets. The response includes only these nodes, which are then redisplayed on the page. PPR technology doesn't require any Javascript to achieve the dynamism; it is *built in* to the components that support it.

For example, when the user selects a **Show More Search Options** Hide/Show link, the associated content refreshes using PPR as illustrated in Figure 1:

Figure 1: example of PPR refresh behavior for the Hide/Show component.



In OA Framework, PPR technology is automatically implemented for the following components and actions. Assuming the prerequisites listed later in this document are satisfied, the targets of these actions are rendered using PPR.

- Table (data set navigation, sorting, totaling, row insertion, row and cell-level Hide/Show toggling)
- Hide/Show (toggling)
- HideShowHeader (toggling)
- List of Values field (populating LOV input)

You can also declaratively define your own PPR events for selected components. For example, you can:

- Configure the selection of a poplist to cause related fields to render, be updateable, be required, or be disabled based on the selected value.
- Configure the value change of a text field to set related field values. For example, set a Supplier value and tab out, the dependent Supplier Site defaults automatically.
- Configure the selection of a master table's **singleSelection** radio button to automatically query and display related rows in a detail table.

Contents

- Implementing Partial Page Rendering Events
  - Prerequisites
  - Back Button Behavior

- Programming Alternatives to PPR
- Changing UI Properties
- PPR and Tables
- Cascading Data Changes
- Coordinating Master/Detail Tables
- PPR Event Queuing
- PPR Auto-Tabbing
- Setting the Focus in a PPR Event
- PPR Debugging

Implementing Partial Page Rendering Events

You can declaratively enable PPR events for the following item styles:

- resetButton
- link
- singleSelection
- messageCheckBox
- messageTextInput
- messageChoice
- button
- selectionButton
- submitButton
- radioSet
- radioGroup
- radioButton
- spin box
- updateable Gantt chart columns mapped to a view attribute used for drawing the Gantt (see Charts and Graphs for additional information after you read this document)
- subtabs (see Sub Tab Navigation for additional information after you read this document)

When the user selects a PPR enabled clickable source item (and the item's internal `ON-CLICK` event is fired), or changes the data value in a **messageTextInput** (its internal `ON-CHANGE` event fires), OA Framework sets the PPR event name for the request parameter named "event."

Prerequisites

For partial page rendering to work, the following prerequisites must be satisfied:

- Any bean to be partially refreshed *must* have an associated `ID`, either set declaratively in JDeveloper or specified when calling `createWebBean()` programmatically). This `ID` *must* be unique on the page.

  Tip: UIX does not render `IDs` for **rawText** (`OARawTextBean`) items. To refresh the content in a rawText item, first add it to a stack layout or other region, then make this containing region the refresh target.

- Your web bean hierarchy *must* remain unchanged for the life of the page. In other words, build the page to include all conditionally displayed components, then use the technique described below to map the `Rendered` property to a special "application properties" view object. Do *not* programmatically add or remove components from the hierarchy.

- The user's browser must support iFrame. For other browsers, full page rendering is used.

- The user must not use browser-based autocomplete mechanisms (such as Google Suggest), as these interfere with PPR and cause the user to see error messages.
- The **FND: Disable Partial Page Rendering** profile option *must* be set to **No**. If this profile value is set to **Yes**, a **Go** button renders next to each item you configure to raise a PPR event.

Back Button Behavior

When using the browser Back button, PPR behavior varies according to the browser's cache setup as well as the sequence of user actions. For example, a user navigates from Page A to Page B. On Page B, they perform an action which generates a PPR refresh (this page is now in a "B1" state). They then perform *"n"* number of PPR actions so the page renders in a "Bn" state.

- If the user navigates from Bn to Page C and then selects the browser Back button, they should return to Page Bn. This is assuming that the application module was retained and the page is regenerated from the server. If the page is regenerated locally from the browser's cache, the behavior is undefined.

- If the user is on page Bn (after performing *"n"* PPR actions) and selects the browser Back button, the behavior is undefined. In

all likelihood, it will take the user "n" back-button clicks, (amount equal to the number of PPR actions performed), to return to the last page they visited before the PPR page -- in this case, Page A.

Note: The user can access only one state of the previous page. Although they cannot go back to the intermediate PPR states, the user may need to click the browser Back button more than once to go back to the previous page.

Programmatic Alternatives to PPR

Previously, if a page needed to change in response to user actions, such as executing a search or pressing a button, you typically handled the initiating event in a `processFormRequest()` method, then issued a JSP forward back to the same page so that web bean hierarchy changes in `processRequest()` could be made.

With PPR, this is no longer necessary for most UI changes. Whenever possible, leverage the declarative PPR features instead of the programmatic solution.

Javascript remains a prohibited technology for Oracle's internal E-Business Suite developers (see Implementing the Controller: Javascript).

Changing UI Properties

Assuming you've added an item to your page with a style in the list above, the following instructions describe how to configure it to generate PPR events to control the properties of related items. For example, hide or show appropriate fields when a poplist value changes. You can use the industry-standard *Expression Language* (EL) binding () to declaratively bind the values of certain properties to different attributes or parameters.

The following are the only properties for which EL bindings can be defined:

- Read Only
- Rendered
- Required
- Disabled
- Prompt
- Sort Allowed
- Maximum Length

Note: Refer to the EL (Expression Language) Support topic for more information.

Tip: To control the value of other properties, use bound values. For example, use an `oracle.apps.fnd.framework.webui.OADataBoundValueViewObject` to bind a property to a view object attribute.

Note: To change page properties based on a user's List of Values selection, follow a different procedure. See Use the LOV as a Context Switcher in the List of Values document for specific instructions after you read this document.

Step 1: Select your item in the JDeveloper Structure pane and set the following properties:

- `Action Type` - set to `firePartialAction` to enable a PPR event. (The default value is **none**).
- `Event` - set to the name of the event that OA Framework puts on the request when a PPR action occurs. (The default value is **update**).

  Note: OA Framework does not impose any restrictions on the event name.
- `Parameters` - To pass parameters for `firePartialAction`, select the **...** button in the `Parameter` property to open a Parameters window, then in the Parameters window, specify the name and value of each parameter that you want to pass. The values can be static text, bound values using EL syntax, or bound values using a combination of static text and EL syntax. For example:

  `<Text>`

  or

  `${oa.<viewInstanceName>.<viewAttributeName>}`

  or

  `${oa.text.<TEXT>}+${EL}`

  Where *EL* is any valid EL-type expression. Refer to the EL Support topic for detailed information.
- `Submit` - set to **True** for the PPR action to submit the form; or **False** for the PPR action to perform an `HTTP GET` request. In

both instances the event is handled in your `processFormRequest()` method.

Note: False is not a supported value. However, you may experiment with it.

- `Disable Client Side Validation` - set to **True** if you don't want client side validation to be performed when the PPR event is fired.

  Note: This applies *only* if the PPR item is configured to perform a form submit. See Implementing the Controller: Disabling Validation for additional information.

- `Disable Server Side Validation` - set to **True** if you don't want server-side validation errors to be displayed. See Implementing the Controller: Disabling Validation for additional information.

Step 2: Create the items whose property values should be set in response to the PPR event. For example, assume you have several text fields whose `Rendered` and `Read Only` properties are determined by a poplist value. Create these **messageTextInput** items and configure their standard properties.

Note: UIX currently does *not* support the ability to hide components whose direct parent region is a **cellFormat** or a **rowLayout**. As a workaround, to hide a component such as a button in a **cellFormat**, add **flowLayout** to your **cellFormat** region first, then add the button to the **flowLayout**.

Additionally, if the target component that you want to effect with the PPR event is a table content switcher, you *must* add a **flowLayout** to your table, then add the **switcher** region to the **flowLayout** region. Configure the **flowLayout** region's properties as described in Step 4 below, and not the **switcher's** properties.

Step 3: In the same package where you created the page's root UI Application Module and any UI-specific view objects, create a special *application properties* view object. Add this view object to your page's root UI application module or, if you're implementing PPR in a shared region, associate this view object with the shared region's application module.

Note: This view object should follow the naming convention of: ApplicationModuleName PVO. Therefore, an application properties view object created for an EmployeeAM should be named EmployeePVO. Furthermore, each application module should have no more than one application properties VO.

- Add a `Number` primary key transient attribute named **RowKey**. (A default value of 1 will be set in Step 7 below). Select the **Key Attribute** checkbox.
- Add one or more transient attributes for the properties that you want to set in response to PPR events.
- Verify that all the attributes are designated as **Updateable Always**.

The transient attributes should be of the following types based on the properties to be set in response to PPR events. Note the list of corresponding valid values for reference when you write code to set these attribute values.

Property	Attribute Data Type	Valid Values
Required	String	• yes • no • uiOnly • validatorOnly
Read Only	Boolean	• Boolean.TRUE • Boolean.FALSE
Rendered	Boolean	• Boolean.TRUE • Boolean.FALSE
Disabled	Boolean	• Boolean.TRUE • Boolean.FALSE

For example, in the ToolBox Sample Library, we created a SampleBrowserPVO (associated with the SampleBrowserAM) with the following transient attributes:

- PoApproveReadOnly
- PoApproveRequired
- PoApproveRender

- PoApproveReject

Although these transient attributes serve a specific purpose, they should be as abstract as possible to allow use by several different components within a page or shared region. For example, the `PoApproveReadOnly` property listed above is intended to be referenced by several different components when the PO approval status changes. This approach is more *abstract* than creating a VO property specifically intended to control the updateable property of "component X."

Note: Individual attributes should *not* be set and referenced by multiple pages; each attribute should be used exclusively within a single page. Since each application module has only one application property view object, and a single root UI application module can be associated with many pages, your application property VO may include attributes for several pages.

Step 4: Open the JDeveloper property inspector for each of the PPR event target items you created in Step 2. Set the `Rendered`, `Read Only`, `Disabled` and/or `Required` property values to bind to the view object attributes you created in Step 3 using the following EL syntax:

```
${oa.<ViewInstanceName>.<ViewAttributeName>}
```

For example, in the ToolBox Tutorial Sample Library, a text field has its `Rendered` property configured to:

```
${oa.EmployeePVO1.PoApproveRender)
```

Step 5: In the application module that contains your application properties view object, add a method to set the application property values. For example, in the ToolBox Tutorial Sample Library we have a method called `handlePoApprovaChangeEvent()` that reads the current value of the PPR event source poplist from the page's underlying entity object, and sets the appropriate property values as shown:

```
public void handlePoApproveChangeEvent()
{
  // Get the special, single-row application properties and make the first
  // (only) row current.
  OAViewObject vo = (OAViewObject)findViewObject("SampleBrowserPVO1");
  OARow row = (OARow)vo.first();

  // Get the value of the view object attribute with the PO Approval status.
  OAViewObject poVO = (OAViewObject)findViewObject("PurchaseOrderHeadersVO1");
  OARow poRow = (OARow)poVO.getCurrentRow();
  String status = (String)poRow.getAttribute("StatusCode");

  // Set the application property values based on the PO Approval status value.
  if ("APPROVED".equals(status))
  {
    row.setAttribute("PoApproveRender", Boolean.TRUE);
    row.setAttribute("PoRejectRender", Boolean.FALSE);
    row.setAttribute("PoApproveReadOnly", Boolean.TRUE);
    row.setAttribute("PoApproveRequired", "yes");
  }
  else if ("REJECTED".equals(status))
  {
    row.setAttribute("PoApproveRender", Boolean.FALSE);
    row.setAttribute("PoRejectRender", Boolean.TRUE);
  }
  else
  {
    row.setAttribute("PoApproveRender", Boolean.TRUE);
    row.setAttribute("PoRejectRender", Boolean.TRUE);
    row.setAttribute("PoApproveReadOnly", Boolean.TRUE);
    row.setAttribute("PoApproveRequired", "no");
  }
} // end handlePoApproveChangeEvent()
```

Step 6: In an appropriate controller for handling the PPR event source item's actions, add code to the `processFormRequest()` method to detect the PPR event and invoke the application module method you created in Step 4. The ToolBox Sample Library's controller for the purchase order approval poplist includes the following code.

Note: This example is checking for two different PPR events. The second is described below in Coordinating Master/Detail Tables.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormRequest(pageContext, webBean);
  OAApplicationModule am =
    (OAApplicationModule)pageContext.getApplicationModule(webBean);

  String event = pageContext.getParameter("event");
```

```
  // If the user changes the value of the po approval poplist, call the
  // event handler in the AM to set the appropriate SampleBrowserPVO
  // values.

  if ("poApproveChange".equals(event))
  {
    am.invokeMethod("handlePoApproveChangeEvent");
  }
  else if ("supplierSelect".equals(event))
  {
    am.invokeMethod("handleSupplierSelectionEvent");
  }
} // end processFormRequest()
```

Tip: If PPR is disabled (the **FND: Disable Partial Page Rendering** profile value is **Yes**), this same code executes when the user selects the PPR event source item's **Go** button.

Step 7: In the application module that contains your application properties view object, add a method to initialize this view object (if this application module already has a page initialization method, simply add this code to it). For example, in the ToolBox Sample Library we have a method called `initializePPRExamplePage()` that automatically creates a purchase order header EO and initializes the application properties VO:

```
public void initializePPRExamplePage()
{
  // Create purchase order header
  ...

  OAViewObject appPropsVO =
    (OAViewObject)findViewObject("SampleBrowserPVO1");

  if (appPropsVO != null)
  {
    // If the VO already has a row, skip its initialization.  Note that
    // calling getFetchedRowCount() won't perform a DB hit on a VO with
    // no SELECT and only transient attributes.

    if (appPropsVO.getFetchedRowCount() == 0)
    {
      // Setting the match fetch size to 0 for an in-memory VO
      // prevents it from trying to query rows.
      // Calling executeQuery() is a small workaround to a known
      // BC4J issue that ensures that modified rows in this
      // transient view object are not lost after commit.  See
      // View Objects in Detail for additional information about
      // both of these calls.

      appPropsVO.setMaxFetchSize(0);
      appPropsVO.executeQuery();

      // You must create and insert a row in the VO before you can start
      // setting properties.

      Row row = appProposVO.createRow();
      appPropsVO.insertRow(row);

      // Set the primary key value for this single-row VO.
      row = (OARow)appPropsVO.first();
      row.setAttribute("RowKey", new Number(1));

    }

    // Initialize the application properties VO (and the UI) based on the
    // default PO approval value set on the underlying object.
    handlePoApproveChangeEvent();
  }
  else
  {
    // throw exception
  }
} // end initializePPRExamplePage()
```

Step 8: In your page's `processRequest()` method, invoke the application module page initialization method you created in Step 7. For example:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  OAApplicationModule am =
    (OAApplicationModule)pageContext.getApplicationModule(webBean);
  am.invokeMethod("initializePPRExamplePage");

} // end processRequest()
```

PPR and Tables

You can also fire row-level PPR events for items in a table, including advanced tables and table-in-table. First and foremost, configure the items as described above.

Note: If you are working with a table-in-table or HGrid that uses view links, you need to modify your SPEL syntax as shown: `${oa.current.viewAttrName}`. The inclusion of *current* keyword lets you say "get the value from whatever row BC4J is using to render the current row" since you won't know how to access the view objects and row sets created dynamically for these complex components.

To ascertain in which row the column's PPR event was fired, add the following code to your `processFormRequest` method:

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormRequest(pageContext, webBean);

  OAApplicationModule am =
    (OAApplicationModule)pageContext.getApplicationModule(webBean);
  String event = pageContext.getParameter("event");

  if ("<ItemPPREventName>").equals(event))
  {
    // Get the identifier of the PPR event source row

    String rowReference =
      pageContext.getParameter(OAWebBeanConstants.EVENT_SOURCE_ROW_REFERENCE);

    Serializable[] parameters = { rowReference };

    // Pass the rowReference to a "handler" method in the application module.

    am.invokeMethod("<handleSomeEvent>", parameters);
  }
}
```

In your application module's "handler" method, add the following code to access the source row:

```
OARow row = (OARow)findRowByRef(rowReference);
if (row != null)
{
  ...
}
```

Cascading Data Changes

Use one of the following recommendations when managing cascading data changes during a PPR request:

- If the cascaded data change is dependent on more than one EO attribute, then this validation/resolution and final setting of the dependent attributes should be in the `validateEntitiy()` method. This is because BC4J does not guarantee the order in which the EO attributes are set. See the Update/Validate section of the Java Entity Objects document for more information.
- Perform the necessary validation/resolution of dependent attributes in the `setAttribute()` method and set the dependent attributes. See the following example.

  Note: If this attribute is an editable field, this newly set value may be overwritten by the form data that is being processed.

To update data values when the user changes the value in a given field, such as if the user specifies a supplier value you want to set the default supplier site, follow these instructions:

Step 1: Configure a **messageTextInput** item to issue a PPR event (see Step 1 in Changing UI Properties above). When the data value changes and a PPR event is fired, OA Framework sets the selected data on the underlying view object.

Step 2: Assuming the underlying view object is based on an entity object, the entity object setter associated with the PPR source item should implement the logic to call the setters on the associated target entity object attributes. For example, in a purchase order header the `setSupplierID()` method calls the `SupplierEOImpl's` Entity Expert to obtain the `ID` for its default supplier site, which it then passes to the `setSupplierSiteID()` method.

Note: In some circumstances, you may want to suppress this behavior in your entity object. For example, when loading data from an XML file and both the supplier and site IDs are specified. This is currently an open design issue; contact the OA Framework team if you need assistance.

To cascade data in a region whose items have no data source, such as (a search criteria region, set the downstream item values directly in your controller. You can still invoke a custom method on the application module that asks a static Entity Expert for information on what value to set, and returns this value to the controller. Whenever possible, all data-related logic should be consolidated in the entity objects and their experts.

See the OA Framework ToolBox Tutorial for a complete, working example of cascading data changes.

Note: If you want a poplist to automatically reflect changes to its data as a consequence of a PPR event, you *must* specify the `Picklist View Instance` when you define it (also set by calling `setPicklistViewUsageName()`) and *not* the `Picklist View Definition` (also set by calling `setPicklistViewObjectDefinitionName()`). This will *not* work if you choose to create a poplist that is cached in the JVM.

Coordinating Master/Detail Tables

To automatically display rows in a detail table based on the selection of a row in a master table, and the tables are displayed on the same page, follow these instructions to comply with the BLAF Master/Detail Page Template:

Step 1: If you have not already done so, create an application properties view object as described above.

Step 2: Add a `String` attribute to your application properties view object. Give it a name like `DETAIL_TABLE_TEXT`. Set this value dynamically to display a contextual title for the details table as shown in Figure 2. In this example from the ToolBox Sample Library, when the user selects a supplier from the master table, the contextual title for the details table indicates that it is displaying supplier sites for the selected supplier.

Figure 2: Example of Master / Detail Tables



Step 3: Create a message in Message Dictionary to be used for the details table title. It should include a token for the master table unique (user) identifier. For example: `&SUPPLIER_NAME: Supplier Sites`. Also, create a generic message to display when no master rows are selected. (In this case, we want to display *Supplier Sites*).

Step 4: Create the master and detail view objects, then configure a view link between them as described in Implementing the Model: View Objects and View Links. Add the master and detail view objects (with the detail view object accessed via the view link) to the page's application module. Finally, add an updateable, transient "SelectFlag" column to the master view object to use as a data source for the table's **singleSelection** radio button item.

Step 5: Create the master table as you normally would (see the Advanced Tables documentation for additional information) and include a **singleSelection** component. Bind the items to the master view object you created in Step 4, and configure the **singleSelection** item to issue a PPR event (see Step 1 in Changing UI Properties above).

Step 6: Create the detail table and bind it to the detail view object that you created in Step 4.

Step 7: Add a method to the page's application module to handle the **singleSelection** choice. This code must find a selected row in master view object's data set and simply mark it as the current row. Because of the view link configuration, BC4J automatically queries the detail view objects. This logic also updates the application properties `DetailTableText` attribute value based on the current master row.

For example:

```
import oracle.apps.fnd.common.MessageToken;
import oracle.apps.fnd.framework.OAViewObject;
import oracle.apps.fnd.framework.server.OADBTransaction;
...

public void handleSupplierSelectionEvent()
{
  OADBTransaction txn = getOADBTransaction();
  String detailTableText = null;

  // Find the selected radio button so we can mark the current row
  OAViewObject vo = (OAViewObject)findViewObject("SuppliersVO1");

  Row row = vo.getFirstFilteredRow("SelectFlag", "Y");

  // This check assumes getFirstFilteredRow returns null if
  // it finds no matches.
  if (row != null)
  {
    // Set the master row and get the unique identifier.
    Row masterRow = row;
    vo.setCurrentRow(masterRow);
    String supplierName = (String)masterRow.getAttribute("Name");

    MessageToken[] tokens = { new MessageToken("SUPPLIER_NAME", supplierName)};

    detailTableText =
      txn.getMessage("AK", "FWK_TBX_SITES_FOR_SUPPLIER", tokens);
  }
  else
  {
    // If there is no selected row, display a default generic message.
    detailTableText =
      txn.getMessage("AK", "FWK_TBX_SUPPLIER_SITES", null);
   }

  // Now set the text message on the DETAIL_TABLE_TEXT attribute in
  // the application properties VO.

  SampleBrowserPVOImpl appPropsVo = getSampleBrowserPVO1();
  Row appPropsRow = appPropsVo.getCurrentRow();

  if (appPropsRow != null)
  {
    appPropsRow.setAttribute("DetailTableText", detailTableText);
  }
} // handleSupplierSelectionEvent()
```

Step 8: In your root application module's initialization method where you configured your application properties view object, call the event handler you created in Step 7. This ensures that the header text renders properly when the page first displays before the user makes a selection. For example, in the ToolBox Sample Library, we have the following initialization logic that calls the `handleSupplierSelectionEvent()` method.

```
...
  OAViewObject appPropsVO = (OAViewObject)findViewObject("SampleBrowserPVO1");

  if (appPropsVO != null)
  {
  // This checks the in-memory cache (doesn't cause a database hit).
  // If the VO doesn't include a row yet, create and insert one.

  if (appPropsVO.getFetchedRowCount() == 0)
    {
      // Calling executeQuery() is a small workaround to a known
      // BC4J issue that ensures that modified rows in this
      // transient view object are not lost after commit.  See
```

```
      // View Objects in Detail for additional information about
      // both of these calls.

      appPropsVO.setMaxFetchSize(0);
       appPropsVO.executeQuery();

      appPropsVO.insertRow(appPropsVO.createRow());

      // Set the primary key value for this single-row VO.
      OARow row = (OARow)appPropsVO.first();
      row.setAttribute("RowKey", new Number(1));
    }

  ...

  // Initialize the header text for the supplier sites detail table.
  handleSupplierSelectionEvent();

...
```

Step 9: In an appropriate controller for your master table, add code to `processFormRequest()` to detect the radio button selection and invoke the application module method that marks the current row in the master data set.

Step 10: Add the following `processRequest()` logic to your controller to bind the child table header's `Text` property to a specially created attribute in the application properties VO.

```
import oracle.apps.fnd.framework.webui.OADataBoundValueViewObject;
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.layout.OAHeaderBean;
...

public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  ...

  // This also works with an advanced table.
  OAHeaderBean header =
    (OAHeaderBean) webBean.findIndexedChildRecursive("<RegionName>");
  header.setAttributeValue(OAWebBeanConstants.TEXT_ATTR,
        new OADataBoundValueViewObject(header, "<AttributeName>",
                                               "<AppPropertiesViewInstanceName>");
  ...
}
```

PPR Event Queuing

By default, when a PPR event fires, all subsequent events on the page are queued and processed. For example, a page has a text input field configured to fire a PPR event when the user leaves and also has a **Submit** button. The user makes a change to the data in the text input field and then selects the **Submit** button. In this case, the text field's PPR event fires and the button click is queued. After receiving the PPR response, the Submit action is performed.

Note: To disable this feature, set the value of the profile FND_PPR_EVENT_QUEUE_DISABLED to **Y**. In this case, when a PPR event fires, all subsequent events on the page are ignored.

To implement event queuing on a specific page, add the following code to the `processRequest()` of that page. The default behavior is then enabled for *only* that page:

```
import oracle.apps.fnd.framework.webui.beans.OABodyBean;
...

public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  OAWebBean body = pageContext.getRootWebBean();

  if (body instanceof OABodyBean)
  {
  ((OABodyBean)body).setFirstClickPassed(true);
  }
  ...
  ...
}
```

Note: In an advanced table under a layout region, a queued event may throw an error in some cases. To mitigate this issue, code any event handling pertaining to table rows in a separate controller associated with the table.

PPR Auto-Tabbing

By default, when you tab out of a text input field that triggers a partial page rendering event, the focus moves to the next field on the page. If the profile option FND_PPR_DISABLE_AUTOTABBING is set to '**Y**', the focus will not be saved during PPR and the focus handling is left to the browser. Currently, tabbing out of a PPR enabled field in Mozilla takes the focus to next focusable item but in Internet Explorer (IE) it remains in the same field. The default for this profile option is 'N'.

Setting the Focus in a PPR Event

By default, OA Framework/UIX tries to retain the focus to the same element during a PPR event or to the next field on the page based on the value of the profile option FND_PPR_DISABLE_AUTOTABBING. If the focus handling is left to the browser, as described in PPR Auto-Tabbing, it is possible for you to set the focus to a specific element on the page by using the `setRequestedFocusId(String requestedFocusId)` API available in `OABodyBean`. Note that the node to which focus is to be set must have an ID associated with it. Similarly, you can use the `setInitialFocusId(String initialFocusId)` API available in `OABodyBean` to set the focus to an specific element during a full page refresh.

PPR Debugging

The PPR Debugging feature is used to see the changes in the partial targets. By default, the PPR Debugging is disabled. This feature can be enabled by setting the value of the profile option FND_ENABLE_PPR_DEBUGGING to **Y**. (See the PPR section of the Profile Options document for more information about this option). When the PPR debugging feature is enabled, the partial targets are displayed at the top of the screen as the page renders.

Known Issues

- See a summary of key Partial Page Rendering issues with suggested workarounds if available.

Personalization Considerations

- None

Component-Based Function Security

You can also configure item properties using function security and the same SPEL syntax that you use to bind item properties to the "application properties" view object. Read Implementing Partial Page Rendering Events before reading this section.

Currently, you can control the following item properties using this approach:

- Rendered
- Read Only
- Disabled
- Required

Note: This section does *not* describe how to create menus, functions, or grants; it assumes you know what they are and how to create them. For additional information about these topics, see Tabs / Navigation and Page Security.

Step 1: Create a function with a name that describes the rule you want to implement. For example, you have a text field whose `Read Only` property should be **True** if the user does *not* have access to the `SUPPLIER_READ_ONLY` function when logged in using the **BUYER** responsibility.

Step 2: Create a grant for this function. In this example, we would create a function grant for `SUPPLIER_READ_ONLY` in the context of the responsibility **BUYER**.

Step 3: Create the items whose `Read Only` property should be set based on the state of this security grant. Set the `Read Only` property using the following SPEL syntax:

```
${oa.FunctionSecurity.<FunctionName>}
```

The test will return **False** if <*FunctionName*> is granted to the current user/responsibility; otherwise **True**.

In this example, we would set the `Read Only` property to:

```
${oa.FunctionSecurity.SUPPLIER_READ_ONLY}
```

If the user is logged in to the **BUYER** responsibility and has been granted access to this function, OA Framework returns **False** in the function security test. When the `Read Only` property is set to **False**, the item is updateable.

Expressions and Test Results

The following table summarizes the properties you can set and the corresponding results that OA Framework sets as the property's value.

Property	Property Internal Name	Expression	Test Result
Rendered	RENDERED_ATTR	`${oa.FunctionSecurity.<myFunctionName>}`	Returns **True** if <*myFunctionName*> is granted, otherwise **False**.
Read Only	READ_ONLY_ATTR	`${oa.FunctionSecurity.<myFunctionName>}`	Returns **False** if <*myFunctionName*> is granted, otherwise **True**.
Disabled	DISABLED_ATTR	`${oa.FunctionSecurity.<myFunctionName>}`	Returns **False** if <*myFunctionName*> is granted, otherwise **True**.
Required	REQUIRED_ATTR	`${oa.FunctionSecurity.<myFunctionName>}`	Returns **no** if <*myFunctionName*> is granted, otherwise **yes**. *

* To set one of the other two `Required` property values (uiOnly or validatorOnly), you *must* configure the `Required` property to bind to a `String` attribute in the application properties VO, and in this attribute's getter, call function security and return the desired property based on the security test result.

Table Content Switchers

Unlike an Application or Context Switcher, which is a UI control that allows a user to switch the application or page context to display, a table content Switcher is a region with two or more display alternatives. The display alternatives are predefined items of which only one is selectively rendered at any given time.

If your table column is a Switcher, then you can:

- Assign a header label for the column by setting the OA Extension `Prompt` property for each Switcher nested region item.
- Enable sorting for the item by setting the OA Extension `Initial Sort Sequence` property for each Switcher nested region item. OA Framework determines the view attribute to sort by using the following list, in order of precedence:

  1. `Sort By View Attribute` of the Switcher nested region item.
  2. `Sort By View Attribute` of the selected child region item, that is, the named child of the switcher that is selected by the decode SELECT statement.
  3. `View Attribute` of the selected child region item.

  Note: Do not use the `View Attribute` of the Switcher Nested Region item for sorting because it simply determines which named child is selected.

Switcher Usage

Limit the use of Switchers to within tables, advanced tables, or HGrids, particularly when you want to switch between different kinds of web beans, such as a poplist or a checkbox, or when you want to switch between different images. Although you can technically use a Switcher (`OASwitcherBean`) outside a table, instead use SPEL binding for the `Rendered` property of the content that you want to conditionally display.

The image switching case is demonstrated in the ToolBox Tutorial Delete Lab. The tutorial example creates an employee table that contains a Delete column. The Delete column allows you to delete employees from the table, depending on the status of the employee - if the employee is active, the Delete icon is enabled, otherwise it is disabled. However, to meet accessibility standards, ALT text is associated with the enabled icon as well as the disabled icon. At runtime, to be able to display the enabled Delete icon, with its ALT text, or the disabled Delete icon with its appropriate ALT text, the tutorial uses the convenience of a table content Switcher to switch between the two distinct sets of attribute values for the same web bean type.

Using bound values instead of a Switcher in this case, would bind the image source of the Delete icon to a view object attribute to get the image file name, and bind the ALT text to another view object attribute to get the ALT text for the image.

You can implement a Switcher declaratively by defining two or more items representing your display alternatives and adding these to a Switcher region within your parent table.

When `exportByViewAttr` is set on a switcher level, instead of depending on the datatype of the component rendered, OA Framework exports the value based on the sqldatatype (database). For example, the datatype of a component can be either DATE or DATETIME. If the datatype is DATE, *only* the date is exported and if the datatype is DATETIME, *both* the date and the time are exported. However, when basing values on the database, the sqldatatype can be either DATE or TIMESTAMP and both contain date

and timestamps, (such as DD-MM-YYYY hh:mm:ss). In either case, *both* the date and the time are exported.

Declarative Implementation

Step 1: To add a Switcher to a table region, update the SQL query for the table region's view object to include a "Switcher" column or attribute. The Switcher attribute must return the name of the conditionally displayed item or region to render. Remember that a Switcher region can contain two or more nested region items as display alternatives. You can add this "Switcher" attribute to your view object by including a DECODE statement in your SELECT statement. The DECODE statement determines which child item name to return.

For example, in the ToolBox Tutorial Delete Lab, a Delete column is added to a results table in the Employee search page. The Delete column is a Switcher region that can either display the enabled Delete icon and its ALT text or the disabled Delete icon and its ALT text. The underlying `EmployeeSummaryVO` query includes the following DECODE statement to determine whether the employee can be deleted based on the employee's status:

```
decode(nvl(to_char(EmployeeEO.END_DATE), 'N'), 'N','DeleteDisabled',
  'DeleteEnabled') AS DELETE_SWITCHER
```

Step 2: Create a Switcher region, by right-clicking your table region in the Structure pane and selecting **New > switcher** from the context menu. Select the Switcher region and update the following properties in the Property Inspector with these values:

- `ID` - <*the name of the Switcher region*>. Set in accordance with the OA Framework File naming standards
- `Item Style` - **switcher**
- `Prompt` - <*the label that appears in the table column*>
- `Rendered` - **True**
- `View Instance` - <*name of the underlying the view instance*>
- `View Attribute` - <*name of the "Switcher" attribute that you added to the view object in Step 1*>

Step 3: Right-click the default Switcher case that JDeveloper created for you and select **New > Item** or **New > Region** from the context menu.

- Select the item/region and update the `ID` property with the OA Framework File Standards.

  Note: The `ID` *must* match the corresponding SQL `DECODE` function return value that determines when this item or region should be displayed. For example, if you are creating a Switcher for the ToolBox SQL snippet shown above, the image item that you define for the disabled Delete icon will have its `ID` set to **DeleteDisabled**.

- Configure other properties as needed for your item or region.

Step 4: Add additional cases to represent display alternative for the Switcher region. Right-click the Switcher region in the **Structure** pane and select **New > case** from the context menu. Configure your item or region as described in Step 3, and repeat as necessary until all display alternatives are defined.

Step 5: (Required only if you add images to a Switcher in a classic table) You *must* manually center align the images as shown in the code example below:

```
import oracle.cabo.ui.data.DictionaryData;
import oracle.cabo.ui.data.DataObjectList;
import oracle.apps.fnd.framework.webui.beans.table.OATableBean;
...

public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  // This controller is associated with the table.
  OATableBean table = (OATableBean)webBean;

  // We need to format the Switcher image column so the image is centered
  // (this isn't done automatically for Switchers as it is for
  // plain image columns). We start by getting the table's
  // column formats.

  // NOTE!!! You must call the prepareForRendering() method on the table *before*
  // formatting columns. Furthermore, this call must be sequenced *after* the
  // table is queried, and *after* you do any control bar manipulation.
```

```
  // We need to get a handle to the table so we can set it's width to 100%.

  table.prepareForRendering(pageContext);

  DataObjectList columnFormats = table.getColumnFormats();
  DictionaryData columnFormat = null;
  int childIndex = pageContext.findChildIndex(table, "DeleteSwitcher");
  columnFormat =(DictionaryData)columnFormats.getItem(childIndex);
  columnFormat.put(COLUMN_DATA_FORMAT_KEY, ICON_BUTTON_FORMAT);
}
```

Runtime Control

There are no programmatic steps necessary to implement a Switcher region. However, if you wish to set a default case for a Switcher region, you can do so programmatically by calling the following `oracle.cabo.ui.beans.SwitcherBean` API:

```
public static void setDefaultCase(MutableUINode bean,
  java.lang.String defaultCase)
```

Usage Notes

- If the view object associated with a switcher returns a value for a case that does not match any of the existing cases, then:
  - If a default case is defined, the default case renders. See the Runtime Control section for more information.
  - If a default case is not defined, a developer mode error is thrown.
- For a table or HGrid web bean, you can display a named child under a switcher. However, if the switcher's view attribute returns a value that does not match the named child's name, nothing renders (that is, the switcher region is blank). You can use this behavior to your advantage, such that you do not need to define any additional spacer bean if you purposely do not want anything to display for a case.
- If `ExportByViewAttr` is set on the switcher level, the datatype of the exported value will depend on the sqldatatype and *not* the datatype of the component rendered.

Personalization Considerations

- See a summary of Dynamic User Interface personalization considerations in the Oracle Application Framework Personalization Guide.

Related Information

- BLAF UI Guidelines
  - Partial Page Rendering (PPR)
  - Master/Detail Page Template
- Javadoc Files
  - `oracle.cabo.ui.beans.SwitcherBean`
  - `oracle.apps.fnd.framework.webui.beans.OASwitcherBean`
- OA Framework Developer's Guide
  - EL (Expression Language) Support
  - OA Framework File Standards
  - Classic Tables
  - Advanced Tables
  - Tabs / Navigation
- OA Framework ToolBox Tutorial / Sample Library
  - ToolBox Tutorial Partial Page Rendering (PPR) Lab
  - ToolBox Tutorial Delete Lab (table switcher)
  - `oracle/apps/fnd/framework/toolbox/sampleLib/webui/PartialPageExamplePG.xml`
  - `oracle/apps/fnd/framework/toolbox/sampleLib/webui/PartialPageExampleCO.java`
  - `oracle/apps/fnd/framework/toolbox/sampleLib/server/SampleBrowserAMImp.java`
  - `oracle/apps/fnd/framework/toolbox/sampleLib/server/SampleBrowserPVO.xml`

EL (Expression Language) Support

Overview

Expression Language (EL) is a scripting language that enables you to easily access and manipulate Java application data through JSP. In Release 12.2, EL support is enhanced to include more EL-supporting properties, more EL types, the use of operators in EL, and the use of static text with EL.

Note: Prior to Release 12.2, EL support was referred to as SPEL (Simplest Possible Expression Language) support, which included a subset of the currently supported EL types and EL-supporting properties.

Contents

- Supported EL Types
- Using Operators in EL
- Combining Static Text with EL
- EL-Supporting Properties

Supported EL Types

The following table lists the EL types that are supported by OA Framework.

EL Type	Description
`${oa.<ViewInstance>.<ViewAttr>}`	Evaluates to <*ViewInstance*>.getAttribute("<*ViewAttr*>").
`${oa.current.<ViewAttr>}`	Evaluates to <*ViewInstance_attached_to_the_bean*>.getAttribute("<*ViewAttr*>").
`${oa.encrypt.<ViewInstance>.<ViewAttr>}`	Returns the encrypted value of the <*ViewInstance*>.getAttribute("<*ViewAttr*>") statement.
`${oa.FunctionSecurity.<FunctionName>}`	Returns true or false depending on whether *FunctionName* is granted to the current user.
`${oa.message.<ProductCode>.<MessageName>}`	Evaluates to pageContext.getMessage("*<ProductCode>", "<MessageName>", null*); **Notes:** This EL type only supports FND_MESSAGEs without tokens.
`${oa.profile.<ProfileName>}`	Evaluates to pageContext.getProfile("*<ProfileName>*").
`${oa.request.<paramName>}`	Evaluates to pageContext.getParameter("*<paramName>*").
`${oa.session.<paramName>}`	Evaluates to pageContext.getSessionValue("*<paramName>*").
`${oa.sessiondirect.<paramName>}`	Evaluates to pageContext.getSessionValueDirect("*<paramName>*").
`${oa.txn.<paramName>}`	Evaluates to pageContext.getTransactionValue("*<paramName>*").
`${oa.txntrans.<paramName>}`	Evaluates to pageContext.getTransactionTransientValue("*<paramName>*").
`${oa.pageContext.<methodName>}`	Evaluates to pageContext.<*methodName*>(). **Note:** Only public methods without parameters may be invoked using this EL syntax.
`${oa.class.<className>.<methodName>}`	Invokes the method in the specified class. **Notes:** • Provide <*className*> as a fully qualified class name. • The class must have a no-argument (zero argument) constructor. • Only public methods without parameters may be invoked using this EL expression.

Using Operators in EL

You may use the following operators in an EL expression:

- +
- -
- *
- /
- %
- >
- >=
- <
- <=
- ==
- !=
- &&
- ||

Usage Notes:

- Use `oa.boolean.false` or `oa.boolean.true` to represent `Boolean.FALSE` or `Boolean.TRUE`, respectively.
- For the negation or `!` (NOT) operator, use `oa.boolean.false`, as in the example, `${oa.session.isItemRendered == oa.boolean.false}`.
- For an EL expression:
  - You may only specify a static numeric value.
  - Use `${oa.text.<TEXT>}` to represent a string.
  - Use `oa.boolean.false` or `oa.boolean.true` to represent a Boolean.

Sample Usage

```
${oa.session.param1 == oa.request.param1}
${oa.session.paramLength + ( oa.request.fieldLength * 2 ) }
${oa.session.isItemRendered == oa.boolean.false}
${oa.current.customerType == "Organization"}
```

Combining Static Text with EL

Use the syntax `${oa.text.<TEXT>}` to prepend or append static text to an EL expression.

Usage Notes

- **Warning:** If you append static text to an EL expression, be aware that the static text will not get translated and the order of the text may vary depending on the language.
- You may append text to an EL expression in any order, as shown in these examples, where ${EL} represents any valid EL type:
  - `${oa.text.<TEXT>} + ${EL}`
  - `${EL} + ${oa.text.<TEXT>} + ${EL}`

Sample Usage

```
${oa.text.Welcome Mr.} + ${oa.EmpVO.Empname}
```

EL-Supporting Properties

The following region and/or item properties support EL:

Property	Attribute Data Type	Valid Values
Required	string	• yes • no • uiOnly • validaterOnly
Read Only	boolean	• Boolean.TRUE • Boolean.FALSE
Rendered	boolean	• Boolean.TRUE • Boolean.FALSE

Disabled	boolean	• Boolean.TRUE • Boolean.FALSE
Prompt	string	• Any String value
Sort Allowed	boolean	• Boolean.TRUE • Boolean.FALSE **Note**: EL support for the Sort Allowed property is available only for items and not for regions.
Maximum Length	number	• Integer.intValue()

All properties listed above, with the exception of Disabled, may be personalized. Refer to the Personalizing EL-supporting Properties topic in the *OA Framework Personalization Guide* for additional information.

Related Information

- BLAF UI Guidelines
- OA Framework Developer's Guide
  - Changing UI Properties
- Javadoc

File Upload and Download

Overview

The File Upload feature enables the uploading of a file from a client machine to the middle tier, and is implemented by `oracle.apps.fnd.framework.webui.beans.message.OAMessageFileUploadBean`. The File Download feature enables downloading of a file from the middle tier to a client machine and is implemented by `oracle.apps.fnd.framework.webui.beans.message.OAMessageDownloadBean`.

Contents

- File Upload
  - Declarative Implementation
  - Runtime Control
- File Download
  - Declarative Implementation
  - Runtime Control
- Usage Notes
- Personalization Considerations
- Known Issues
- Related Information

File Upload

You can now specify a view object instance and view attribute for the messageFileUpload web bean and associate a data type to it. The File Upload feature appears as an input field with a prompt, followed by a Browse button, as shown:



If the view attribute returns a non-null value, that is, a file is already uploaded, OA Framework renders the File Upload feature as a View link with a prompt, followed by a Clear button:



You can select the View link to see information about the uploaded file. If you select Clear, the feature clears the View link and redisplays an input field with a Browse button so you can specify a new file to upload.

Note: You can alter the text of the View link to display some other text or the file name of an already uploaded file. See the Runtime Control section for more details.

Note: You can set the profile option called UPLOAD_FILE_SIZE_LIMIT to specify the maximum size of the file a user can upload. For example, if you set UPLOAD_FILE_SIZE_LIMIT to 500K, then during the http POST request, OA Framework reads only up to 500K from the stream and throws an exception if the uploaded file is larger than 500K.

AntiSamy Support and Attaching HTML Files

Previously, OA Framework handled the uploading of files as a byte stream, and could upload any file regardless of its character encoding. Now with the introduction of AntiSamy HTML sanitizing support in Oracle E-Business Suite Critical Patch Updates (CPU) for October 2011, January 2012, April 2012 and July 2012, if you upload an HTML file you must do the following:

1. Ensure that a `<meta>` tag for "Content-Type" exists in that HTML file with the correct character encoding name. For example:

   ```
   <meta http-equiv="Content-Type" content="text/html;charset=UTF-8">
   ```

2. If you cannot insert the `<meta>` tag into the HTML file, you must ensure the FND_NATIVE_CLIENT_ENCODING profile option value is set to the character encoding of the HTML file being uploaded.

AntiSamy is an API provided by the Open Web Application Security Project (OWASP) to remove malicious pieces of code from HTML files. OA Framework automatically applies this API if a file being uploaded has a "Content-Type" of "text/html". As mentioned earlier, OA Framework typically handles file uploads as a byte stream, but the AntiSamy API requires the HTML file to be scanned as a string. As a result, OA Framework must convert the contents of the HTML file to a string. To ensure proper conversion, OA Framework must know the character encoding information of the file being uploaded to avoid corrupting non-ascii characters in the HTML. For more information about the AntiSamy check, refer to *Security Configuration Mechanism in the Attachments Feature in Oracle E-Business Suite My Oracle Support* (formerly Oracle *MetaLink*) Knowledge Document 1357849.1.

Declarative Implementation

Perform the following steps to implement the File Upload feature declaratively in an OA Extension page.

Step 1: Create a region in your page layout region, with the **`Form`** property set to **true** for the page layout.

Step 2: In the new region, create an item of item style **messageFileUpload**.

Step 3: In the OA Extension Property Inspector, set the following properties for the messageFileUpload item:

- **`View Instance`** - A view object instance of the underlying data source.
- **`View Attribute`** - A view attribute in the specified view object instance, that maps to the column for storing the file content.
- **`Data Type`** - The Oracle data type of the view attribute. The BLOB data type is supported for File Upload. For example, if you set the data type to BLOB, the view attribute must map to a column whose data type is also BLOB.

  Note: The data type must be set to the same data type as the column that the view attribute maps to, otherwise an error occurs when the user attempts to commit the file upload.

  Note: If you set `Data Type` to **BLOB** and you store your file content in `FND_LOBS`, be sure to populate the column `FILE_CONTENT_TYPE` (File MIME Type). Since FILE_CONTENT_TYPE is a non-null column, you will encounter an error is this column is not populated. Refer to the `oracle.apps.fnd.framework.webui.beans.message.OAMessageFileUploadBean` Javadoc for additional information.

- **`Prompt`** - The text label for the File Upload feature.

Step 4: As of Release 12.2, you can enable virus scanning on all files uploaded through the messageFileUpload item. Control virus scanning by setting the `Virus Scan Enabled` property on the messageFileUpload item and the profile option FND: Disable Virus Scan as shown in the table below:

"FND: Disable Virus Scan" Profile Value	**`Virus Scan Enabled`** Property Value	Scan Done?
No	Yes	Yes
No	No	Yes
Yes	Yes	Yes
Yes	No	No

Note: In order for virus scanning to work properly, the following profiles must also be set appropriately:

- FND Attachment Antivirus Software Type
- FND Attachment Antivirus Server

The anti-virus software repairs the file if a virus is detected before uploading the repaired file. If the file can not be repaired, an error message displays about the infected file and the file is not uploaded.

Runtime Control

You can programmatically alter the text of the link that appears when a file has already been uploaded to be something other than "View". To change it to some other static text, then in your controller code, call the **`setDisplayName`** method from the **`OAMessageFileUploadBean`** and pass in the text to display for the link.

If you wish the link text to be dynamically determined, for example, to display the name of the file that is already uploaded, then you can data bind the display name as follows:

```
OADataBoundValueViewObject displayNameBoundValue =
  new OADataBoundValueViewObject(uploadBean, "FileName");
uploadBean.setAttributeValue(DOWNLOAD_FILE_NAME,displayNameBoundValue);
```

File Download

The File Download feature appears as linked text on a page. For example, in the figure below, a default single column region contains a messageTextInput item (File), followed by a messageDownload item that appears as a File Download link (setup.htm). The text that appears for the File Download link is the value returned by the View Attribute specified for the messageDownload item. When you select the file download link, a small window opens in your Browser. You can either open the file and display the content or save the file. If you choose Save, the file is created and saved to your client machine.



Declarative Implementation

Perform the following steps to implement the File Download feature declaratively in an OA Extension page.

Step 1: Create a region in your page layout region, with the **`Form`** property set to **true** for the page layout.

Step 2: In the new region, create an item of item style **messageDownload**.

Note: If you implement a messageDownload item in a table or advanced table region, the view object used to render the table or advanced table must have a designated primary key, otherwise the messageDownload web bean will repeatedly download content from the first row.

Step 3: In the OA Extension Property Inspector, set the following properties for the messageDownload item:

- **`View Instance`** - The view object instance of the underlying data source.
- **`View Attribute`** - The view attribute that maps to a column in the underlying data source.
- **`File View Attribute`** - The view attribute that maps to the column that stores the file content.
- **`File Name Override`** - The file name to save to when you select the File Download link and choose the Save option in the File Download window to save the file. The default file name that appears in the File Name field of the Save As dialog window is derived from the value returned from the view
  attribute specified by the View Attribute property. The value of the File Name Override property overrides that default file name and is especially useful if the view attribute returns instructional text, such as "Click on this link to download the file". If the File Name Override property is not defined, then
  the file name to save to is the value returned from the view attribute.
- **`File MIME Type`** - The MIME type of the file. See the Runtime Control example below if you do not want to specify a static value for this property.
- **`Data Type`** - The data type of the File View Attribute. The BLOB datatype is supported for File Download.
- **`Prompt`** - The text prompt that proceeds the File Download link.

Runtime Control

If the file MIME type is stored in a view attribute, you can retrieve it through a data bound value programmatically. The following code example illustrates how this is done:

```
// if content type is stored in a view attribute, it can be retreived through
// data bound value. Otherwise, a static value can also be set:
// e.g. downloadBean.setFileContentType("text/html")
OADataBoundValueViewObject contentBoundValue = new OADataBoundValueViewObject(downloadBean,
  "FileContentType");
downloadBean.setAttributeValue(FILE_CONTENT_TYPE, contentBoundValue);
```

Usage Notes

- Avoid defining both a messageFileUpload and a messageDownload item in a region and mapping both items to the same view attribute. If you map both items to the same view attribute, the Clear button in the messageFileUpload web bean that clears the View link, will also clear the link for the messageDownload web bean.
- A messageFileUpload item is not supported in a Table-in-Table region or Advanced Table-in-Advanced Table region.

Personalization Considerations

There are no personalization restrictions.

Known Issues

See a summary of key file upload/download issues with suggested workarounds if available.

Related Information

- BLAF UI Guidelines
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageDownloadBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageFileUploadBean`

Flexfields

Overview

A flexfield is a placeholder set of fields that can be configured by customers for use by their organizations. Once configured, the customer-defined fields in the set (label/widget pairs) may appear in either a form or tabular layout. Each field in the set has a name and a set of valid values. There are two types of flexfields, key and descriptive.

Key flexfields provide a way for Oracle E-Business Suite to represent objects such as accounting codes, part numbers, or job descriptions, which comprise of multiple fields (or segments) as a single object of concatenated segments. For example, the Key Accounting Flexfield is a key flexfield that represents the multiple accounting codes throughout Oracle E-Business Suite.

Similarly, descriptive flexfields provide a flexible way for Oracle E-Business Suite to provide customizable "expansion space" in applications, as well as provide a way to implement context-sensitive fields that appear only when needed. In essence, descriptive flexfields allow customizations of pages without writing either XML or Java and are configured as a set of fields on a page, much like the fields of the core application.

Both types of flexfields let you customize Oracle E-Business Suite features without programming, and these customizations are fully supported within Oracle E-Business Suite.

Note: This document assumes a working knowledge of flexfields. If you are not familiar with flexfields or need further information, please consult the *Oracle E-Business Suite Flexfields Guide Release 12* and Chapter 14 ("Flexfields") in the *Oracle E-Business Suite Developer's Guide Release 12*.

Contents

This document contains the following topics:

- Overview
  - Flexfield Features Supported and Unsupported by OA Framework
- Descriptive Flexfields
  - Setting Up a Descriptive Flexfield in Oracle E-Business Suite
  - Declarative Implementation in JDeveloper
    - Descriptive Flexfield Segment List
  - Runtime Control
  - Descriptive Flexfield Usage Restrictions
  - Personalization Considerations
- Key Flexfields
  - Key Flexfield UI in a Table
  - Key Flexfield Combination LOV UI
  - KFF Combinations LOV in Popup
  - KFF Combinations LOV Look Ahead Search
  - Dynamic Insertion
  - Required Segments
  - PPR Support
  - Setting Up a Key Flexfield in Oracle E-Business Suite
  - Declarative Implementation in JDeveloper
    - Key Flexfield Segment List
  - Runtime Control
  - Personalization Considerations
- Reference Fields
- Values and Value Sets
- Flexfield Validation
- Flexfields and Standard Request Submission
- Developer Mode Errors
- Troubleshooting Flexfield Problems
- Known Issues
- Related Information

Flexfield Features Supported and Not Supported by OA Framework

The following table lists the flexfield functionality (above and beyond regular flexfield functionality) that is supported and not supported by OA Framework. The list also provides links to the area of this document where the supported or unsupported feature is discussed in more detail.

Note: Warnings are raised at setup time in the Forms-based flexfield administration screens if a flexfield is using features not supported in OA Framework.

Flexfield Features	Supported	Not Supported
General	• Specifying segments for a particular context of a descriptive flexfield • Hiding certain segments for Key and Descriptive flexfields • Read-only segments for Key and Descriptive Flexfields • Descriptive flexfield default values • Using Reference fields	• Query a range of key flexfield segments

Value Sets

- Format Rules
- Displaying as Poplist
- Displaying as LOV
- Using a $FLEX$ reference
- Using a $PROFILES$ reference

Validation Types

- None
- Independent
- Dependent
- Table
- Special
- Pair

Key Flexfield-specific

- Dynamic Insertion
- Key flexfield within a table
- Multiple key flexfields on a page
- Multiple key flexfields in a table
- Generating the CCID
- Combination LOV UI
  - Required key flexfield LOV
  - Dynamic Insert
  - LOV over LOV
- Adding a VRule to a Key Flexfield web bean
- Date Validation
- Setting additional WHERE clause
  - static string
  - dynamic string
  - dynamically binded on a per row basis for tables
- Specifying a segment description

Descriptive Flexfield-specific

- Context values
- Context switching
- Setting context value through an API
- Descriptive flexfield within a table
- Multiple descriptive flexfields on a page
- Multiple descriptive flexfields in a table
- Setting a descriptive flexfield to be Read-only
- Hiding or showing the context
- Search
- Specifying a segment description

Flexfield Core UI

- Date picker - supported for both key and descriptive flexfields.
- Multiple key or descriptive flexfields using the same view object.
- Programmatic implementation of descriptive and key flexfields
- Merging a flexfield with its parent layout (see workarounds for key and descriptive flexfields)

Flexfield Validation

- Turning On/Off unvalidated submit

Flexfield Customization

- Overriding an descriptive flexfield segment LOV to populate multiple descriptive flexfield segments with values.

Segments lists

- enable/disable, reorder and mark as read-only for Descriptive and Key flexfields
- Read-only and Rendered bound for descriptive and key flexfields

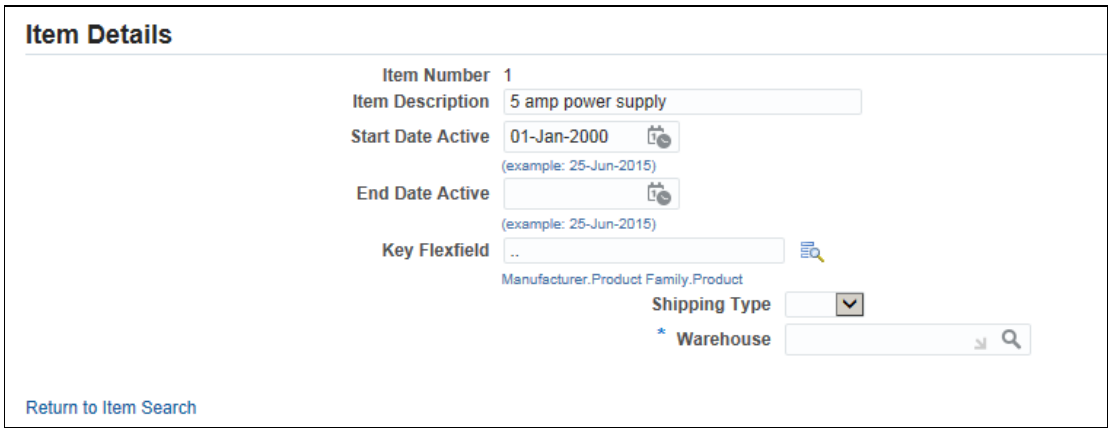
Flexfields with other components	• Query web bean (turning validation off) • Descriptive flexfield or key flexfield in a messageLayout region • Embedded mode - flexfields are rendered when its parent page is embedded in a JTT application. • HGrid • Table Detail Disclosure	• Descriptive flexfield in a Master/Detail page • Table-in-Table - a developer mode error results if you try to implement a flexfield in a Table-in-Table.
PPR support	• Descriptive flexfield context switching • Dependent LOV PPR • Use descriptive flexfield or key flexfield as a PPR target	
Exporting	• Descriptive flexfield data • Key flexfield data	

Descriptive Flexfields

A descriptive flexfield is implemented as an `oracle.apps.fnd.framework.webui.beans.OADescriptiveFlexBean`. An `OADescriptiveFlexBean` automatically renders a layout for user input of segment values**.** For each descriptive flexfield, if the `Display Context` property is set to **True**, a **context** field renders as a poplist in the first row. After the first row, any global segments defined are rendered, followed by the context-sensitive segments that correspond to the selected value in the **context** poplist.

Each descriptive flexfield segment has a prompt aligned to the right, and a corresponding input field aligned to the left. Figure 1 below is an example of the flexfield UI for a standard vertical layout:

Figure 1: Visual Example of both a Key and Descriptive Flexfield on a Page



The above example displays a descriptive flexfield. **Shipping Type** is the context field for the descriptive flexfield and **Warehouse** is a global field. No context-sensitive elements are displayed in this example because a context has yet to be selected.

Currently flexfields support three types of input style:

- Text Input (not shown above)
- PopList, as shown for the segment **Shipping Type**
- LOV, as shown for the segment **Warehouse**

When you add an `OADescriptiveFlexBean` to your page, it:

- Displays flexfield segments that allow input or update and may populate segments with database values from corresponding view objects.
- Automatically refreshes with corresponding flexfield segments when a new context is selected.
- Validates flexfield segment input values.
- Automatically transfers valid values to the view object so that the calling page has access to valid flexfield values if no errors exist. If errors exist, the current page is redrawn with corresponding error messages.
- If you need to wrap a long text of the MessageTextInput of a descriptive flexfield bean to fit the page, you can now call setWrapEnabled() on the descriptive flexfield bean before calling processFlex() by using the following command: `dffBean.setWrapEnabled(true);` Wrapping is disabled by default.

PPR Support

OA Framework provides PPR support for descriptive flexfields in the following ways:

- If you change the context of a descriptive flexfield, OA Framework uses PPR to render the segments for the new context.

  Note: OA Framework will perform a page refresh rather than PPR if your controller code moves the segment web beans of the descriptive flexfield

out of the descriptive flexfield container.

- If a PPR action occurs on a page, and any view object attribute for the descriptive flexfield is changed during the processFormRequest method, OA Framework automatically adds the flexfield web bean as a target for PPR and re-renders the descriptive flexfield.

  Note: If the context attribute for the descriptive flexfield is changed during the processFormRequest method, the flexfield web bean is not added as a target for PPR and you therefore will not see a change in the descriptive flexfield structure. To show the structure change, OA Framework must redirect back to the same page, in which case, you may need to add code to your processRequest method if this is a concern.

Setting Up a Descriptive Flexfield in Oracle E-Business Suite

Before you can add a descriptive flexfield to an OA Framework page, you must first set up the descriptive flexfield in Oracle E-Business Suite. To start, review Chapter 3 ("Planning and Defining Descriptive Flexfields") in the *Oracle E-Business Suite Flexfields Guide Release 12.*

When you have a clear plan for the descriptive flexfield you wish to set up, refer to the section titled "Implementing Descriptive Flexfields" in Chapter 14 ("Flexfields") of the *Oracle E-Business Suite Developer's Guide Release 12* for instructions to the following general steps:

Step 1: Define descriptive flexfield columns in your database.

Step 2: Register your descriptive flexfield table with Oracle Application Object Library.

Step 3: Register your descriptive flexfield using the **Descriptive Flexfields Window**.

Refer to the *Oracle E-Business Suite Flexfields Guide Release 12* for instructions relating to the last of the general steps:

Step 4: Define your value sets in the **Value Sets Window**, as described in Chapter 5 ("Values and Value Sets").

Step 5: Define your descriptive flexfield structure using the **Descriptive Flexfield Segments Window**, as described in the "Descriptive Flexfield Segments Window" section of Chapter 3 ("Planning and Defining Descriptive Flexfields").

Recall that the value of a descriptive flexfield context field determines the context of the descriptive flexfield and the context-sensitive segments (if any) that are displayed. The section titled "Context Fields and Reference Fields" in Chapter 3 ("Planning and Defining Descriptive Flexfields") discusses context fields in more detail.

Step 6: When you are ready to add the descriptive flexfield to an OA Framework page, follow the steps outlined in the Declarative Implementation and Runtime Control sections below.

Declarative Implementation

The following steps describe how to add a descriptive flexfield item to a OA Framework region:

Step 1: Ensure that the view object underlying the region includes all the database columns necessary for this descriptive flexfield. You should not change the database column names for this flexfield because the `OADescriptiveFlexBean` uses the same naming convention that the view object generation routine uses to find the corresponding attribute names from your view object.

Step 2: Define an item of the `item style` **flex** in your region.

Note: You cannot create a flex item directly under a messageComponentLayout region, but you can create a messageLayout region under the messageComponentLayout region and add the flex item under the messageLayout region.

Step 3: Set the `Read Only` property to **True** or **False**, depending on whether you want the descriptive flexfield to be read only.

Step 4: Specify a `View Instance` for your flexfield. The view instance should be the same as the view instance (view object) specified for your region.

Note: OA Framework supports multiple descriptive flexfields on the same view object.

Note: If a flexfield's view object does not return a row, an OAException will not be thrown so that the controller's processRequest method can still execute and render the flexfield.

Step 5: Set the `Appl Short Name` property to the short name of the application to which the descriptive flexfield is registered. (Step 3 of Setting Up a Descriptive Flexfield in Oracle Applications).

Step 6: Set the `Name` property to the name of the descriptive flexfield as it was registered.

Note: This differs from how Key Flexfields are defined by shorthand codes.

Step 7: Set the `Type` property to **descriptive**.

Step 8: Set the `Segment List` property as appropriate (see Descriptive Flexfield Segment List).

Step 9: You may set the `Display Context Field` to **True** or **False**, depending on whether you want to hide or show the context for the descriptive flexfield.

Note: By default the Context Field of a read only descriptive flexfield will be hidden. It can be shown by setting the attribute OAWebBeanConstants.FLEXFIELD_READONLY_DISPLAY_CONTEXT to be true in the controller.

Step 9: Beginning in Release 12.2.5, you may render a descriptive flexfield within a table as a pop-up. The table displays an icon for each row, which upon selection, displays the descriptive flexfield in the pop-up. This usability enhancement allows the descriptive flexfield structure to vary

depending on each table row's context. After you follow Steps 1 through 8 to define the **flex** item under the appropriate element of your table or advanced table region, set the `Render as Popup` property on the **flex** item to **True**.

Descriptive Flexfield Segment List

If you leave the `Segment List` property empty, all segments render. The value you specify for this property must use the following format:

```
Global Data Elements|[global segment1]|[global segment2]||...||Context1|[segment1 for context1]|
[segment2 for context1]||...||Context2|....
```

For our example shown in Figure 1, **Shipping Type** is the context field and **Warehouse** is the global field for the descriptive flexfield. To always display the **Warehouse** global field and display the context-sensitive fields **Items per Box** and **Box Size** for the **Box** context and **Items per Pallet** and **Pallet Weight** for the **Pallet** context, you would specify the following value for the `Segment List` property:

```
Global Data Elements|Warehouse||Box|Items per Box|Box Size||Pallet|Items per Pallet|Pallet Weight
```

As shown, segments within a certain context are separated by a single pipe, "|", while data from a different context is separated by a double pipe, "||".

Read-Only Segments

You can also add the read-only token (RO) after any of the segments in the list. For example, the (RO) designator below sets Segment1 to be read-only:

```
Context1|Segment1($RO$)|Segment2...
```

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically-created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

If you have declaratively created a descriptive flexfield and want to autorender the whole descriptive flexfield structure, you do not have to write any extra code in your controller's `processRequest` method.

If you need to programmatically create your descriptive flexfield, you can include code similar to the example below in your controller's `processRequest` method. In this example, DescFF is the item name (of the `item with the style` **flex**) in your region:

```
OADescriptiveFlexBean dffBean = (OADescriptiveFlexBean)
  createWebBean(pageContext, DESCRIPTIVE_FLEX_BEAN, null, "DescFF");
webBean.addIndexedChild(dffBean);
dffBean.setAttributeValue(OAWebBeanConstants.VIEW_USAGE_NAME,"FlextestVO1");
dffBean.setAttributeValue(OAWebBeanConstants.FLEXFIELD_APPLICATION_SHORT_NAME,
  "FND");
dffBean.setAttributeValue(OAWebBeanConstants.REGION_APPLICATION_ID,
  new Integer(0));
dffBean.setAttributeValue(OAWebBeanConstants.FLEXFIELD_NAME,"SimpleFlex");
//As of Release 12.2.5, you can also render the Descriptive Flexfield as a pop-up within a table.
dffBean.setRenderAsPopup(true);
```

In the `processFormRequest` method, you can also get valid descriptive flexfield data from your view object's corresponding attributes without any extra coding.

You should also consider these other descriptive flexfield runtime control topics:

- Merging Descriptive Flexfield Segments with the Parent Layout
- Altering Flexfield UI Layout
- processRequest Method
- Overwriting the Descriptive Flexfield Context Value
- Descriptive Flexfield in a Master/Detail Page
- Descriptive Flexfield in a Search Region
- Read Only and Rendered Bound Values
- Populating a Descriptive Flexfield with Default Values
- Overriding a Descriptive Flexfield Segment LOV

Merging Descriptive Flexfield Segments with the Parent Layout

By default, flexfield segments are aligned within themselves but they are not aligned within the whole (parent) region. If you want to merge descriptive flexfield segments to the parent layout, you must find the `OADescriptiveFlexBean by attribute` and call the method `mergeSegmentsWithParent` on `oracle.apps.fnd.framework.webui.beans.OADescriptiveFlexBean` in your controller's `processRequest` method.

The following code example merges descriptive flexfield segments to the parent layout:

```
public class RegionCO extends OAControllerImpl
{
```

```
  public void processRequest(OAPageContext pageContext, OAWebBean webBean)
  {
      super.processRequest(pageContext, webBean);

      //find the flexfield that is defined in this region as the item "DescFF" and merge each
      //individual segment to the outside layout

      OADescriptiveFlexBean dffBean = (OADescriptiveFlexBean)webBean.findIndexedChildRecursive("DescFF");
      flexBean.mergeSegmentsWithParent(pageContext);
  }

  public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
  {
      super.processFormRequest(pageContext, webBean);
   }
}
```

For example, you could add the following line to the end of the "create descriptive flexfield" code example above:

```
dffBean.mergeSegmentsWithParent(pageContext);
```

Altering the Flexfield UI Layout

Note: This is not recommended by OA Framework Development and may cause problems with PPR.

If you want to do something unusual with the descriptive flexfield UI layout (like insert a button after every segment or move the segments around), you need to follow these steps:

1. Find the `OADescriptiveFlexBean` by attribute name and call the `processFlex` method on `OADescriptiveFlexBean`. After this, you will be creating the flexfield segment web beans based on metadata and the view object attribute.
2. Call the `getIndexedChild(int)` method on the flexfield web bean to go through all the child of the flexfield and alter the layout as desired.

processRequest Method

If you change a descriptive flexfield's context value, the descriptive flexfield code must redirect back to the same page. As a result, the controller `processRequest` method is called again. If this behavior causes problems on the page, please use the following code in your `processRequest` method:

```
  public void processRequest(OAPageContext pageContext,
    OAWebBean webBean)
  {String formEvent = pageContext.getParameter(FLEX_FORM_EVENT);
    if (formEvent == null )
      {
       //processRequest ...
      }
  }
```

Overwriting the Descriptive Flexfield Context Value

If you need to overwrite the descriptive flexfield context value with a value other than the context derived from the view object, you can do so programmatically using the `setFlexContext` method on `OADescriptiveFlexBean`.

Descriptive Flexfield in a Master/Detail Page

Suppose you implement a Master/Detail page where, if a user selects the single selection radio button to select a row in the master region, a PPR event is issued. The page's controller then executes the `processFormRequest` method to detect the radio button selection and invokes a method in the application module to mark that row in the master data set as the current row in the detail region.

If you want to add a descriptive flexfield to the detail region, you update your controller code with the following workaround to ensure that the descriptive flexfield gets updated with the correct context structure when a different row in the master table is selected:

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)

{
 super.processFormRequest(pageContext, webBean);
 pageContext.getApplicationModule(webBean).invokeMethod("refreshDetail");
 pageContext.setForwardURLToCurrentPage(null,true,null,(byte)0);
}

public void refreshDetail()
{
 Row[] rows = getFndApplicationVO().getFilteredRows("selection","Y");
 if (rows != null)
 {
  getFndApplicationVO().setCurrentRow(rows[0]);
 }
}
```

Descriptive Flexfield in a Search Region

Currently if a Search region includes a descriptive flexfield that has a segment of type LOV, any search criteria that you specify in that flexfield LOV will be auto-validated. LOVs, however, should not be validated in a search criteria region. To turn validation off, use the following API in `OADescriptiveFlexBean`:

```
public void setUnvalidated(boolean unvalidated);
```

If the `unvalidated` value for this method is `true`, OA Framework will disable LOV validation and disable server-side validation for the descriptive flexfield. See Flexfield Validation for more details.

Read Only and Rendered Bound Values

If you have to implement a flexfield in a table, where the flexfield is read-only or rendered in some rows and not others, you can do so programmatically using bound values. There is currently no declarative support for this.

First, create your bound value using `OADataBoundValueViewObject` or some other boundValue class. Then set the attribute in your controller's `processRequest` method as shown in the examples below:

```
dffBean.setAttributeValue(READ_ONLY_ATTR,boundValue);
```

or

```
dffBean.setAttributeValue(RENDERED_ATTR,boundValue);
```

Populating a Descriptive Flexfield with Default Values

Previously, if you wanted display the default values of segments when a descriptive flexfield rendered, you had to use the `populateRowWithDFFDefaultValues` API in `oracle.apps.fnd.framework.OAFlexUtils`. This API would populate a view object row with the descriptive flexfield default values. Now, OA Framework displays the default values automatically in the descriptive flexfield whenever you create a new row. If the current row is loaded from the database, the descriptive flexfield displays the values stored in the database.

Overriding a Descriptive Flexfield Segment LOV

In OA Framework, the LOV for a descriptive flexfield segment is automatically mapped to the SQL criteria and results for that single segment. However, there may be cases when you want to associate multiple segments in a descriptive flexfield to the LOV results of a single segment. In other words, you may want to override a segment LOV such that when a user selects a value from that segment's LOV, the LOV returns result values for all the other associated flexfield segments as well. You can override a segment LOV programmatically, using the following method on `OADescriptiveFlexBean`:

```
public void setFlexLovOverrideInfo(Dictionary[] flexLovInfo)
```

The `flexLovInfo` parameter is an array of dictionaries. The dictionaries contain the information necessary to override the flexfield LOV relations so that the segment LOV also returns results to multiple other segments of that descriptive flexfield in the base page. Each dictionary contains the following properties, which are explained in detail in the Javadoc:

- `FLEX_LOV_OVERRIDE_SEGMENT` - the database column name of the flexfield segment whose LOV is to be overridden.
- `FLEX_LOV_OVERRIDE_RESULTS` - a string array of the database column names of the flexfield segments that you want to associate with the LOV to override, to return results.
- `FLEX_LOV_OVERRIDE_VIEWUSAGE` - view usage name for the view object containing the SQL to override the LOV segment.
- `FLEX_LOV_OVERRIDE_AM` - full path of the application module containing the view object specified by `FLEX_LOV_OVERRIDE_VIEWUSAGE`.

The following code example illustrates how this API is used:

```
//Create a dictionary array and use arrayMap for each dictionary
  Dictionary[] flexLovOverrideInfo = new Dictionary[1];
  for (int i = 0; i< flexLovOverrideInfo.length; i++)
  {
    flexLovOverrideInfo[i] = new ArrayMap(4);
  }

//Specify AM and VO name for the overriden LOV
  flexLovOverrideInfo[0].put(FLEX_LOV_OVERRIDE_VIEWUSAGE,
    "USCityLovVO");
  flexLovOverrideInfo[0].put(FLEX_LOV_OVERRIDE_AM,
    "oracle.apps.per.selfservice.personalinformation.server.AddressUpdateAM");

//The LOV result will be returned to the following segments,
//and in the new LOV VO, there should be a corresponding column
//for each segment.  If the VO column name is different than the
//segment name, the name should be specified as "segmentName:lovColumnName".
//For example: "POSTAL_CODE:ZIP_CODE"
  String[] flexLovOverrideResults = new String[4];
  flexLovOverrideResults[0] = "TOWN_OR_CITY";
  flexLovOverrideResults[1] = "REGION_2";
  flexLovOverrideResults[2] = "POSTAL_CODE:ZIP_CODE";
  flexLovOverrideResults[3] = "REGION_1";
```

```
  flexLovOverrideInfo[0].put(FLEX_LOV_OVERRIDE_RESULTS,
  flexLovOverrideResults);

//The LOV will be attached on this segment:
  flexLovOverrideInfo[0].put(FLEX_LOV_OVERRIDE_SEGMENT,
    "TOWN_OR_CITY");

//Set the override info into flexfield web bean.
  flexBean.setFlexLovOverrideInfo(flexLovOverrideInfo);
```

Descriptive Flexfield Usage Restrictions

1. A descriptive flexfield is not supported in the table-in-table feature. See the Oracle Browser Look-and-Feel (BLAF) Guidelines: Flexfields and its Open Issues section for more details.
2. As of Release 12.2.5, if you must display descriptive flexfield information from a table, where the structure may vary depending on the context, you should implement the descriptive flexfield as a pop-up. This feature is only supported for descriptive flexfields in classic and advanced tables.
   A. To render a descriptive flexfield in a table as pop-up, do not call `OADescriptiveFlexfield.processFlex` in the controller's `processRequest` method.
   B. To programmatically enable a descriptive flexfield in a table as pop-up, you should only include `OADescriptiveBean.setRenderAsPopup(true);` in the controller of the table or the parent web bean of the table.
3. As of Release 12.2.5, to implement a descriptive flexfield as a pop-up in a non-table case, follow the instructions below that apply to your scenario:
   - Scenario 1: If you have declaratively enabled the Render as Popup property, call the method `mergeSegmentsWithParent` on `oracle.apps.fnd.framework.webui.beans.OADescriptiveFlexBean` in your controller's `processRequest` method.
   - Scenario 2: If you have implemented the descriptive flexfield pop-up programmatically, call the method `mergeSegmentsWithParent` AFTER calling `setRenderAsPopup(true);` on `oracle.apps.fnd.framework.webui.beans.OADescriptiveFlexBean` in your controller's `processRequest` method. Please be sure to make the calls in the order described.
4. Multiple descriptive flexfields are supported in a single table and hence a single view object, but there should be no overlap between the attributes that the different flexfields use, otherwise they will overwrite each other's values.

   Since you cannot change the view object attribute mapping that the flexfield web bean uses (for example, it automatically uses the Segment1 view object attribute if a segment uses the SEGMENT1 table column), you need to define a prefix for the other flexfields using the `setFlexPrefix` method on the `OADescriptiveFlexBean`.

   To illustrate, suppose descriptive flexfield DFF1 use `ATTRIBUTE1` and `ATTRIBUTE2`, which map to the view object attributes `Attribute1` and `Attribute2` and descriptive flexfield DFF2 use `ATTRIBUTE2` and `ATTRIBUTE3` which map to the view object attributes `Attribute2 and Attribute3`. Without a prefix, both flexfields update view object `Attribute2` so one of the values gets overwritten.

   If you call `DFF2Bean.setFlexPrefix("DFF2")`, the `DFF2 OADescriptiveFlexBean` will map to `DFF2Attribute2` and `DFF2Attribute3` and as a result, will no longer conflict with the attributes of the DFF1 `OADescriptiveFlexBean`.
5. Refer to known flexfield issues for other restrictions.

Personalization Considerations

- See a summary of Flexfields personalization considerations in the Oracle Application Framework Personalization Guide.
- The contents of a pop-up for a descriptive flexfield in a table are not personalizable.

Key Flexfields

A key flexfield is implemented as an `oracle.apps.fnd.framework.webui.beans.OAKeyFlexBean`. An `OAKeyFlexBean` automatically renders the layout for the input of segment values**.** Because a key flexfield does not have a context field, all the segments for the specified structure code render.

Recall that each key flexfield has a corresponding table, known as a combinations table, that the flexfield uses to store a list of the complete codes (one column for each segment of the code) along with the corresponding unique ID number (code combination ID or CCID) for that code. Pages whose underlying entity objects contain a foreign key reference to the combinations table are referred to as "foreign key pages", while the page whose underlying entity object uses the combinations table itself are referred to as "combinations pages" or "maintenance pages".

Note: the `OAKeyFlexBean` currently supports only "foreign key pages".

Additional Information: For further discussion of CCIDs, refer to the topics "Combination" and "Dynamic Insertion" in Chapter 2 ("Planning and Defining Key Flexfields") of *Oracle Applications Flexfields Guide Release 12.* See also the "Flexfields" chapter of *Oracle E-Business Suite Developer's Guide.*

When you add an `OAKeyFlexBean` to your page, it will:

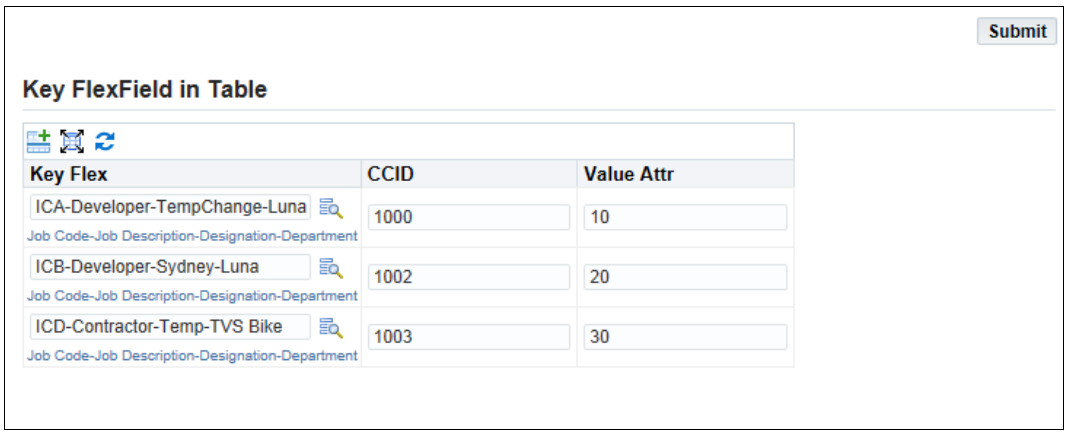
- Display flexfield segments for input or update and may populate flexfield segments with database values from corresponding view objects.
- Validate input values for flexfield segments and if such a combination already exists, uses that current combination's CCID to update the view object. If no such combination yet exists, it inserts a new (CCID) row to the combinations table.

- Automatically use the CCID to update the CCID attribute of the view object so that the calling page has access to those values if there are no errors. If there are errors, the current page is redrawn by OA Framework with corresponding error messages.

Key Flexfield UI in a Table

When a key flexfield is displayed in a table, the concatenated segment descriptions are also displayed, as shown in Figure 2.

Figure 2: Key Flexfield in a Table

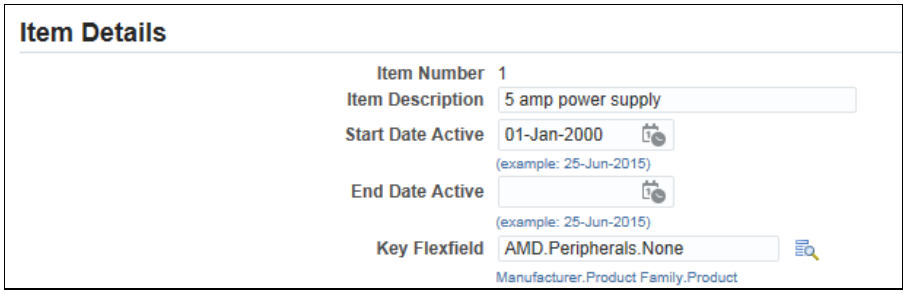


Key Flexfield Combination LOV UI

Based on the Oracle Browser Look-and-Feel (BLAF) Guidelines: Flexfields, key flexfields are implemented as code *combination LOVs* rather than as individual segments in the UI. You can type in a combination code directly in the code combination LOV input.

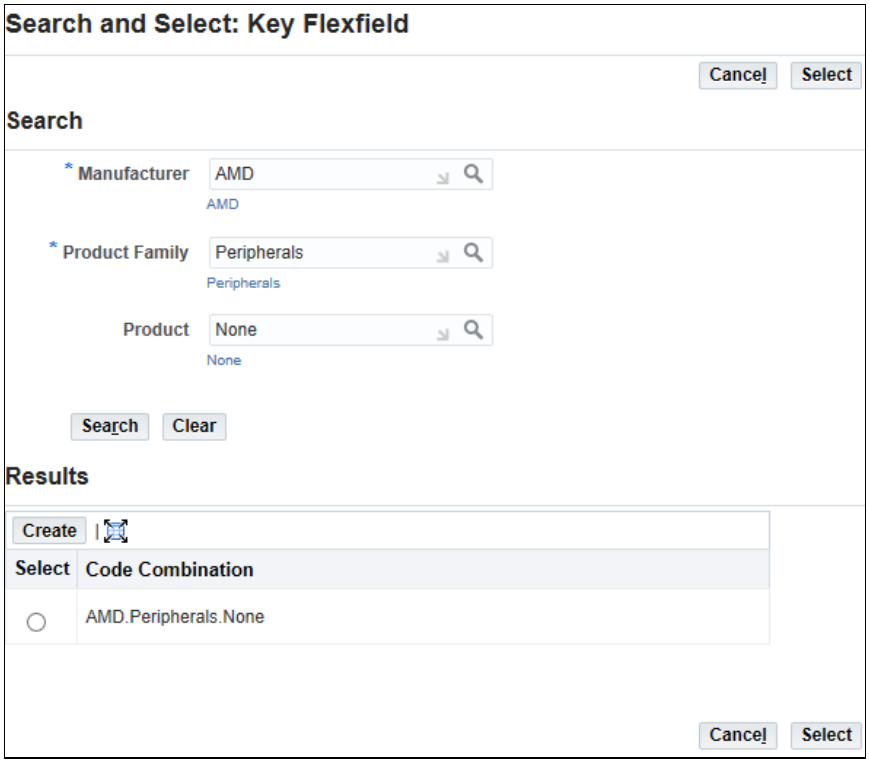
Using this new UI, you can also select the code combination LOV icon to the right of the input field, as shown in the Key Flexfield in Figure 3.

Figure 3: Combination LOV Key Flexfield on a Base Page



The resulting advanced Search page can be used to search for an existing key flexfield combination. The Search page is rendered as two regions, the Search region and the Result region. Each segment of the key flexfield is rendered in the Search region. You do not need to enter values for all segments when searching for a key flexfield code combination.

Figure 4: Search Page for Key Flexfield Combination LOV



Note: When defining a KFF search item in the search panel, the Flex item should have the "View Instance Name" property as empty, otherwise OA Framework will ignore this KFF item and skip processing.

Note: OA Framework currently does not support searching across a range of concatenated segments or individual segments and returns a developer mode error if specified. See the "Range form" topic in Chapter 2 ("Planning and Defining Key Flexfields") of *Oracle E-Business Suite Flexfields Guide Release 12* for more information about this feature in Oracle E-Business Suite.

Note: Backwards compatibility - The new key flexfield combination LOV UI generates one LOV web bean for the key flexfield instead of multiple child web beans for each segment of the flexfield. Some controllers may have dependencies on the old key flexfield UI. To fix this backwards-compatibility issue, you can turn off the new key flexfield UI by setting the FND_FWK_COMPATIBILITY_MODE profile to 11.5.9 or using the following Java API:

```
OAKeyFlexBean.useCodeCombinationLOV (boolean false);
```

KFF Combinations LOV in Popup

From Release 12.2.9, the KFF combinations LOV are by default displayed in a popup. Apart from the UI enhancement, there is also added PPR and event support on the original KFF combinations LOV.

This feature can be turned off by setting the profile option 'FND_ENABLE_OPEN_LOV_IN_POPUP' to 'False'. Be aware that this profile also controls a standard LOV behavior. By setting the above profile to 'False', the KFF combinations LOV will revert to the old behavior, and the LOV will be rendered inside a modal window.

Key characteristics of this transition are:

- Migration of KFF Combinations LOV from deprecated OAMessageLovTextInputBean to OAMessageLovInputBean.
- Layout of KFF combinations LOV window/popup is similar to before the migration. However, it has been enhanced to interact with base page with LOV events (for example, lovPrepare, lovValidate, and lovUpdate).
- Support LookAheadLOV for the combinations LOV lookup on base page.

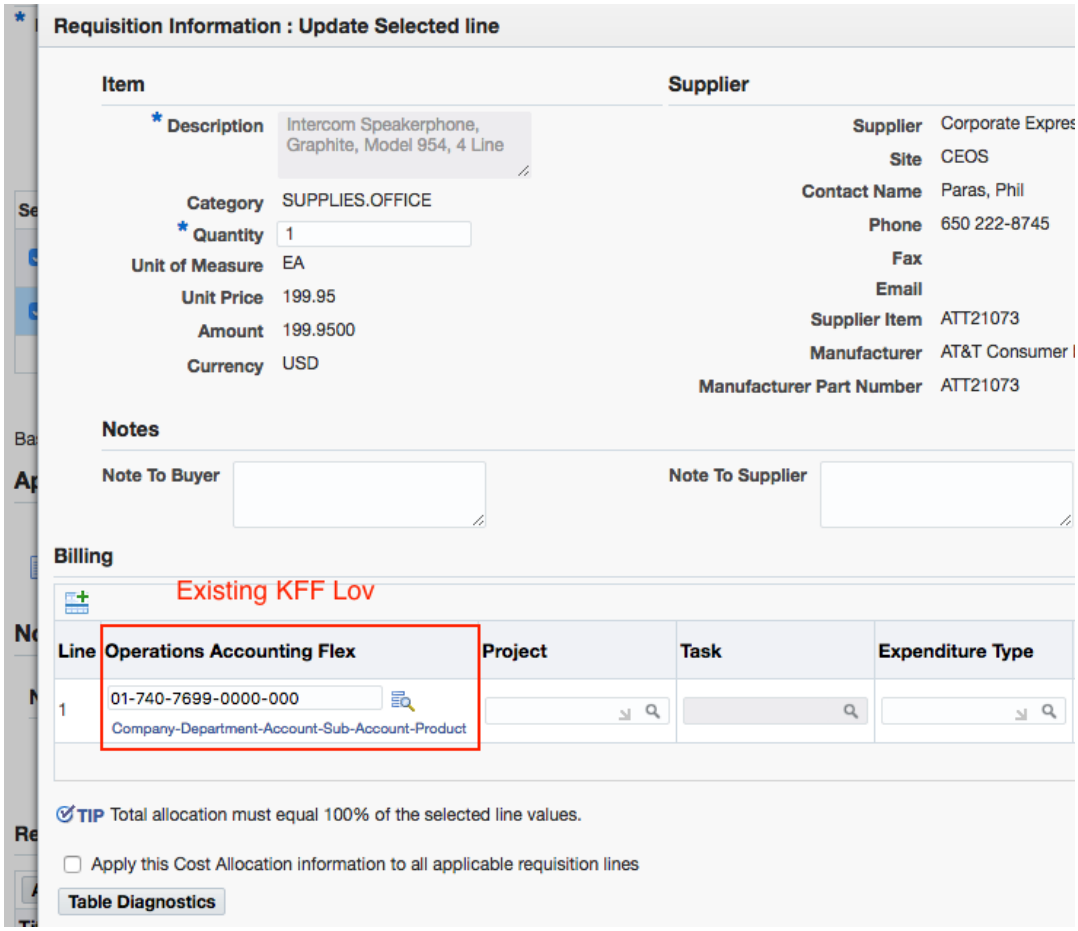
In summary, UI behavior of the KFF combinations LOV on base page is improved, and now has the consistent look and feel of a regular LOV. In addition, support has been added to the KFF combLov both for LOV events and lookAheadLOV, which also improves the user experience.

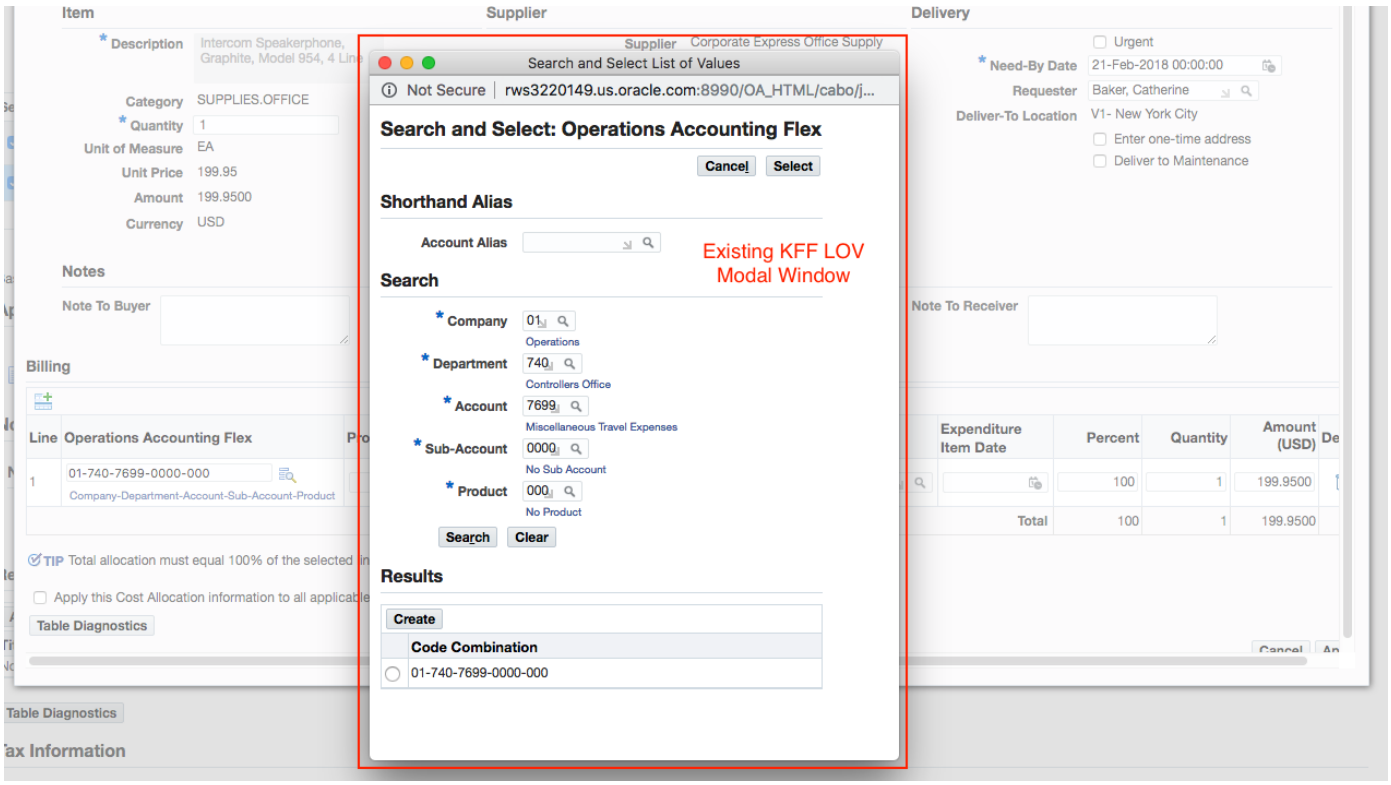
Setup is via the profile option mentioned: 'FND_ENABLE_OPEN_LOV_IN_POPUP' with a default value of 'True'. In addition, the profile option FND_FWK_COMPATIBLITY_MODE needs to be set to either '1.5.10' or 'Null' (for 11.5.9, the KFF was not rendered in combination LOV format: instead, each segment was rendered as individual LOV input on the base page).

In terms of programmatic controls, no code change is required to enable this feature. However, if any controller has invoked the API `KFFBean.useCodeCombinationLOV(false)` to disable the combinations LOV, this feature would not take effect on that KFF.

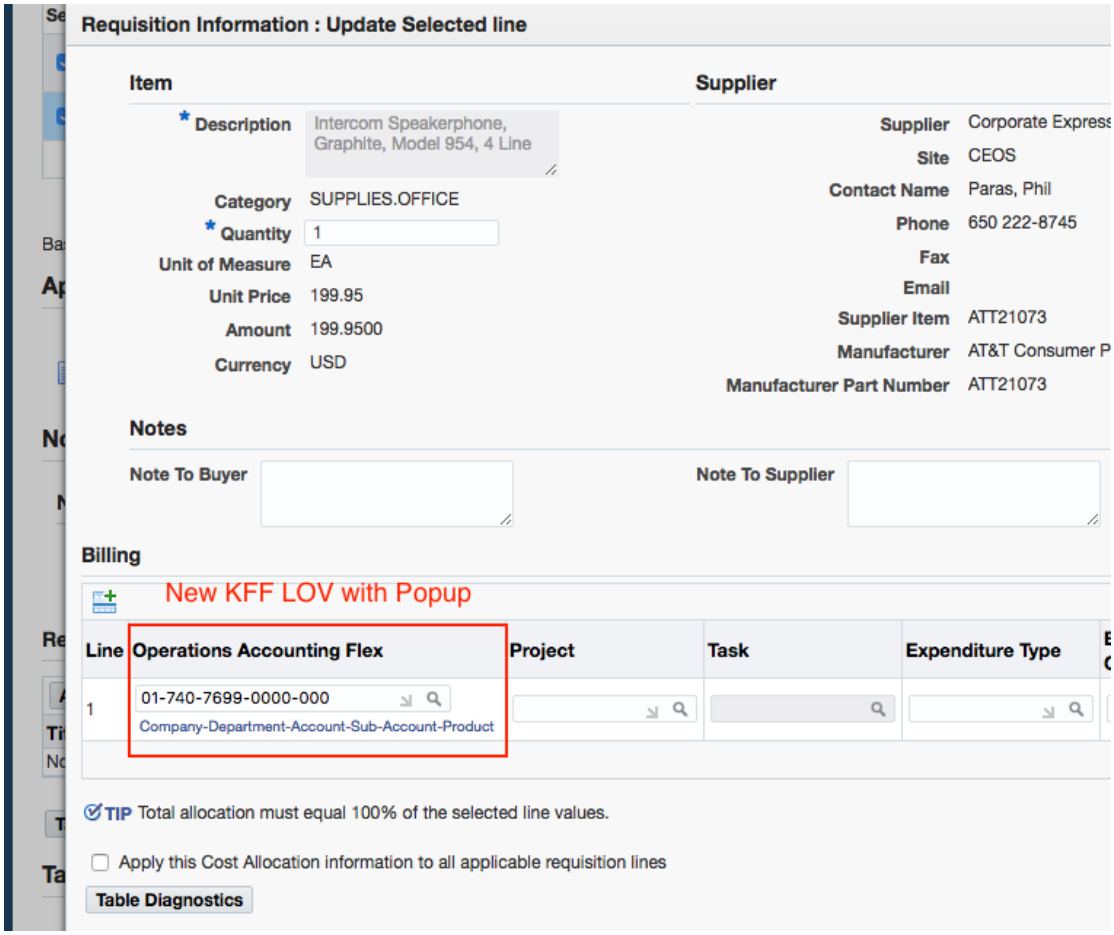
For a UI comparison, refer to the following screenshots.

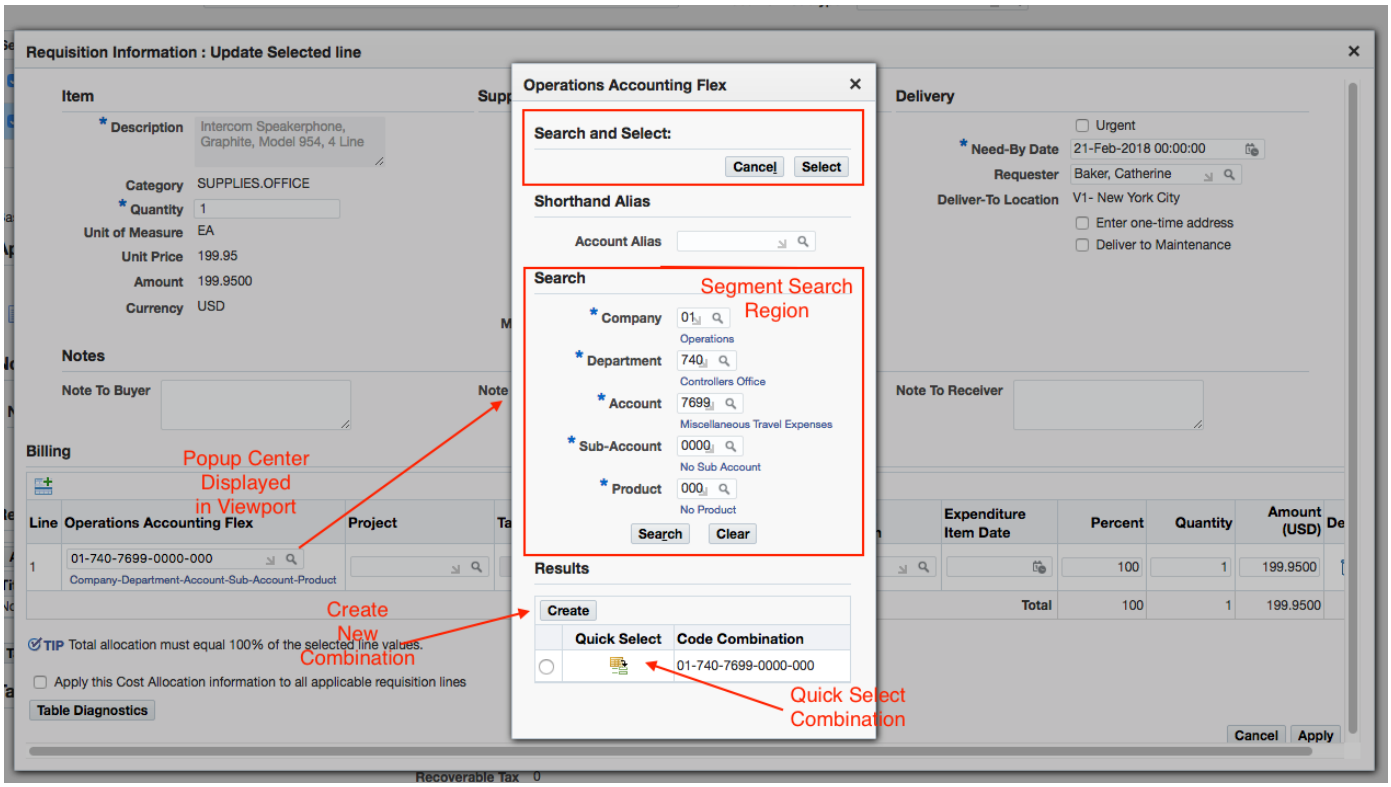
Previous KFF LOV UI





Migrated New KFF LOV UI





KFF Combinations LOV Look Ahead Search

Introduced in 12.2.9, this feature allows users to perform look ahead search on a KFF combination when typing in the base page LOV input field. The feature is only enabled when the KFF LOV is displayed in popup mode, for example when the profile option 'FND_ENABLE_OPEN_LOV_IN_POPUP' is set to 'True'. The user can type three characters to trigger the look ahead search on the KFF

combinations LOV.

This feature is enabled automatically whenever KFF LOV in Popup is enabled — no extra setup is required.

By default, the look ahead search will perform case sensitive search. However, there is a property on the FlexField item to control this behavior: LOOK_AHEAD_LOV_CASE_SENSITIVE_SEARCH. This can have the values True or False. The default value is Null (True).

Look Ahead Search Behavior

You need to type at least three characters to trigger a look ahead search.

Important search scenarios include:

- Case sensitive search is performed on KFF combinations value.
  - A personalizable property, LOOK_AHEAD_LOV_CASE_SENSITIVE_SEARCH, can be used to control case sensitivity of the search.
- Search can be by individual or multiple segment values:
  - For Individual Segment Value, the user can type in a specific segment value or partial segment value.
  - For Search by Multiple Segment Value, the user can type in multiple segment values for the search. In this case, a delimiter needs to be included to properly separate the segments.
- Search by consecutive characters in the combinations string: in this case, KFF LOV performs a 'constains' search, and as a result theuser can type in a sequence of characters that are contained within the combination string.

Note that depending on the search scenario, some searches perform better than the others. In general, better performing searches are:

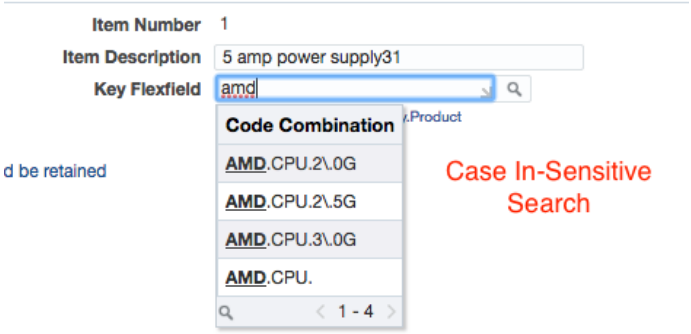
- Search With Correct Case
- Search With Single Complete Segment Value
- Search With Multiple Complete Segment Values

These search scenarios are therefore recommended for best performance. For the other types of search, although the best matches will still be returned to the user, it may require the server to send a much larger result set to the client for further filtering.

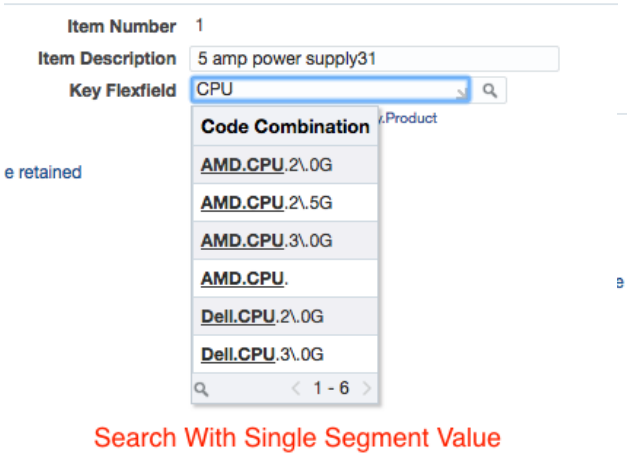
Examples of Look Ahead Search Scenarios

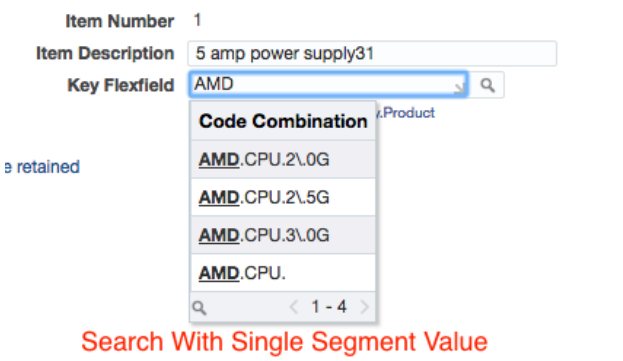
The following screenshots demonstrate various look ahead searches.

Example 1: Case insensitive search

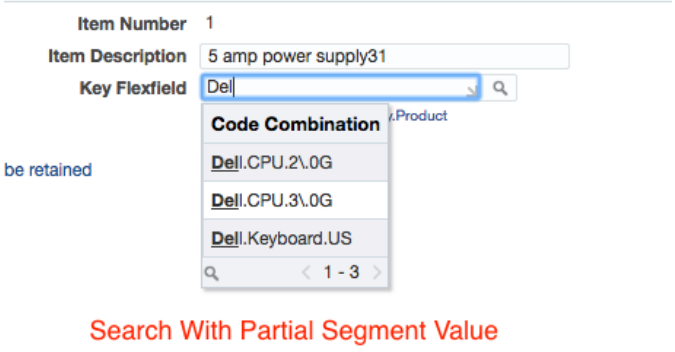


Example 2: Search by single complete segment value (two variants)

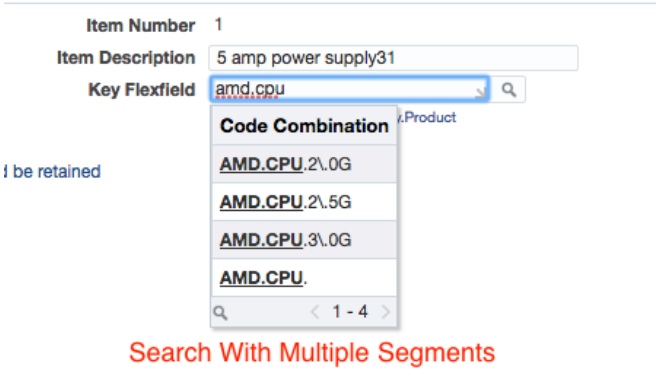




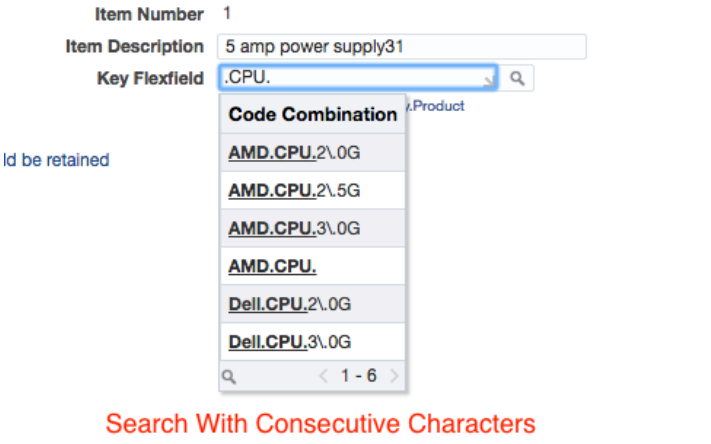
Example 3: Search by single partial segment value



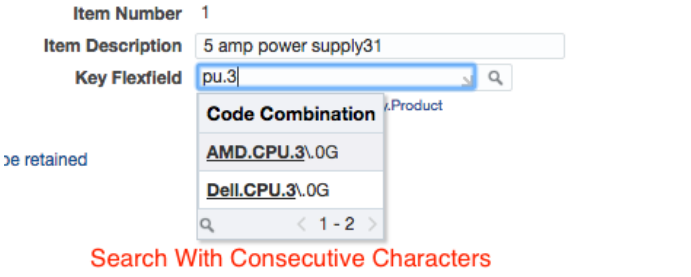
Example 4: Search by multiple segment values



Example 5: Search with segment delimiter in place



Example 7: Search with consecutive characters in the combination



Dynamic Insertion

When dynamic insertion is turned on for a key flexfield, a Create button is enabled in the key flexfield combination LOV Search page. If dynamic

insertion is allowed and you enter all the required segments for a key flexfield and find no matches on the combination LOV search page, you can select the Create button to create the new code combination.

Required Segments

If a key flexfield segment is defined as "required" in Oracle Applications, OA Framework displays an asterisk (*) before the segment. When you perform a search in the combination LOV Search page, you do not have to specify a value for a "required" segment. OA Framework only considers the "required" attribute of the segment when you attempt to create a new code combination with dynamic insertion.

PPR Support

If a PPR action occurs on a page and the view object attribute for the key flexfield is changed during the processFormRequest method, OA Framework automatically adds the flexfield web bean as a target for PPR and rerenders the key flexfield.

Setting Up a Key Flexfield in Oracle E-Business Suite

Before you can add a key flexfield to an OA Framework page, you must first set up the key flexfield in Oracle E-Business Suite. To start, review Chapter 2 ("Planning and Defining Key Flexfields") in the *Oracle E-Business Suite Flexfields Guide.*

When you have a clear plan for the key flexfield you wish to set up, refer to the section titled "Implementing Key Flexfields" in Chapter 14 ("Flexfields") of the *Oracle E-Business Suite Developer's Guide Release 12* for instructions to the following general steps:

Step 1: Define key flexfield columns in your database, including your combinations table.

Step 2: Register your key flexfield table with Oracle Application Object Library.

Step 3: Register your key flexfield using the **Key Flexfields Window**.

Next, refer to the *Oracle E-Business Suite Flexfields Guide* for instructions to these general steps:

Step 4: Define your value sets in the **Value Sets Window**, as described in Chapter 5 ("Values and Value Sets").

Step 5: Define your key flexfield structure and segments using the **Key Flexfield Segments Window**, as described in the "Defining Key Flexfield Structures" and "Defining Segments" sections of Chapter 3 ("Planning and Defining Key Flexfields").

Step 6: When you are ready to add the key flexfield to an OA Framework page, follow the steps outlined in the Declarative Implementation and Runtime Control sections below.

Declarative Implementation

The following steps describe how to add a key flexfield item to an OA Framework region:

Step 0: Ensure that the view object underlying the region includes the CCID column (the foreign key reference).

Step 1: In your region, choose **New > Item** from the context menu and set the `Item Style` of the new item to **flex**.

Note: You cannot create a flex item directly under a messageComponentLayout region, but you can create a messageLayout region under the messageComponentLayout region and add the flex item under the messageLayout region.

Step 2: Specify the `View Instance` for your flexfield. This view object instance should be the same view object instance that you specified for the region. Note that when you create the view object, you need only include the CCID column.

Note: OA Framework can support multiple key flexfields in a page, as well as in a table and on the same view object as long as each key flexfield is using a different CCIDAttributeName. See the Runtime Control section for details on how to specify the CCID column for your view object.

Note: If a flexfield's view object does not return a row, an OAException will not be thrown so that the controller's processRequest method can still execute and render the flexfield.

Note: OA Framework automatically handles dynamic insertion for you. See the Runtime Control section if you want to handle the dynamic insertion yourself.

Step 3: Set the key flexfield's `Appl Short Name` property to the short name of the application to which the key flexfield is registered.

Step 4: Set the key flexfield's `Name` property to the code of the key flexfield as it is registered.

Note: This differs from how a descriptive flexfield is defined by a Name.

Step 5: Set the key flexfield's `Type` property to **key**.

Step 6: Finally, set the `Segment List` property as appropriate (see Key Flexfield Segment List).

Key Flexfield Segment List

You may fill in the `Segment List` property if you want to show some, but not all of the segments in your flexfield. If you leave this property empty, all segments are rendered. The syntax is similar to that described for descriptive flexfields, only you should use structure codes to replace context values. The format is:

```
structure code1|segment1's name|segment2's name...||structure code2|segment4's name|segment5' name...
```

Segments within a certain structure code are separated by a single pipe, "|", while data from a different structure codes is separated by a double pipe, "||". The example below lists segments for just one structure code:

```
FWK Item Flexfield|Manufacturer|Product Family|Product
```

Read-Only Segments

You can add the read-only token (RO) after any of the segments in the list. For example, the (RO) designator below sets Segment1 in structure code 1 to be read-only:

```
Structure code1|Segment1($RO$)|Segment2...
```

Runtime Control

Warning: You should only create web beans programmatically if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines you should consider when writing web bean manipulation code.

If you need to programmatically create your key flexfield, it is essential to include the following calls in your code:

- Call the `setStructureCode` method on the `OAKeyFlexBean` to specify the structure for the key flexfield.
- Call the `setCCIDAttributeName` method on the `OAKeyFlexBean` to specify the CCID attribute name in your view object if it does not include all the individual flexfield segments in the view object. When you input a value in a key flexfield segment and submit the page, the key flexfield determines the code combination from the combinations table (maintained by OA Framework) and sets the `CCID` on your view object's `CCID` column. If dynamic insertion is enabled for a key flexfield and you input a combination that is not in the combinations table, OA Framework will create that combination as a record in the combinations table, and return a valid `CCID`. If dynamic insertion is not enabled, OA Framework returns `"-20"` as the `CCID`, indicating that the input combination is not found in the combinations table and dynamic validation is turned off.

The following code is an example of how to programmatically create a key flexfield:

```
public class RegionCO extends OAControllerImpl
{
  public void processRequest(OAPageContext pageContext, OAWebBean webBean)
  {
      super.processRequest(pageContext, webBean);
      OAKeyFlexBean kffBean = (OAKeyFlexBean)
        createWebBean(pageContext, KEY_FLEX_BEAN);
      kffBean.setAttributeValue(OAWebBeanConstants.VIEW_USAGE_NAME,
        "FNDFlexTestVO");
      kffBean.setAttributeValue(OAWebBeanConstants.FLEXFIELD_APPLICATION_SHORT_NAME,
        applicationShortName);
      kffBean.setAttributeValue(OAWebBeanConstants.REGION_APPLICATION_ID,
        new Integer(applicationId));
      kffBean.setAttributeValue(OAWebBeanConstants.FLEXFIELD_NAME,idFlexCode);
      kffBean.setCCIDAttributeName("UniqueIdColumn2");
      kffBean.setStructureCode(idFlexStructureCode);

      //optional:
      kffBean.setDynamicInsertion(isdynamicInsertion); //set dynamic insertion
      kffBean.setAttributeValue(OAWebBeanConstants.FLEXFIELD_SEGMENT_LIST,
        segmentList); //set segmentlist
      kffBean.useCodeCombinationLOV(false);
      //if you need old style key flexfield in 5.10 mode

  }

  public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
  {
      super.processFormRequest(pageContext, webBean);
   }
}
```

You should also consider these other key flexfield runtime control topics:

- Merging Key Flexfield Segments with the Parent Layout
- Handling Dynamic Insertion
- Read Only and Rendered Bound Values
- Adding a VRule to a Key Flexfield Web Bean
- Date Validation in a Key Flexfield
- Setting An Additional WHERE Clause
- Hiding the Description Text of a Key Flexfield Combination LOV

Merging Key Flexfield Segments with the Parent Layout

By default, flexfield segments are aligned within themselves but they are not aligned within the whole (parent) region. If you want to merge key flexfield segments to the parent layout, you must find the `OAKeyFlexBean` by attribute name and call the method `mergeSegmentsWithParent` on `OAKeyFlexBean`.

The following code example merges key flexfield segments to its parent layout. In this example, KeyFF is the item name (of the `item with the style` **flex**) in your region:

```
public class RegionCO extends OAControllerImpl
{
  public void processRequest(OAPageContext pageContext, OAWebBean webBean)
  {
      super.processRequest(pageContext, webBean);

      //find the flexfield that is defined in this region as item "KeyFF" and merge each
      //individual segment to the outside layout

      OAKeyFlexBean kffBean = (OAKeyFlexBean)webBean.findIndexedChildRecursive("KeyFF");
      flexBean.setStructureCode("FWK Item Flexfield");
      flexBean.setCCIDAttributeName("FwkitemId");
      flexBean.mergeSegmentsWithParent(pageContext);
  }

  public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
  {
      super.processFormRequest(pageContext, webBean);
   }
}
```

Handling Dynamic Insertion

If you wish to handle dynamic insertion yourself, you must first disable dynamic insertion by calling `setDynamicInsertion(false)` on the `OAKeyFlexBean.` Additionally, the submitButton for the page that contains the key flexfield must have its validation disabled.

Note: For additional information about dynamic insertion, refer to the "Dynamic Insertion" section of Chapter 2 ("Planning and Defining Key Flexfields") in the *Oracle E-Business Suite Flexfields Guide Release 12*.

Read Only and Rendered Bound Values

If you have to implement a flexfield in a table, where the flexfield is read-only or rendered in some rows and not others, you can do so programmatically using bound values. There is currently no declarative support for this.

First, create your bound value using `OADataBoundValueViewObject` or some other boundValue class. Then set the attribute in your controller's `processRequest` method as shown in the examples below:

```
kffBean.setAttributeValue(READ_ONLY_ATTR,boundValue);
```

or

```
kffBean.setAttributeValue(RENDERED_ATTR,boundValue);
```

Adding a VRule to a Key Flexfield Web Bean

A VRule is a validation rule that allows you to put extra restrictions on what values a user can enter in a key flexfield segment based on the values of segment qualifiers (which are attached to individual segment values).

Additional Information: Refer to the "Qualifiers" topic in Chapter 2 ("Planning and Defining Key Flexfields") of *Oracle E-Business Suite Flexfields Guide* for further discussion of flexfield and segment qualifiers.

You can add one or more VRules to a key flexfield web bean by using the `addVRule` method on `OAKeyFlexBean`. This method allows you to specify the flexfield qualifier and segment qualifier to validate, validation value(s) for the segment qualifier, whether to Include (true) or Exclude (false) the user entered value if its segment qualifier matches the specified validation value, and the Message Dictionary error message to display if the user enters an improper value. See the `addVRule` Javadoc for complete usage information about this method. This method should be called before the `processFlex` or `mergeSegmentsWithParent` methods.

As an example, suppose you implement a page where you want to prevent your users from entering segment values into segments of Oracle General Ledger's Accounting Flexfield, for which detail posting is not allowed. DETAIL_POSTING_ALLOWED is the segment qualifier, based on the global flexfield qualifier GL_GLOBAL, that you want to use in your rule. You want to exclude all values whose segment qualifier value of DETAIL_POSTING_ALLOWED is N (No). Your message name is ”GL Detail Posting Not Allowed”, and it corresponds to a message that says ”you cannot use values for which detail posting is not allowed.” You would add this VRule as follows:

```
kffbean.addVRule (GL_GLOBAL, DETAIL_POSTING_ALLOWED,
  "GL Detail Posting Not Allowed", false, N);
```

When a user enters an excluded value in one of the segments affected by this qualifier, the user gets the error message specified. In addition, the excluded values do not appear in the LOV of your segments. All other values, not specifically excluded are displayed.

Additional Information: For more information about VRules, refer to Chapter 9 ("Key Flexfield Routines for Special Validation") in the *Oracle E-Business Suite Flexfields Guide Release 12*.

Date Validation in a Key Flexfield

As of Release 12.2, you can control end date validation by specifying the date to be considered as the end date for a key flexfield's segment value or code combination.

For a declarative implementation:

1. Open the property inspector for the flex item.
2. Under 'Flexfields', add the Validation Date in the format:
   yyyy/mm/dd
   This will set the Validation Date for the kff bean.

To specify the validation date programmatically, call the `setValidationDate` method on `OAKeyFlexBean` with the desired value of `java.util.Date` object.

There are four possible options for the `DATE` input parameter required by the `setValidationDate` method:

1. `NULL` - Disables date validation.
2. `SYSDATE` - Checks date against `SYSDATE`.
3. A constant date - Checks date against a specified constant date.
4. Bound value (to a VO attribute) - Checks date against the date value set by the web bean code as resolved from the bound attribute value.

Note: Options 1 and 4 are currently not supported. For now, this method interprets a value of `NULL` as equivalent to a value of `SYSDATE`.

The example below illustrates setting the validation date to a constant date:

```
public class RegionCO extends OAControllerImpl
{
  public void processRequest(OAPageContext pageContext, OAWebBean webBean)
  {
      super.processRequest(pageContext, webBean);
      // find the flexfield that is defined in this region as item "KeyFF" and
      // set the validation date on the bean.
      try
      {
        SimpleDateFormat formatter = new SimpleDateFormat("yyyy/MM/dd");
        Date validationDate =  formatter.parse("2011/12/10");
        OAKeyFlexBean kffBean = (OAKeyFlexBean)webBean.findIndexedChildRecursive("KeyFF");
        kffBean.setValidationDate(validationDate );
       } catch (ParseException e)
       {
         System.out.println(e.toString());
         e.printStackTrace();
       }
  }
}
```

Setting An Additional WHERE Clause

Beginning in Release 12.2.5, you can use programmatic control to dynamically limit the set of key flexfield combination records displayed with either a static WHERE clause string or with a dynamic WHERE clause string that is dependent on the reference field. For example, if a key flexfield combinations table contains values and meanings for all 30 employees, but you only want to display entries for the 5 employees located in California, you can now include an additional WHERE clause (WHERE Location = 'CALIFORNIA') to the key flexfield that limits the valid values displayed.

Additionally, if the key flexfield web bean is inside a table or advanced table, you can also add a WHERE clause string that is dynamically bound on a per row basis.

As of Release 12.2.6, you may also overwrite the default error message when validation against the WHERE clause fails.

Note: To add a WHERE clause, the key flexfield must have combination LOVs enabled.

You may use the following new APIs with a plain string value as well as with references to `.$PROFILES$` syntax as shown in the more complex examples below.

1. APIs for setting a WHERE clause parameter on OAKeyFlexBean:

```
public void setWhereClause(String whereclause);
public String getWhereClause();
public void setWhereClauseforCreateCCID(String whereclause);
public String getWhereClauseforCreateCCID();
public void setWhereClause(String whereClause, Message errorMessage);
public void setWhereClauseforCreateCCID(String whereClause, Message errorMessage);
```

As an example, to set a string as the WHERE clause parameter:

```
String whereClause = "organization_id = :$PROFILES$.*ORGID
  AND SUBINVENTORY_CODE = :$PROFILES$.*Subinventory";
```

```
keyFlexBean.setWhereClause(whereClause);
```

Where `:$PROFILES$.*ORGID` and `:$PROFILES$.*Subinventory` are bound to reference field values. Create a flexfield map accordingly.

2. APIs for setting a key flexfield dynamic insertion reference field:

   The key flexfield LOV web bean is enhanced to fire an event `KFF_COMBINATION_LOV_UPDATE_EVENT` whenever the key flexfield combination LOV field value is updated. Call the following API to enable this new event:

```
public void setCodeCombinationLOVEventEnabled (boolean value);
public String isCodeCombinationLOVEventEnabled ();
```

   The following code example illustrates how to set up the key flexfield web bean and set the WHERE clause parameter:

```
if  (flexBean != null)
     {
       flexBean.setStructureCode("STOCK_LOCATORS");
       flexBean.setCCIDAttributeName("LocatorId"); //FwkitemId
       //Set DynamicInsertion bound value string
       flexBean.setDynamicInsertionReference(":$PROFILES$.*LocatorDynamic");
       String orgId = (String)pageContext.getTransactionValue(RcvUtils.SEARCH_SHIP_TO_ORG_ID);
       RcvUtils.log(MODULE, method, pageContext, "orgId is :" + orgId);
       if (RcvUtils.isNotEmpty(orgId))
       {
         flexBean.setDataSet(Integer.parseInt(orgId));
       }
       //Setting WHERE clause on the Locator KFF
       String whereClause = "SUBINVENTORY_CODE = :$PROFILES$.*Subinventory";
       flexBean.setWhereClause(whereClause);
       flexBean.setCodeCombinationLOVEventEnabled(true); //enable kff lov event
       flexBean.mergeSegmentsWithParent(pageContext);
     }
```

Hiding the Description Text of a Key Flexfield Combination LOV

To hide the description text that appears below a key flexfield combination LOV, call the following method on `OAKeyFlexBean`:

```
public void setSegmentDescRendered(boolean rendered)
```

The following code example does not render the description text below each key flexfield combination LOV:

```
OAKeyFlexBean flexBean = (OAKeyFlexBean)
webBean.findIndexedChildRecursive("RshLocatorKFF");
if (flexBean != null)
{
flexBean.setSegmentDescRendered(false);
flexBean.mergeSegmentsWithParent(pageContext);
}
```

Personalization Considerations

- See a summary of Flexfields personalization considerations in the Oracle Application Framework Personalization Guide.

Reference Fields

A reference field is a web bean that participates in evaluating one or more properties of a flexfield, such as default segment value, value set and context reference field for Descriptive flexfields. Such reference web beans are mapped to `:$PROFILES$.*<NAME>` references that are used in flexfields' properties. `<NAME>` can take any alphanumeric value thus providing a potentially infinite namespace of named parameters that can be passed from the OA Framework UI to the flexfield definition. Named parameters are mapped to reference web beans using flexfield maps. Whenever the value in the reference web bean changes, the page is refreshed with changes to the corresponding flexfield properties. The reference web bean display value is used for substituting the references in the properties.

Note: OA Framework does not have explicit support for `:<BLOCK>.<FIELD>` references that map fields in a form to a flexfield in Forms-based applications. If a flexfield used in a Forms-based application is to be re-used in an OA Framework based application, all the `:<BLOCK>.<FIELD>` references should be replaced with `:$PROFILES$.*<NAME>` references.

Declarative Implementation:

By default no flexfield maps metadata exists under a flex item. The following steps describe how to set up reference fields by adding flexfield maps to a flex item:

1. Right click on the flex item and select **New > flexMaps**
2. Specify a standards-compliant ID, an Attribute Set and other properties as usual for any item.
3. Specify the parameter name in the Name property of the flex maps. For example, if your Flexfield definition uses a named parameter `$PROFILES$.*paramName`, then set the Name property as `*paramName`.

4. Specify the value or value binding that needs to be set on the parameter.
   The value can be one of the following:
   - A Constant Value
   - A View attribute, represented by SPEL (**S**implest **P**ossible **E**xpression **L**anguage) as`${oa.<viewUsage>.<viewAttribute>}`
   - The name of the web bean on the current OA Framework page that will act as the reference field, represented by the SPEL - `${oa.page.<referenceId>}`
5. (Optional): (Only for Descriptive Flexfields) Set the Refer Context property to `true` if this flex map acts as a Reference Field.

   Repeat the above steps for creating as many flex maps as there are references.

Example

Suppose your descriptive flexfield has the context segment with Reference field set and you change the reference field value to use a dummy profile option such as $PROFILES$.*ASSET_CATEGORY. Perform the following steps to create a flex map in the OA Framework page that maps the value of this segment to a web bean, whose ID is ReferenceField, to drive the context field value:

1. Right click on the descriptive flexfield item and select **New > flexMaps**
2. Specify the **ID** as **sampleMap** and **Comments** as **A sample Flex Map**.
3. Specify the **Name** property of the flex maps as ***ASSET_CATEGORY**.
4. Specify the **Reference Value** as **${oa.page.ReferenceField}**.
5. Set the **Refer Context** property to **True** since this flex map acts as a Reference Field for the context of this descriptive flexfield.

Runtime Control:

You can also access or modify the flex map metadata to remove existing flex maps programmatically. Doing so, can override the behavior due to flex map metadata.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   super.processRequest(pageContext, webBean);
   ...
   OADescriptiveFlexBean flexBean =
 (OADescriptiveFlexBean)webBean.findIndexedChildRecursive("FlexItem");
    OAFlexMaps flexMaps = new OAFlexMaps();
    if (dffBean != null)
    {
      flexMaps = flexBean.processFlexMap(pageContext);
    }
    flexMaps.remove("flexMap");
    OAWebBean referenceField = webBean.findChildRecursive("referenceItem");
    OAFlexMap map1 = new OAFlexMap("flexMap1", new
OABoundValueFlexReference(referenceField), "*paramName", true);
       flexMaps.add(map1);
 flexBean.setFlexMaps(flexMaps);
...
}
```

If you need to programmatically create flexfield maps for your flexfield, you can include code similar to the example below in your controller's **processRequest** method. In the following example, `flexBean` is the flexfield web bean that contains the new flexfield maps and referenceItem is the Name of the reference web bean.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
super.processRequest(pageContext, webBean);
...
OAWebBean rootBean = pageContext.getRootWebBean();
OAWebBean referenceBean = rootBean.findIndexedChildRecursive("referenceItem");
OAFlexMaps flexMaps = flexBean.getFlexMaps();
if (flexMaps == null) flexMaps = new OAFlexMaps();
OAFlexMap map = new OAFlexMap("flexMap", new OABoundValueFlexReference(referenceBean), "*paramName");
flexMaps.add(map);
flexBean.setFlexMaps(flexMaps);
...
}
```

Note the following important restrictions:

- OABoundValueFlexReference is the equivalent of the SPEL binding `${oa.page.<id>}`. The OAFlexMap can take an optional fourth parameter of Boolean type. A `true` value indicates that value change in this field is going to trigger a change in the Context field. Please refer to the Javadoc of OAFlexMaps for more details.
- When a flex map is defined for a context reference field (of a descriptive flexfield) and the synchronize context flag is selected in the Flexfield definition, then the context field is disabled in the UI as the context reference web bean will act as the single source for the context.
- When a flex item is in update mode and the value derived from the view object row associated with this flexfield is different from the value supplied in the reference web bean, the following message displays as a Developer Mode exception:

  "The value in flexfield context reference web bean does not match with the value in the context of the Descriptive flexfield web bean PermitFlex. If this is not intended, please go back to correct the data or contact your Systems Administrator for assistance."

- OA Framework automatically associates a standard flex event named "FlexReferenceUpdate" with any reference web bean, but if the reference web bean already has an event associated with it, OA Framework will not associate the flex event "FlexReferenceUpdate" to the reference web bean.

Usage Restrictions

1. If a reference web bean is to have a default value, then this should be set either declaratively or in the processRequest of a controller of any of the container region of the flex web bean.
2. Reference web beans cannot be under a Table, an Advanced Table or an HGrid.
3. Modifying the definition of a flexfield setup as described above in the Reference Fields section may cause the flexfield to not work in the Oracle E-Business Suite Forms-based environment. For the same existing flexfield setup to work in both the Forms-based and OA Framework-based Oracle E-Business Suite environments, you must also personalize the flexfield in the Oracle E-Business Suite Forms interface. Use the Form Personalization feature to create one or more of the following trigger events to get the appropriate value populated into the profile option *<*Name*> when a user performs any of the relevant actions.
    - WHEN-NEW-FORMS-INSTANCE
    - WHEN-NEW-ITEM-INSTANCE
    - WHEN-NEW-RECORD-INSTANCE

    For example, to create Form Personalizations for the "Define Person Extra Info" function (Function Name: PERWSPEI), invoke that function from the Oracle E-Business Suite Navigation menu. While in the form, choose **Help >Diagnostics >Custom Code > Personalize** from the pulldown menu. The Personalization form opens and queries the existing Rules for that function.

    1. Enter values for **Seq** and **Description** to create a new Rule. Set **Level** to **Form** and check **Enabled**.
    2. In the **Condition** region, select **WHEN-NEW-FORMS-INSTANCE** for **Trigger Event** and set **Processing Mode** to **Not in Enter-Query Mode**.
    3. In the **Actions** region, set **Type** to **Builtin**, **Language** to **All**, **Builtin Type** to **Set Profile Value in Cache**, and **Profile Name** to ***X**. Select **Apply Now**.
    4. Repeat steps 1 to 3 to create a rule for the WHEN-NEW-RECORD-INSTANCE trigger event.
    5. Repeat step 1 to create a rule for the WHEN-NEW-ITEM-INSTANCE trigger event.
    6. In the **Condition** region, select **WHEN-NEW-ITEM-INSTANCE** for **Trigger Event**, enter **PEI.DF** for **Trigger Object** and set **Processing Mode** to **Not in Enter-Query Mode**.
    7. In the **Actions** region, set **Type** to **Builtin**, **Language** to **All**, **Builtin Type** to **Set Profile Value in Cache**, and **Profile Name** to ***X**. Enter **:PEI_INFORMATION1** for the **Profile Value**. Select **Apply Now**.

    For more details on Form personalizations, refer to the "Form Personalization" topic in the "Developer Tools" chapter of *Oracle E-Business Suite Setup Guide, Release 12.2* and My Oracle Support Knowledge Document 395117.1, *Form Personalizations in Oracle E-Business Suite (Release 12).*

4. Refer to known flexfield issues for other restrictions.

Personalization Considerations

- See a summary of Flexfields personalization considerations in the Oracle Application Framework Personalization Guide.
- System Administrator can create flexMaps using personalization UI.

Values and Value Sets

Generally, the value entered in a key or descriptive flexfield segment is validated against a predefined set of valid values (a "value set"). When you define your value set, you can specify the types of values that can fit into the value set by setting up formatting rules. The validation type that you specify for your value set will determine how a user can enter or use the segment in the flexfield. The validation types are:

- **None** - no validation, so a value can be entered as long as it meets the formatting rules.
- **Independent** - provides a predefined list of values for a segment that are stored in an Oracle Application Object Library table.
- **Table** - provides a predefined list of values like an independent value set, but its values are stored in an application table. The values are defined by the table you specify and the WHERE clause that limits the values to use in the set. You can use bind variables such as $FLEX$ and $PROFILES$ in your WHERE clause to base your list of possible values on other values. For example, $PROFILES$ lets you bind the value to a specific profile option and $FLEX$ lets you bind to the value of a prior segment.

  Additional Information: Refer to the topic "Bind Variables" in Chapter 4 ("Values and Value Sets") of the Oracle E-Business Suite Flexfields Guide Release 12 for details on how to use these bind variables.

- **Dependent** - provides a predefined list of values, but the values depend on the independent value specified in a prior segment of the flexfield structure.
- **Translatable Independent** and **Translatable Dependent** - similar to Independent and Dependent value sets, respectively, except that a translated value can be used.
- **Special** and **Pair Value Sets** - these value sets provides a mechanism to pass key flexfield segment values or combinations as report criteria for Standard Request Submission (also referred to as a "flexfield-within-flexfield"). The "Pair" validation type is not yet supported in OA Framework and will result in a developer mode error if specified.

  Note: Beginning in Release 12.2.6, the "Special" validation type is supported in the context of concurrent program pages. Concurrent program parameters are defined as a descriptive flexfield and you can define a descriptive flexfield segment (a parameter in the context of a concurrent program) using a value set with validation type "Special". Configuring the descriptive flexfield to use the "Special" value set to validate the segment allows you to display the descriptive flexfield segment as a key flexfield. As a result, a user can enter an entire key flexfield combination within a single segment of that descriptive flexfield. The concurrent program parameter with the "Special" value set is also supported in both read-only and non-read-only modes. This feature, however, is meant only to be used in

the context of concurrent program pages. Should you attempt to directly embed such a descriptive flexfield in a product page, an error will result.

Additional Information: To learn more about defining a report parameters window (descriptive flexfield) for Standard Request Submission or about "Special" or "Pair" value sets, refer to Chapter 7 ("Standard Request Submission") in the Oracle E-Business Suite Flexfields Guide Release 12.

When a value set's list type is 'Poplist' or 'List of Values', OA Framework automatically renders the segment as a poplist or a LOV, respectively.

Additional Information: To learn more about value sets, refer to Chapter 4 - "Values and Value Sets", in the Oracle Applications Flexfields Guide Release 12.

Flexfield Validation

When you implement a page with an Action (Submit) Button, you have the option of setting the page submit, when the button is selected, to be 'validated' or 'unvalidated'. The behavior of a flexfield in such a page will vary depending on whether you set the **submitButton** properties `Disable Server Side Validation` and `Disable Client Side Validation` to `true` or `false`.

Note: For Flexfields, to address backwards-compatibility, setting either the `Disable Server Side Validation` or `Disable Client Side Validation` property to `true` disables ALL validation, that is, both server and client validation.)

The following table describes a flexfield's behavior when both the client and server validation is disabled and enabled.

Disable Validation (Client and/or Server)	Effect on Flexfield Behavior
False	1. On the client side, client Javascript checks for required field(s) and date format(s) and displays a Javascript error if error is found. 2. On the server side, all data entered in the UI is posted to the `FlexJ object`, and if any segment input data is invalid (such as a wrong format), an error is thrown. 3. The `FlexJ object.validate` method is called to validate the whole flexfield. If there is any error, an exception is thrown on the UI. 4. If all validation passes, the data is populated to the view object.
True	1. Any client Javascript is not triggered. 2. On the server side, when all data is posted to the `FlexJ object`, if any user input for any segment has an invalid format, the value is discarded, but no error will be thrown. 3. The `FlexJ object.validate` method is not called. 4. All data is posted to the view object as is.

An example of when you may want an unvalidated submit is when you have a descriptive flexfield in a Search region.

Flexfields and Standard Request Submission

The Report Parameters window for Standard Request Submission (SRS), that allows users to enter criteria for their reports, is a special descriptive flexfield. While many of the setup steps for Report Parameters are similar to that of a descriptive flexfield, the main differences are that you use the Concurrent Programs window to define your segments instead of using the Descriptive Flexfield Segments window and the way you define and use value sets for SRS report parameters is more complex.

Additional Information: Refer to Chapters 7 ("Standard Request Submission") and 8 ("Reporting on Flexfields Data") in the Oracle E-Business Suite Flexfields Guide Release 12.

If you want to provide a flexfield as a report parameter (also referred to as a flexfield-within-a-flexfield), you can use a flexfield routine to specify the type of flexfield you want as a value set for the report parameter.

Additional Information: Chapter 9 ("Key Flexfield Routines for Special Validation") in the Oracle E-Business Suite Flexfields Guide Release 12 lists the syntax for the key flexfield routines.

Developer Mode Errors

Oracle E-Business Suite throws a developer mode error if any of the following flexfield implementations are detected:

- A reference field is used in flexfield.
- A pair value set is used in flexfield.
- A special value set is used in flexfield.
- A flexfield is in a Table-in-Table layout.
- A flexfield is directly put in a Message layout.
- A flexfield is in a Query bean layout

Troubleshooting Flexfield Problems

If your flexfield is not working as expected or there is an error on the flexfield, you can follow these troubleshooting steps to help identify the source of the problem:

Step 1: Check your flexfield setup

Login to your Oracle E-Business Suite Forms-based environment as an applications developer, and navigate to the **Segments** window (**Flexfield** > **Descriptive/Key** > **Segments**). Identify the flexfield you are using, then make sure the **Freeze Flexfield Definition** checkbox is checked for this flexfield. Compile the flexfield again. If your flexfield is using some feature that is not supported by OA Framework page, a warning message will appear.

Step 2: Run the Flexfield Test tool under the guidance of your support representative

Login to your Oracle E-Business Suite Forms-based environment as an applications developer, and navigate to the **Flexfield Text** window (**Flexfield** > **Flexfield Test**). Identify the flexfield you are using and follow the instructions provided by your support representative to use the Flexfield Test window.

Step 3: Check if there are any developer mode errors for your flexfield on the OA Framework page

At run time, OA Framework checks if the flexfield is properly configured and used. If it finds anything wrong with the flexfield, OA Framework renders a developer mode error on the page as well as lists these errors, if any, in the "About" Page. To enable Developer Test Mode diagnostics, you must set the profile option FND: Developer Mode / FND_DEVELOPER_MODE to **Yes**. To render the "About this page" link at the bottom of every OA Framework page, you must set the profile option FND: Diagnostics / FND_DIAGNOSTICS to **Yes**.

Step 4: Use the "About" page to get more information about the flexfield

On the bottom of every OA Framework page, there is an "About this page" link (if the profile option FND: Diagnostics / FND_DIAGNOSTICS is set to **Yes**). When you open the "About" page, you can see the page structure as well as the flexfield detail information in the Page tab. All flexfield information is listed in the **Flexfield References** section, including:

- The flexfield name and Applications ID.
- Detail information about each segment, such as segment name and segment value set.
- Whether there is any developer mode error for the flexfield.

Known Issues

See a summary of known flexfield issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
  - Flexfields
- Javadoc File(s)
  - `oracle.apps.fnd.framework.webui.beans.OADescriptiveFlexBean`
  - `oracle.apps.fnd.framework.webui.beans.OAKeyFlexBean`
  - `oracle.apps.fnd.framework.webui.beans.OAWebBean`
  - `oracle.apps.fnd.framework.OAFlexUtils`
  - `oracle.apps.fnd.framework.webui.OAFlexMap`
  - `oracle.apps.fnd.framework.webui.OAFlexMaps`
  - `oracle.apps.fnd.framework.webui.OABoundValueFlexReference`
- Lesson(s)
  - Framework Toolbox Tutorial Supplemental Lesson: Creating Descriptive and Key Flexfields
- ToolBox Tutorial / Sample Library
- Search
- FAQs
  - Flexfields

Gesture Support

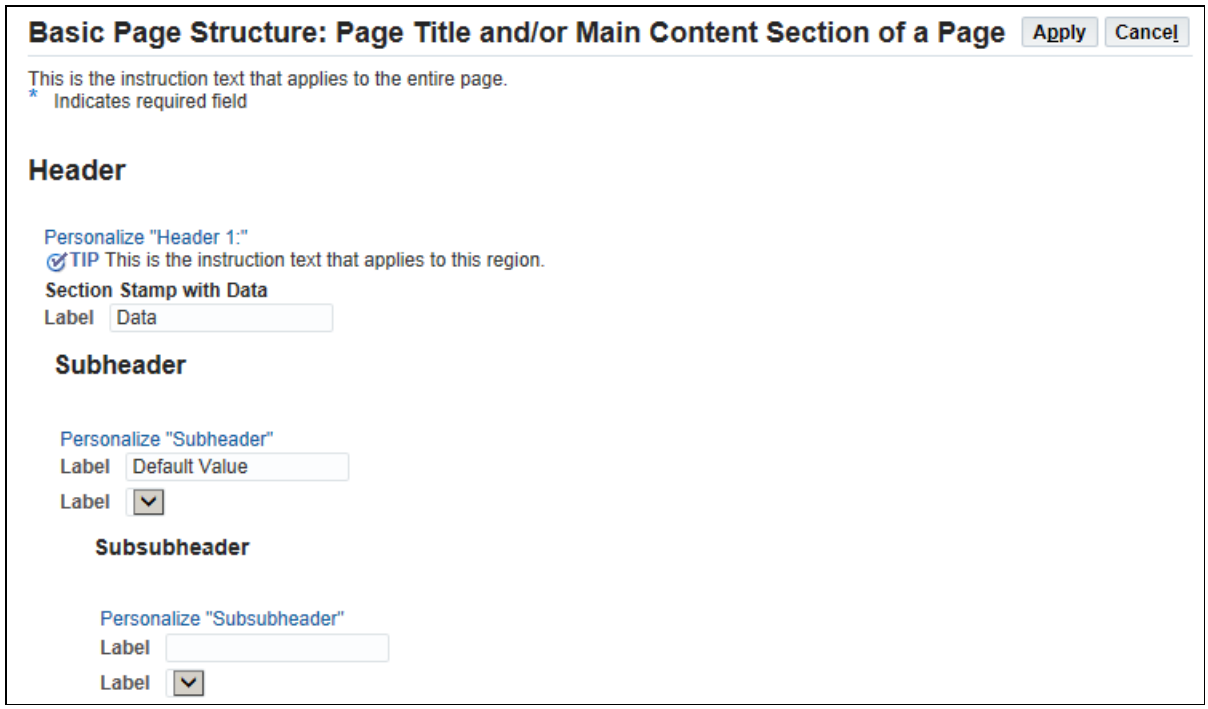
Refer to the Mobile Applications topic for information on gesture support.

Headers and Subheaders

Overview

Per the BLAF UI Guide: Headers and Subheaders specification, the header component is used to title and separate page contents as illustrated below.

Figure 1: Example of headers, subheaders and subsubheaders in a page.



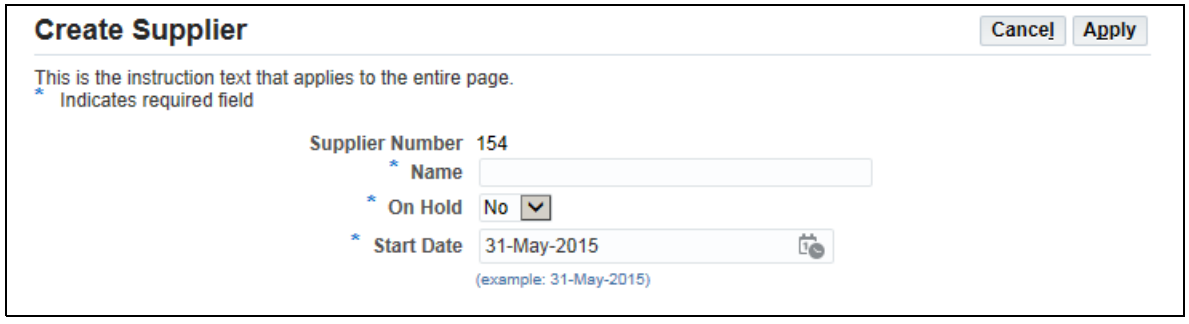
Primary Header (Page Title)

All pages, except for the "Home" page, should have a page title that describes the page's contents. Not only does this text provide useful information to the user, but the OA Framework also uses this value to:

- Set breadcrumb text when you drill down from the page (if the page title property isn't set, the framework will use the browser window title instead -- and if you've followed the UI guidelines in setting the window title, your breadcrumbs will be incorrect)

Note: As of Release 12.2.5, the page-level action/navigation buttons render on the same line as the page title, as a header-level button bar.

Figure 2: Example of page-level action/navigation buttons rendering on the same line as the page title.



Declarative Implementation

To add a page title, simply specify the `Title` property for your **pageLayout** region.

Note: The **pageLayout** region also has a `Window Title` property which is used to specify the browser's window title. This has nothing to do with specifying the page title.

Warning: Although your page may *appear* to have a page title if you add a header or one of the "default" regions to your pageLayout, the OA Framework does not interpret this as a page title. You must explicitly specify the region property as described.

Runtime Control

Warning: See the OA Framework Controller Coding Standards for guidelines that you should consider when writing web bean manipulation code.

Instantiate

Since the page title is a property of the page layout itself, you can't instantiate a page title directly. Instead, you set it on the `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean` as shown below.

Control Visual Properties

To set the page title programmatically (which is a common practice if you need to specify context for the header like you would with "Update Employee: Fred Flintstone"), do the following:

```
import oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean;
...
processRequest(OAPageContext pageContext, OAWebBean webBean)
{
...
    // Assuming the controller is associated with the pageLayout region
    OAPageLayoutBean page = (OAPageLayoutBean)webBean;

   // Assuming the controller is associated with a child/grandchild region of the
   // pagelayout region
   OAPageLayoutBean page = (OAPageLayoutBean)pageContext.getPageLayoutBean();

   // Always set translated text Strings obtained from Message Dictionary
   page.setTitle(<title text>);
...
```

See Example: Implementing Search & Drilldown to Details for a more detailed example of setting a contextual page title.

Subheaders

The UI guidelines allow for two levels of subheaders below the page title: a "subheader" and a "subsubheader" as shown in Figure 1 above.

Declarative Implementation

- To add a **subheader** to your page add a region with one of the styles listed in Figure 3 to a **pageLayout**.
- To add a **subsubheader**, add a region with one of the styles listed in Figure 3 to any subheader. Remember to specify its `ID` property in accordance the OA Framework Package / File / Directory naming standards.

In both cases, set the `Text` property to specify the header title. OA Framework automatically indents the header in relation to its parent region, and sizes the header text in accordance with the UI guidelines.

Tip: The classes corresponding to each of the "default" region styles subclass `oracle.apps.fnd.framework.webui.beans.layout.OAHeaderBean`, so they all behave as headers in your page. If you want to use these regions as layout templates, and you don't want the header line to show, set the `Hide Header` property to **True**.

Figure 3: Relationship between header region styles and OA Framework classes

Region Style	OA Framework Class
header	`oracle.apps.fnd.framework.webui.beans.layout.OAHeaderBean`
hideShowHeader	`oracle.apps.fnd.framework.webui.beans.layout.OAHideShowHeaderBean`

The `OADefaultSingleColumnBean` and `OADefaultDoubleColumnBean` classes have been deprecated. In their place, you should use a combination of an `OAHeaderBean` followed by an `oracle.apps.fnd.framework.webui.beans.layout.OAMessageComponentLayoutBean`. See Page Layout (How to Place Content) for additional information.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created

web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

Instantiate

You can instantiate any of the classes described above by calling the appropriate `createWebBean()` method in the `oracle.apps.fnd.framework.webui.OAControllerImpl` class. If you select a signature that requires a constant to determine what kind of bean to create, use the following for each class:

Figure 4: OAWebBeanConstants for creating corresponding OA Framework classes

Constant	OA Framework Class
OAWebBeanConstants.HEADER_BEAN	`OAHeaderBean`
OAWebBeanConstants.HIDE_SHOW_HEADER_BEAN	`OAHideShowHeaderBean`

Note: You should not instantiate and programmatically add contents to any of the "default" regions. You may, however, instantiate regions that you define declaratively in JDeveloper.

Control Visual Properties

To change the header's text value, get a handle to the right class (based on what you instantiated, or specified declaratively) and call `setText(OAPageContext pageContext, String text)`.

To achieve the correct text size as specified in the UI Guidelines when headers are used in side navigation components, or displayed in the "Home" page main content area (in an "At a Glance" region, for example), call `setSize(int size)` on the header bean with one of the following values.

- 0 - the "large" size
- 1 - the "medium" size (used for headers displayed in the "Home" content page area).
- 2 - the "small" size (used for headers added to side navigation components)

See the ToolBox Sample Library for an example of a "Home" page including headers in the main content area and in a side navigation component.

To set the associated icon in your processRequest method, call `setIcon(String icon)` as shown:

```
header.setIcon(OAWebBeanConstants.APPS_MEDIA_DIRECTORY + "<icon_name>.gif");
```

Note: In R12, the header region without a title appears as a long blue line.

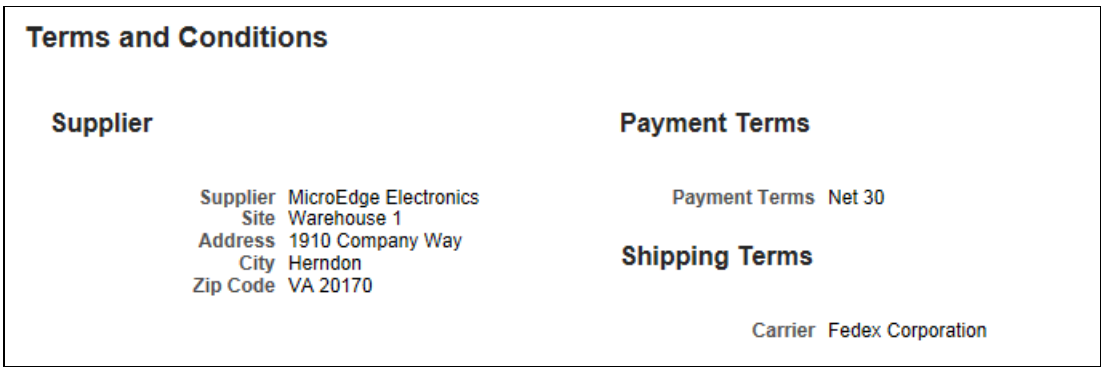
As of Release 12.2.5, you may also use the new `OAHeaderBean` API to add a header-level button bar:

```
OAHeaderBean oa = <>;
oa.setHeaderButtons(buttonRegion);
```

Adjacent Subheaders

The UI Guidelines allow multiple subheaders to be used side-by-side in a page a shown in Figure 4.

Figure 4: Example of adjacent subheaders.



Declarative Implementation

You create the headers themselves as described in the Subheaders section above. Creating the layout to hold you adjacent subheaders is a different matter. For help with creating complex layouts declaratively, see Page Layout (How to Place Content).

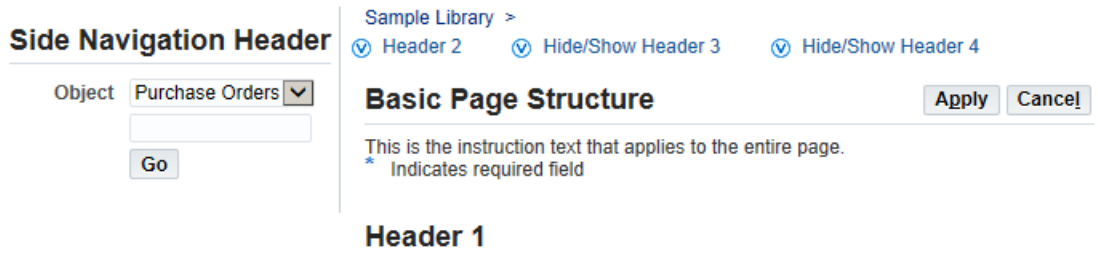
Runtime Control

For any programmatic changes to the headers, also see the Subheaders section.

Headers in Side Navigation

You can also add headers to side navigation controls as shown in Figure 5.

Figure 5: Example of headers in side navigation



Declarative Implementation

Currently, you can't add a Side Navigation (including a header) to your page declaratively. See the Tabs / Navigation document for instructions on creating a Side Navigation component. Once you've created your Side Navigation, you simply add your header to it as you would to any other component.

Runtime Control

Control Visual Properties

When you add a header to a container, the OA Framework automatically sets the text and line colors based on the corresponding background color. You do not need to set any color properties.

The only change that you're likely to make to a header when you add it to a side navigation is to change the size of the header text since this is *not* configured automatically.

Note: You cannot configure header text size by setting a CSS style; this is not supported. See the instructions for changing the header size in the Subheaders section above.

Hide/Show Subheaders

- See the Hide/Show documentation for a description of this feature and implementation instructions

Known Issues

- None

Related Information

- BLAF UI Guidelines:
  - Headers and Subheaders
  - Locator Element (Breadcrumbs)
- Developer's Guide:
  - Hide/Show
  - Content Containers
  - Page Layout (How to Place Content)
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAHeaderBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAHideShowHeaderBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAMessageComponentLayoutBean`
- ToolBox Tutorial / Sample Library

HGrid

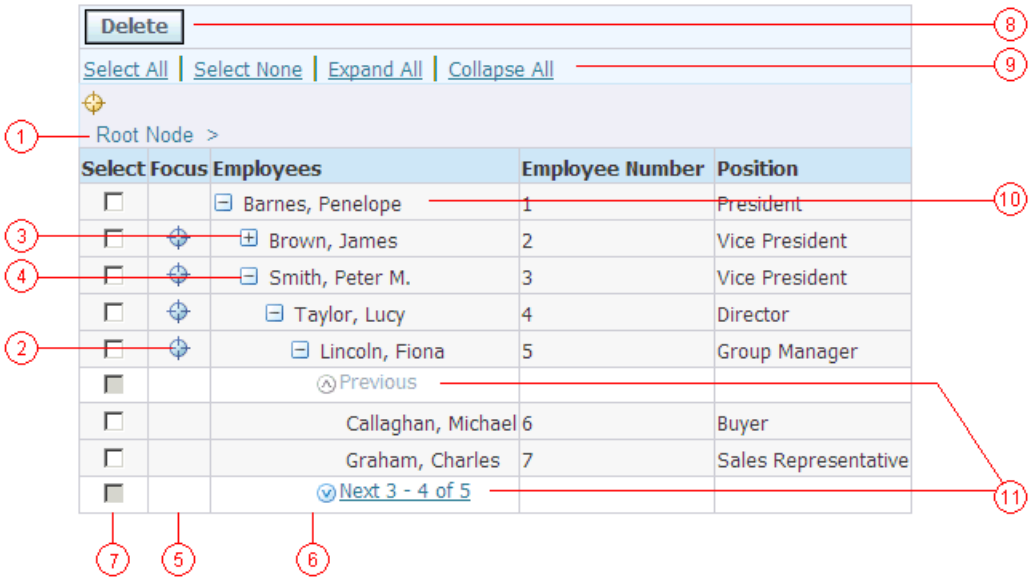
Overview

A HGrid, otherwise known as a hierarchy grid, allows users to browse through complex sets of hierarchical data. Certain types of data, such as the organizational structure of employees within a company, are naturally hierarchical and best displayed as a tree structure. There are two UI components that display hierarchical information on a page: the HGrid web bean and the Tree web bean. A Tree is generally used when you want to put emphasis on the hierarchy and the relationship between different sets of objects in a hierarchy. A HGrid is more appropriate when you want to display the hierarchy, but also give more detailed information at each node of the hierarchy.

Consider using a HGrid instead of a Tree when you want your users to either:

- Manipulate the objects in the hierarchy (add, delete, move, reorder, etc.). Note that in certain cases, you may want your users to navigate from a Tree to a HGrid to perform these actions. This is done by providing a button that switches to an "update" or "edit" mode, which displays a HGrid instead of a Tree.
- See more object details than can be displayed in a Tree, plus the amount of required detail per object does *not* warrant use of a Tree with a Master/Detail view.

Following is an example of a HGrid.



A HGrid consists of the following components/features, as pointed out in the figure above:

1. Breadcrumbs
2. Focus icon
3. Expand node icon
4. Collapse node icon
5. Focus column
6. Object hierarchy column
7. Multiple selection
8. Control bar
9. Selection/Expansion control
10. Root node
11. Record navigator

Each row in a HGrid corresponds to a tree node. A HGrid also has two special columns: a focus column and an object hierarchy column. The object hierarchy column identifies the current tree node and allows you to expand or collapse this node. When you expand on a node that contains more records than the defined record set size, "Next" and "Previous" record navigation links appear. If the total number of records is known, the record navigation links also display the record set range and the total number of records.

The focus column allows you to select a new root for the tree. You can zoom into a sub-tree of a massive tree by selecting the focus icon for that sub-tree row. The HGrid also renders bread crumbs, allowing you to focus out (or zoom out) of the current sub-tree, and renders links to allow you to quickly expand or collapse all the nodes under the current focus root.

This document describes the declarative definition and APIs provided by the HGrid

(`oracle.apps.fnd.framework.webui.beans.table.OAHGridBean`) component within OA Framework. As described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: HGrid specification, the HGrid feature shares many properties with tables, most notably that it is a display of information in tabular format. The main difference between the two is that a table displays a flat list of objects, whereas a HGrid displays objects in a hierarchy. You should be familiar with the construction of a table (setting up the metadata as well as the business components). If you are not familiar with tables, please take a quick look at the Tables documentation before proceeding.

Contents

- Defining Business Components
- Declarative Implementation
  - Defining a HGrid
  - Enabling Search on a HGrid
- Runtime Control
  - processRequest method
  - processFormRequest method
  - Getting and Setting the Initial Focus Path
  - Deleting Rows in a HGrid
  - Using FireAction on the Hierarchy Column
  - Using Save Model on the Hierarchy Column
  - Per Row Dynamic Poplist
  - Getting a Hold of the viewInHierarchy Named Child's Link Web Bean
  - Preserving HGrid State
    - HGrid Data Proxy in UIX
    - Proxy Use in the OA Framework
    - Support for Dynamic HGrids
  - Optimizing Child Row Retrieval
- Rich HGrid Interactions
- Support for Touch Devices
- Personalization Considerations
- Known Issues
- Related Information

Defining Business Components

As with all other beans supported by OA Framework, the data for the HGrid component is derived from BC4J objects. Just as the HGrid displays data in hierarchical format, the structure of the source data is also hierarchical, based on multiple view objects connected via view links. Each instance of a view object is connected to the HGrid at a particular level, allowing the HGrid to display all the rows in range at that level. The view links that define the parent-child relationship between a row in the master view object and the detail view object allow you to drill down in the HGrid and show details at the next level. When a user selects the hide/show (expand/collapse node icon) to show more details, OA Framework traverses the view link and fetches/displays all the detail records for the master row that is selected.

Note: It is possible to use a different view object for the children rows as long as the parent view object and the child view object share the same attribute names. The reason for this becomes clear when setting up OA Extension metadata. An unlimited number of view links can be used to define the necessary parent-child relationships at each level of the hierarchy.

When defining a HGrid in OA Extension, specify the name of the view link instance for each node that is used to drill down to the correct detail view object instance. To ensure that the HGrid performs well, the view objects and view links defined *must* be efficient. In general, the view links get fired at each level to determine whether or not there are children. As a result, multiple queries are performed if there are multiple nodes displayed at a level, therefore, record navigation is recommended for better performance.

Note: If your data model supports it, designate an alternate master view object attribute as the one to use to determine whether there are children. This avoids the need to fire multiple view link queries.

Each row in a view object provides data for a corresponding row in the HGrid at each level. An initial instance (top most) of the view object should return the root node of the HGrid. This top level view object should return *exactly* one row. The HGrid component relies heavily on the notion of hierarchical data with a unique root. If the top level view object returns multiple rows of data, those rows are automatically made children of a dummy root node. The automatically generated parent dummy node renders as non-selectable and expanded.

Note: You cannot get a handle to this dummy node as it is generated internally and does not map to any row or view object attached to the HGrid.

The first step in defining a HGrid is to define the business object hierarchy that maps to your business requirements.

To illustrate the above, build a simple HGrid example to display supervisor-employee hierarchy information.

Note: Some of the data model is greatly simplified for this example.) The data for each employee comes from the `PER_ALL_PEOPLE_F`

view. Each employee is uniquely identified by the `PERSON_ID` column in this view. The `PER_ALL_ASSIGNMENTS_F` view describes the supervisor-employee relationship through the `SUPERVISOR_ID` and `PERSON_ID` columns in this view.

Step 1: Set up a view object definition for the `PER_ALL_PEOPLE_F` view, selecting the data that you want to display in the HGrid. Download `oracle.apps.fnd.framework.persontree.server.PerAllPeopleFVO` as an example. You can also download the corresponding VOImpl class.

Note: The `initQuery` method in the VOImpl adds an additional where clause to the view object to fetch the root node.

Step 2: Define the view link used to retrieve subsequent levels of the HGrid. In this example, define a view link that links the `PerAllPeopleFVO` to itself.

a. In JDeveloper, select the package where you want to add the view link. From the context menu, choose **Create View Link ...** to display the "View Link Wizard".

b. In the View Link Wizard, **Step 1 of 6: Name**, enter a name for the view link. (In this example use `PerAllPeopleFVL`).

c. In **Step 2 of 6: View Objects**, choose the source and destination view objects. (In this example use `PerAllPeopleFVO` as both the source and destination).

d. In **Step 3 of 6: Source Attributes**, select the source attributes. These are typically the primary key attributes (or a subset thereof) of the source view object. Select any other columns that are needed to build the where clause used to fetch the detail row set. The values of these attributes from the master view object are used to determine the detail row set. (In this example, use the `PERSON_ID` column as discussed earlier).

e. In **Step 4 of 6: Destination Attributes**, select the destination attributes (as in the previous step).

f. In **Step 5 of 6: View Link SQL**, build the where condition used to get the detail row set. The default where clause simply contains a one-to-one mapping of the source and destination attributes. The bind variables are bound with values of the source attributes in the master row. (In this example, use the `PER_ALL_ASSIGNMENTS_F` view to determine all the persons supervised by the person in the master row). In other words, construct a where clause as follows:

```
person_id in (select person_id from per_all_assignments_f where supervisor_id = :1)
```

g. In **Step 6 of 6: View Link Properties**, ensure that the **Generate Accessor in View Object** checkbox is checked for both the Source and Destination view objects. The accessor name is generated automatically but you can change it if desired.

h. Download the complete definition for PerAllPeopleFVL. Setup additional view links if the master-detail relationships at each level of the HGrid are different.

Step 3: Add the view objects and view links that you created to the application module used for the page.

Warning: Using the Application Module Wizard to add the view link to the application module can be difficult. First add the view objects to the application module. Then to add a view link, select the view link in the left column and select the source view object in the right column. This enables the ">" shuttle control, which you can use to move the view link over to the right.

HGrid Navigation

To implement a HGrid for the purpose of navigating to additional detail information about the selected node in subtabs below the HGrid, consider the following when creating your BC4J components:

- If the detail information consists of other attributes of a UI row selected in the HGrid, then the BC4J row used by the HGrid should ideally be used by the detail section. However, this row exists in a non-default RowSet of an internal view object. Currently our UI components (outside of HGrid and Table-in-Table) can only be bound to the default RowSet of a view object. Therefore, in this situation, you *must* use a separate view object for the detail section to query for the detail information. While this extra query is not ideal, on the positive side:
  - The rows used by the HGrid UI will have fewer attributes to query .
  - The detail section may require joins to other tables to get information and that complexity can be kept out of the view objects used for the HGrid itself.
- As long as the application module is kept around, all the data for the HGrid will be kept as well. However, if a separate view object is used for the detail section as described above, the detail view object will get refreshed constantly. (This would preclude the possibility of updates in the detail section.) If updates are required, you *must* create distinct view objects for the detail section corresponding to each row in the HGrid, resulting in more data being held around. This also means you will have to manage view instances for the detail and change the view usage on the detail UI components each time.

Declarative Implementation

Defining a HGrid

Having defined the data sources for the HGrid bean, the next step is to define the HGrid component in OA Extension. Refer to the OA

Component Reference for additional information about the properties you can set on the hGrid region.

In OA Framework, in addition to containing all the region items that a Table region can contain, a HGrid style region also contains a nested region item of style HGrid Hierarchy. The HGrid Hierarchy region item simply points to a nested region of style Tree Level. A Tree Level region corresponds to a node in a HGridBean. A Tree Level region contains two region items: a Tree Definition and a Tree Child. The Tree Definition region item describes the node and the Tree Child region item points to a nested region of style Tree Level. This allows OA Framework to build complex hierarchies in a declarative way.

The definition of a HGrid is similar to that of a table. Specify the following metadata in OA Extension:

Step 1: Define a top level region and set the `Region Style` property to **hGrid**. A HGrid region may also have an optional controller and application module. You can nest this region within a container region such as PageLayout, Header, Stack Layout, or messageComponentLayout.

Note: OA Extension assumes that for all regions, the `Add Indexed Children` property is set to True. As a result, the `Add Indexed Children` property generally does not appear in the Property Inspector. However, for backwards compatibility, the `Add Indexed Children` property will appear for a region if you previously set this property to False using an earlier version of OA Extension.

Step 2: Select the hGrid region in the Structure pane. In the Property Inspector, set the `Record Set Size` property to control the maximum number of records that can be displayed at a time under each tree node. The syntax used to display the record navigation links is:

```
{ParentName[/ObjectType];} {Next | Previous} [SetRange] [of TotalRecords]
```

For example:

```
Car Division/Brands: Next 11-20 of 26
```

Note When you set the `Record Set Size` property on the hGrid region, the value applies to all tree nodes in the HGrid. To achieve finer control, where each tree node has a different maximum record set size, you can set the `Record Set Size` property on the specific nodeDefinition.

Step 3: `OAHGridBean` supports both single and multiple selection of rows. Refer to the instructions for rendering table selection and selection buttons on the control bar for additional information.

Note HGrid currently does not yet support the Advanced Table selection and control bar implementation.

Step 4: In the OA Extension Structure pane, select the hGrid region and display the context menu. In the context menu, under **New**, select **tree**, to create a HGrid hierarchy column (which distinguishes a HGrid from a table). In the figure below of the HGrid structure, the tree region is labeled as HGridHierarchyRN.

This nested tree region defines a particular level in the hierarchy of the HGrid. The tree region can have two types of named children (members):

- **nodeDefinition** - The nodeDefinition item automatically appears when you create the tree region. It defines the appearance of the node in the hierarchy. For the nodeDefinition item, specify:
  - a value for the `View Instance` property to associate the node with a view instance.
  - a value for the `View Attribute` property, to render the view attribute name as the text of the node in the object hierarchy column.
  - a value for the `Icon URI` property to render an image next to the text in the node in the object hierarchy column.
  - a value for the `Destination URI` property to render the node as a hyperlink in the object hierarchy column.
  - a value for the `Record Set Size` property, if you wish to display a maximum record set size for this node that is different from the value of the `Record Set Size` property set on the hGrid region.
- **childNode** - In the Structure pane, select **members** under the Tree region and display the context menu. Under **New**, select **childNode**.The childNode item defines the parent-child relationship between tree levels.
  - Set the `Ancestor Node` property on this item to indicate the region where you are looping back. The `Ancestor Node` property can be set to another tree region, to the same tree region (for a recursive relationship), or to no tree region (null, indicating that the node is a leaf level in the hierarchy tree). The ancestor node should be set as a fully-qualified path name such as `/oracle/apps/<applshortname>/<module>/<pagename>.<childNode ID >` where the ancestor childNode ID is whatever childNode (node) you are looping back to.

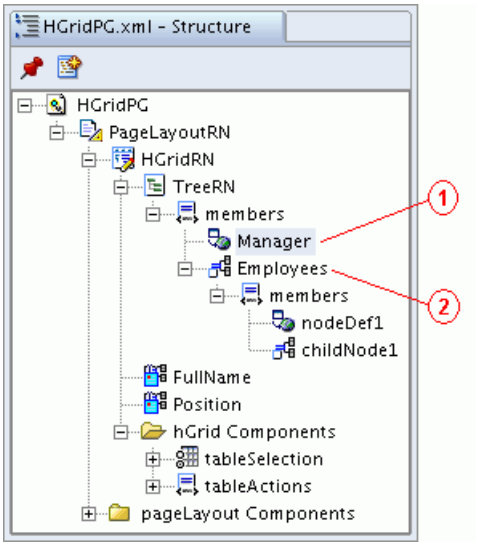
    Attention: For a recursive relationship, as indicated above, you set the ancestor node to the same tree region. However, the "same tree region" refers to the **parent** of the base recursing node and not the recursing node itself.

  - Set the `View Link Accessor` property to the view link accessor name that should be used to retrieve the child nodes at this level.

    Note: Previously, the view link instance used to retrieve the child nodes at a particular level was set via the child node's View Link Instance property. This property is now deprecated and is present only for backwards compatibility. You should only use the View Link Accessor property on the child node to specify the view link instance.

Note: A childNode item can also have other nested members.

Attention: If you intend to support the Export feature on a HGrid, then different viewAttributeNames cannot be used at different levels in the hierarchy column. All levels of the hierarchy column (that is, all nodeDefs) should have the same viewAttributeName. This is analogous to the definition of all other columns of a HGrid. This restriction does not apply if the Export feature is not being used.



Remember that each tree region can contain two members, as called out in the figure above:

1. nodeDefinition - holds the definition for this level, such as icon name, URL, etc.
2. childNode - holds the definition of the view link, pointing to the detail view object.

Step 5: You can define other columns in the HGrid by adding corresponding region items to the HGrid region. The HGrid bean supports all the column types that the table bean supports, including form fields like text inputs and poplists.

Note: HGrid does not yet support the Advanced Table column and column group containers.

Step 6: Set the `View Instance` and `View Attribute` properties on each of the region items representing your columns.

Step 7: To define table actions, select your hGrid region in the Structure pane of OA Extension. Display the context menu and under **New**, choose the **tableActions**. This automatically creates a tableActions named child consisting of a flowLayout region.

Step 8: Specify a standards-compliant ID for the region and leave the `Region Style` as **flowLayout** or set it to **rowLayout**.

Suggestion: If you have only buttons to add to the table actions area, then you can use either layout styles, flowLayout being preferable. However, if you are adding message web beans such as messageChoice or messageTextInput along with buttons to the table action area, then you should use the rowLayout style. Using a flowLayout instead of a rowLayout in this case may cause alignment problems.

Step 9: As of Release 12.2.5, you may use the Display Vertical Gridlines property to specify **True** or **False** to indicate whether to display the vertical gridlines of the table. This property is useful for turning off gridlines in tables displayed on tablet clients.

Step 10: Under the Layout region, layout the children you want to render as table actions, such submitButton or messageChoice. Select the Layout region, and choose **New > Item** from the context menu. Select the new Item that is created and set the `item style` as appropriate.

Enabling Search on a HGrid

You can enable the ability to search on an HGrid. According to the Oracle Browser Look-and-Feel (BLAF) UI Guideline: HGrid Flows specifications, the HGrid search results are displayed in a flat table list. The results table contains a special icon called "View in Hierarchy" in each result row that takes the user to the original HGrid with the focus on the result item.

However, the rendering of the Query region varies, depending on the setting of certain Query region properties and the (user personalized) views, if any, defined by the user:

- If you set the `Initial Panel` property of your Query region to **simple** or **advanced**, an empty flat table displays when the page first renders.
- If you set the `Initial Panel` property of your Query region to **customized** and the user does not have any default view specified, a HGrid displays when the page first renders. However, in this case, once the user selects a view, the following displays, based on the view selected:
  - HGrid - if the selected view does not have any search criteria associated with it.
  - Table - if the selected view does have search criteria associated with it.

- If you set the `Initial Panel` property of your Query region to **customized** and the user has no views defined, the Simple/Advanced Search panels display as expected. However, if you construct your Query region in **resultsBasedSearch** mode, and also specify only the Views panel to display based on the following property settings:
  - `Include Simple Panel` = **false**
  - `Include Advanced Panel` = **false**
  - `Include Views Panel` = **true**

  OA Framework displays the Views panel with an empty Views poplist and HGrid if the user has no views defined.

Step 1: Follow the instructions to set up a search region as described in the Search document.

Step 2: Select your hGrid region in the Structure pane. In the Property Inspector, set the following properties on the **hGrid region**:

- `Search View Usage` - specify the view usage to use for the search results. The view usage *must* have the same attributes as the other view objects that are associated with the nodes of the HGrid.
- `Search Controller Class` - specify a controller class associated with the search results table. OA Framework automatically converts the metadata for the hGrid region to the metadata for the results table by removing the hierarchy column from the hGrid region and rendering the rest of the columns as a flat list. The search controller class allows you to provide additional control over the results table. This controller class can contain `processRequest`, `processFormData`, and `processFormRequest` methods that are invoked during the rendering of the results table as well as for POST requests to the server.

Step 3: Select the **hGrid** region and from the context menu, choose **New > viewInHierarchy** to add a viewInHierarchy named child to the hGrid. The named child creates a **link** item by default. On the **link** item, specify a value for the `Destination URI` property. The `Destination URI` indicates the URI to navigate to when a user selects a result item link from the search results table that directs the user back to the hGrid with the focus on the result item. Use a form submission on this link by setting up a fire action.

Manage this POST request in the Search Controller Class by clearing the state of the hGrid using `OAHGridBean.clearCache` and forwarding back to the same page to redisplay the hGrid region with the focus on the appropriate result. Refer to the Runtime Control section for information on methods that get/set the initial focus path.

According to BLAF guidelines, the View in Hierarchy column typically displays an image icon. You can add the image as a child of the link (just as with a normal link web bean).

Although you should always create the View in Hierarchy column declaratively, you can also get programmatic control of the link web bean as described in the Runtime Control section.

Note: The old mechanism for specifying a URI for the View in Hierarchy link has been deprecated, as described in the OA Framework Coding Standards. Previously, you had to specify the URI in the `View in Hierarchy URI` property of the hGrid region. The new viewInHierarchy named child of the hGrid region takes precedence over this now deprecated property.

Runtime Control

The last step in defining a HGrid is to setup a controller. Refer to the sample controller class `PersonTreePageCO` to continue with the person tree example discussed in the Defining Business Components section.

processRequest method

As is the case with other components in OA Framework, use the `processRequest` method for any custom layout code. Initialize the view object for the root node in this method.

Note: You *must* execute the query on the root view object. In the earlier original implementation of the HGrid, the HGrid used to automatically execute the query for the root view object. To ensure backward compatibility, this behavior is still present in an earlier release of OA Framework, however, consider this behavior deprecated.

As of Release 12.2.5, you may call the `setVerticalGridlinesDisplayed` API in `OAAdvancedTableBean` to set the `verticalGridlinesDisplayed` property at runtime or call the `isVerticalGridlinesDisplayed` API to get the property value.

processFormRequest method

Use the `processFormRequest` method to process form events from the page containing the HGrid. From a HGrid, access data only via the business components. See the PersonTreePageCO for code that walks the business component tree. Refer to the `oracle.apps.fnd.framework.webui.OAHGridQueriedRowEnumerator Javadoc` for additional information.

Getting and Setting the Initial Focus Path

The following methods on `OAHGridBean` are available to get and set the initial focus path for a HGrid:

- `getInitialFocusPath`

- `setInitialFocusPath`

The `setInitialFocusPath` API is used to set the *initial* focus on a particular node of the HGrid. This API applies to both static and dynamic HGrids:

```
setInitialFocusPath(int[] focusRootPath)
```

This signature is used for static HGrids. For example, HGrids on which `setFetchOnDemand(true)` *has not* been called. The `focusRootPath` parameter describes the path from the root of the HGrid to the subnode that *must* be the initial focus root. Each element in this array indicates the child index of the next node on the path.

To focus in on the 2nd child of the 5th child of the root use:

```
int[] focusRootPath = {4, 1};
```

This follows the child at index 4 (which is the 5th child), and then follows that node's child at index 1 (which is its 2nd child).

```
setInitialFocusPath(String[] focusRootPath)
```

This signature is used for dynamic HGrids. For example, HGrids on which `setFetchOnDemand(true)` *has* been called. The `focusRootPath` parameter describes the path from the root of the HGrid to the subnode that must be the initial focus root. Each element in this array indicates the ID of the next node on the path.

Warning: The `setInitialFocusPath` only sets the *initial* focus on the HGrid. For example, when the HGrid is in a pure form with no previously saved state. Future visits to the same HGrid will *not* use the initial focus path. If a HGrid's UI needs to be set back to its initial pure form, then you *must* call `OAHGridBean.clearClientCache`. This applies to both static as well as dynamic HGrids.

Deleting Rows in a HGrid

Although you generally use the `processFormRequest` method to process events on the HGrid, in the case where a user wants to delete rows from a HGrid, you *must* redirect back to the `processRequest` method.

The reason is because OA Framework stores an internal cache to hold on to rows. In general, if you delete a row and then call the `OAHGridBean clearCache` method, it results in a `DeadViewRowAccessException` because the cache is still holding a reference to the deleted row. To avoid this problem, you must always clear the cache first, then delete the row.

Suppose you want to delete some row(s) in a HGrid, where multiple selection is enabled and the Delete button deletes all selected rows. To identify the rows to delete, you need to use OAHgridQueriedRowEnumerator to go through the tree of rows. In the `processFormRequest` method, redirect to the `processRequest` method and use the enumerator to get a handle to the rows selected for deletion, clear the HGrid cache, then delete the rows, as shown in this example:

```
public void processRequest(...)
{
  OAHGridQueriedRowEnumerator enum =
    new OAHGridQueriedRowEnumerator(pageContext, hGridBean);
  Vector listOfRowsToDelete = new Vector(5);
  while (enum.hasMoreElements())
  {
    Row rowToDelete = (Row) enum.nextElement();
    if (rowToDelete != null)
    {
      if ("Y".equals(rowToDelete.getAttribute("DeleteFlag"))
      {
        // Add list Row to be deleted later
        listOfRowsToDelete.add(rowToDelete);
      }
    }
  }
  //Now you have a handle to all the rows to delete

  //Call ClearCache to remove internal cache
  hGridBean.clearCache(pageContext);
  //Delete all the rows in the list
  for (int i = 0; i < listOfRowsToDelete.size(); i++)
  {
    Row rowToDelete = (Row) listOfRowstoDelete.elementAt(i);
    //delete the row
    rowToDelete.remove();
  }

}
```

Note: Previously, if you try to call `clearCache` from the `processFormRequest` method, a developer mode error occurs, due to an

incorrect developer mode check. This has been resolved.

Using FireAction on the Hierarchy Column

You can associate a FireAction element with the hierarchy column of a HGrid to perform a form submit. See Chapter 4: Submitting the Form for more information on submitting a form.

Using Save Model on the Hierarchy Column

To implement a Save Model ("Warn About Changes" dialog with links), on the hierarchy column of the HGrid, you *must* code it manually, by including the following code example:

```
OATreeDefinitionBean webBean = ...
webBean.setAttributeValue(WARN_ABOUT_CHANGES, Boolean.TRUE);
```

Table Row Dynamic Poplist

OA Framework provides programmatic support for the implementation of a choice list (poplist or pulldown) in an updatable multi-row table, such that its poplist values can vary on a row-by-row basis.

Refer to Dynamic Poplists in Standard Web Widgets for programmatic instructions.

Getting a Hold of the viewInHierarchy Named Child's Link Web Bean

Although you should only use declarative steps to enable search on an HGrid, if you must get hold of the link web bean that is in the viewInHierarchy named child of the hGrid region programmatically, use the `findChildRecursive` method:

```
OAHGridBean hGridBean = ...
OALinkBean vihLinkBean = hGridBean.findChildRecursive("vihlink");
// "vihlink" is the ID of the viewInHierarchy link
```

Preserving HGrid State

When a user performs a transaction on your page, you need to ensure that your page preserves its HGrid state. How you support state management for an HGrid depends on the following:

- **Case 1**: If a HGrid cannot have its hierarchy structure change, but its node properties can be altered, it is considered a non-dynamic HGrid. You do not have to do anything extra to support state management for a non-dynamic HGrid.
- **Case 2**: If a HGrid is read-only, it is also a non-dynamic HGrid and you do not have to do anything extra to support state management in this case.
- **Case 3**: If a HGrid's hierarchy structure changes as a result of nodes being added, moved around or deleted, it is considered a dynamic HGrid. To support state management for a dynamic HGrid:

  Step 1: Add the following line of code to your controllers' `processRequest` method:

  ```
  hGridBean.setFetchOnDemand(true);
  hGridBean.setUsePkValuesNodeId(true);
  ```

  Step 2: Remove all calls to `clearClientCache` or `clearCache`.

  Step 3: Change all `initialFocusPath(int[])` calls to `initialFocusPath(String[])`.

The following sections describe in detail how OA Framework supports state management of dynamic HGrids.

HGrid Data Proxy in UIX

Before addressing the details of state management in HGrids, it is necessary to understand what a HGrid proxy object is and how it works in UIX.

In UIX, the state of the HGrid (such as nodes expanded or collapsed, nodes focused into, record navigation, and so on) and all interaction with the data sources is maintained by an object known as the HGrid data proxy. This object needs to be specified on the proxy attribute of the UIX `oracle.cabo.ui.beans.table.HGridBean`. The proxy is actually a fairly complex Java interface (`oracle.cabo.ui.data.tree.HGridDataProxy`) and the UIX team has provided an initial implementation of this interface called `oracle.cabo.ui.data.tree.ClientStateHGridDataProxy`. This particular proxy is a black box to application developers. Application developers are only allowed to create the proxy object and initialize it with the event, state, root and node request parameters as submitted by the client (the browser). Application developers can save the parameters in an appropriate store, such as a session, and use these values to reinitialize the proxy when the HGrid is rerendered. For example, when revisiting a previously rendered HGrid. However, application developers are *not* allowed to peek into the values of these parameters. The values

are internal to the implementation of the `ClientStateHGridDataProxy`.

Proxy Use in the OA Framework

OA Framework manages the `ClientStateHGridDataProxy` transparently for all Oracle E-Business Suite developers. Since most Oracle E-Business Suite require the state of the HGrid to be persisted across visits to the same HGrid, OA Framework caches the event, state, root and node request parameters on the servlet session. These values are then used to reinitialize the HGrid when it is revisited.

While this satisfies requirements for state persistence, the UIX `ClientStateHGridDataProxy` has a significant drawback in that it uses index based tracking for the state. For example, the state might read "expand the 5th child of the 1st child of the root node". If the structure of the HGrid changes through addition or deletion of nodes, any saved state may no longer apply correctly to the HGrid. The most common side effect of this is an `IndexOutOfBoundsException` from the UIX renderers. The other problem is that the state might be applied to the wrong node due to the rearrangement of nodes.

To circumvent these problems, OA Framework provides the `clearClientCache` API on `OAHGridBean` which clears any saved state. This allows changes to the HGrid hierarchy but loses any previous state completely, resulting in the HGrid being redrawn fully collapsed.

Support for Dynamic HGrids

The included UIX release provides a new proxy known as `oracle.cabo.ui.data.tree.DynamicHGridDataProxy`. This proxy was initially designed to support unbounded record navigation in HGrids, such as record navigation, without fetching all rows to the middle-tier. This feature is exposed in OA Framework via the `setFetchOnDemand` API on `OAHGridBean`. The interesting feature of this proxy is that it uses name-based state tracking instead of indices. As a result, it solves the major problems of the `ClientStateHGridDataProxy`. The `IndexOutOfBoundsExceptions` are no longer possible (because there are no indexes) and since the state tracking is by name, there is no possibility of the state being applied to the wrong node.

Optimizing Child Row Retrieval

When any given level of a HGrid is rendered, the rendering engine must first determine whether the node is expandable so that it can render the Expand icon if necessary. To do this, the rendering engine checks whether the node has children by using the HGrid's BC4J view links. This translates to firing the detail view object query just to check for the presence of children, which for large HGrids, can be a serious performance drain.

Using treeChildPresentVOAttr

Since data models in Oracle E-Business Suite often have a master row level attribute that keeps track of the presence or absence of children, you can optimize performance in this case. You can instruct the HGrid to use this attribute instead of firing a detail view object query to determine whether the expansion icon needs to be rendered. In order to do this, set the `treeChildPresentVOAttr` attribute on the `<oa:childNode>` in the metadata. Unfortunately, since this attribute is currently not supported in metadata, you must set this attribute programatically on the `oracle.apps.fnd.framework.webui.beans.nav.OATreeChildBean`, which is the runtime web bean corresponding to `<oa:childNode>`. For example:

```
OATreeChildBean.setChildPresentVOAttr(String childPresentVOAttr)
```

The String parameter in this case is the name of a master view object row attribute that returns "Y" or "N" to indicate whether there are any children.

Important: All Applications should use this feature to avoid unnecessary queries against the database.

Using treeLevelChildCountAttr

Some Oracle E-Business Suite data models also maintain information about the number of children of a master row, at the master row itself. If you have such an attribute available, you may use it instead of `treeChildPresentVOAttr`. The `treeLevelChildCountAttr` has two advantages:

- You can use this attribute to determine whether a node has children and hence, whether an expansion icon needs to be rendered for a node.
- For a HGrid that uses record navigation, when a node is expanded, you can use this attribute to properly adjust the record navigation links ("Previous" and "Next") without triggering the fetch of all rows being displayed.

In order to do use this attribute, set the `treeLevelChildCountAttr` attribute on the `<oa:nodeDef>` in the metadata. Unfortunately, since this attribute is currently not supported in metadata, you must set this attribute programatically on the `OATreeDefinitionBean`, which is the runtime web bean corresponding to `<oa:nodeDef>`. For example:

`OATreeDefinitionBean.setChildCountVOAttr(String childCountVOAttr)`

The String parameter in this case is the name of a master view object row attribute that returns an `oracle.jbo.domain.Number` indicating the number of children. Note that this number must match the number of rows actually returned by the detail view object query.

Rich HGrid Interactions

To enable the rich HGrid interactions introduced in Release 12.2.4, set the profile **FND: Enable Rich Table Interactions** to **True**.

Note: If you set the profile **FND: Enable Rich Table Interactions** to **True**, OA Framework creates a user view of any rich modifications you make to the HGrid. The user view always takes highest precedence when OA Framework displays the HGrid. As a result, if you reorder an HGrid table using the admin Personalization Framework UI, the saved admin personalization will not take effect. If you wish to display the HGrid table columns as ordered according to your admin personalization, you must set the profile **FND: Enable Rich Table Interactions** to **False**.

Note: If you enable search on an HGrid, the HGrid search results that appear under the simple and advanced search panel display in a flat table list. This flat table list does not support rich UI interactions.

Note: From Release 12.2.7, HGrids support rich interactions (such as column freezing and resizing) that were previously only supported in tables.
.

Support for Touch Devices

From Release 12.2.4, OA Framework provides gestures support for rich HGrid interactions. For touch devices, you may **Pan** on a column header to resize or reorder a column, or **Hold** on a column header to reveal the hide/show icon for a column.

Personalization Considerations

- See a summary of HGrid personalization considerations in the Oracle Application Framework Personalization Guide. Also see a summary of Standard Web Widgets personalization considerations if you plan to implement a dynamic poplist in a HGrid.

Known Issues

- See a summary of key HGRID issues with suggested workarounds if available

Related Information

- BLAF UI Guideline(s)
  - HGrid
  - HGrid Flows
- Javadoc File(s)
  - `oracle.cabo.ui.beans.table.HGridBean`
  - `oracle.apps.fnd.framework.webui.beans.table.OAHGridBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OAHGridHierarchyBean`
  - `oracle.apps.fnd.framework.webui.OAHGridQueriedRowEnumerator`
  - `oracle.cabo.ui.action.FireAction`
  - `oracle.cabo.ui.collection.Parameter`
  - `oracle.apps.fnd.framework.webui.beans.nav.OATreeDefinitionBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OATreeChildBean`
  - `oracle.cabo.ui.data.tree.HGridDataProxy`
  - `oracle.cabo.ui.data.tree.ClientStateHGridDataProxy`
  - `oracle.cabo.ui.data.tree.DynamicHGridDataProxy`
- Lesson(s)
- Frequently Asked Questions
  - HGrid FAQ's
- Sample Code
  - PerAllPeopleFVOImpl
  - PersonTreePageCO
  - PerAllPeopleFVL
  - PerAllPeopleFVO

Hide/Show

Overview

As described in the BLAF UI Guideline: Hide/Show specification, the Hide/Show feature lets the user control whether parts of a page are hidden or displayed by selecting a special link (or icon) that toggles between content "disclosed" and "hidden" states.

Figure 1: Example of a Hide/Show control in the hidden (undisclosed) state.

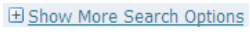
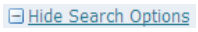


Figure 2: Example of a Hide/Show control in the disclosed state.



Hide/Show can be incorporated into a page design in the following ways. Each implementation is described below.

- Hide/Show in Page Content
- Hide/Show in a Table Row
- Hide/Show in a Table Cell
- Hide/Show in Side Navigation
- Hide/Show Headers

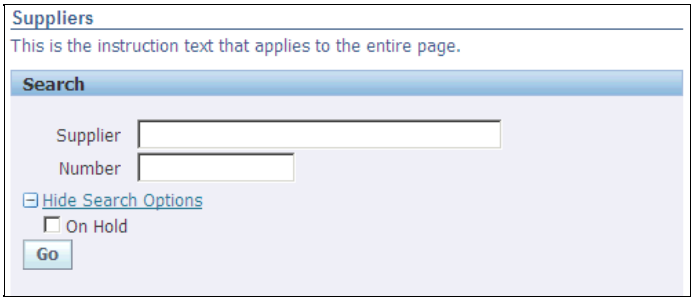
For efficiency, this feature uses Partial Page Rendering (PPR) to redraw only the part of the page that is affected by the Hide/Show component's selection. If PPR is not available (the user is running an unsupported browser or the developer disables the feature), the full page is redrawn for each Hide/Show event.

Hide/Show in Page Content

In this context the Hide/Show control determines whether part of a simple region's contents are hidden or shown. Per the BLAF UI Guidelines, this is typically used in the context of a Search region or within a subheader to control some of its contents.

Tip: If you want to control all the contents beneath a subheader, the Hide/Show Header might be a better choice.

Figure 3: Example of a Hide/Show control in a "Search" region



The OA Framework supports multiple Hide/Show components on a single page. You may even nest them as permitted by the UI Guidelines.

Declarative Implementation

Step 1: to add a Hide/Show component to your page, create region and set its style to **hideShow**. At runtime, the OA Framework will instantiate an `oracle.apps.fnd.framework.webui.beans.layout.OADefaultHideShowBean`.

Note: To ensure that the Hide/Show component is not indented when displayed, add it to a region that does not automatically indent its content (a **stack** or a **header**, for example). If you add it to a region that automatically indents its components (like a **messageLayout** region), the Hide/Show component will not render as specified in the UI Guidelines.

Step 2: Set the following properties for the **hideShow** region:

- `Disclosed Text` - the text to display when the content is disclosed. Per the UI Guidelines, this text should be written in the form of "Hide <*content description*>". If you fail to specify this property, the OA Framework displays "Hide" as the default value.
- `Undisclosed Text` - the text to display when the content is hidden. Per the UI Guidelines, this text should be written in the form of "Show <*content description*>". If you fail to specify this property, the OA Framework displays "Show" as the default value.
- `Initially Disclosed` - controls whether the supplemental content is shown when the page first renders in the session (note that the OA Framework tracks the state of each hide/show component on the servlet session so it will always remember the user's last action when the page is rendered). The default value is "False."

Step 3: Add the content that you want to control (regions and/or items) directly to the **hideShow** region. These regions and/or items will be added as indexed children of the hide/show component.

Step 4 (optional): If you want to control the hideShow bean's disclosure state using a view object attribute, follow these substeps:

- Step 4.1: Define an attribute in the view object you want to reference. This attribute must be a `Boolean` type (in the SQL statement that maps to this attribute, use a `DECODE` to return 0 for false and 1 for true), and it must be updateable so the correct state can be set when the user toggles the control.

- Step 4.2: Set the `View Instance Name` property for the **hideShow** region to point to the view object you want to use.
- Step 4.3: Set the `View Attribute Name` property for the attribute you defined in Step 4.1.
- Step 4.4 Add logic to a controller for the **hideShow** (or higher in the hierarchy) to execute the view object's query. If the query has not been executed when the component is rendered, the `Initially Disclosed` state will be used.If the query has still not been executed when the user selects the link or icon, the OA Framework will throw a developer mode exception if your project has its Developer Test Mode enabled (see Testing for additional information about Developer Test Mode).

Repeating Hide/Show Controls (using Child View Instance)

The `Child View Instance` property is exposed on container beans for the purpose of rendering the containers children multiple times (once for each row in the associated view object). To use this mechanism for displaying a Hide/Show control for each row, follow these steps:

Step 1: Define a layout region (a stack layout, for example) in your page and add a Hide/Show control beneath it.

Step 2: Set the `Child View Instance` property on the layout region to the view object you want to use to control the rendering.

Step 3: Set the `Child View Attribute` property on the layout region to the view object's primary key (note that this does not support composite keys at this time). This step enables the OA Framework to correctly identify the Hide/Show control it creates for each view object row.

Step 4: Set the `View Instance Name` property on the Hide/Show control to the same view object you referenced in Step 2. Also set its **V**`iew Attribute Name` property to a `Boolean` attribute that determines whether the content is hidden or shown (in the SQL statement that maps to this attribute, use a `DECODE` to return 0 for false and 1 for true, and it must be updateable so the correct state can be set when the user toggles the control.).

Step 5: Add the components to the Hide/Show control that you want to display when its state is disclosed. Note that you can add any component you wish, except for a List of Values. Each of these components must bind to a view attribute in the view object you specified in Steps 2 and 3.

Also see the Auto-Repeating Layout topic for additional information about using the `Child View Instance` property.

Hide/Show and the Search "Go" Button

The BLAF UI Guidelines for Hide/Show suggest that the Search "Go" button should change locations based on whether the supplemental content is hidden or shown. For Oracle E-Business Suite products, you may simply add the Go button to your Search region after the Hide/Show component so its location remains the same while the user works in the page. This exception (which was approved by the BLAF UI team although it is not documented in the guidelines) avoids the need to implement a complex workaround in the OA Framework to support the conditional positioning.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

Instantiate OADefaultHideShowBean

Generally, there is little reason to instantiate a Hide/Show control yourself. That said, if absolutely necessary, you should instantiate an `OADefaultHideShowBean` if you want the OA Framework to automatically configure the bean to perform a form submit and handle the hide/show events.

See the `oracle.apps.fnd.framework.webui.OAControllerImpl` Javadoc for other `createWebBean()` signatures.

```
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;

import oracle.apps.fnd.framework.webui.beans.layout.OADefaultHideShowBean;
...
 OADefaultHideShowBean hideShow =
   (OADefaultHideShowBean)createWebBean(pageContext, OAWebBeanConstants.DEFAULT_HIDE_SHOW_BEAN, null, "aName");
```

Once you instantiate the bean, you need to set its disclosed and undisclosed text values as illustrated in the Control Visual Properties section below.

Instantiate OAHideShowBean

You should instantiate an `OAHideShowBean` only if you can't use the declarative implementation, and you need to fully configure the bean (for example, you want the icon/link selection to issue a `GET` instead of a `POST` as the "default" bean does), and you want to implement the event handling yourself (so you will be responsible for manually hiding and showing the supplemental content).

See the `OAControllerImpl` Javadoc for other `createWebBean` signatures.

```
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.layout.OAHideShowBean;

...
 OAHideShowBean hideShow =
   (OAHideShowBean)createWebBean(pageContext, OAWebBeanConstants.HIDE_SHOW_BEAN, null, "aName");
```

Control Visual Properties

You can set the disclosed and undisclosed text values at runtime as shown below. You cannot change the standard hide/show icon.

```
import oracle.apps.fnd.framework.webui.beans.layout.OADefaultHideShowBean;
...
processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   // Get a handle to the hideShow bean if this code is in a controller associated with
   // a parent or grandparent of the bean.
   OADefaultHideShowBean hideShow =
    (OADefaultHideShowBean)findIndexedChildRecursive("<component ID>");

   // Alternatively, if this code is in a controller associated with the hide/show
   // region, simply cast the OAWebBean parameter passed to the processRequest()
   // method.
   OADefaultHideShowBean hideShow = (OADefaultHideShowBean)webBean;

   // Set the undisclosed Text. Always remember to obtain a translated text value
   // from Message Dictionary. NEVER set a hard-coded text value.
   hideShow.setUndisclosedText("<Show...>");

   // Set the undisclosed Text. Always remember to obtain a translated text value
   // from Message Dictionary. NEVER set a hard-coded text value.
   hideShow.setDisclosedText("<Hide...>");

   // Set the default (initial) disclosure state.
   hideShow.setDefaultDisclosed(pageContext, Boolean.TRUE;

}
```

Control Behavior and Data

The OA Framework automatically configures the OADefaultHideShowBean to perform a form submit when the user selects the link or icon. If you need to turn off client side Javascript validation when the form is submitted (you're using the hideShowBean in a data entry page), get a handle to your Hide/Show component and call the following in your processRequest method:

```
hideShow.setUnvalidated(true);
```

If you need to disable Partial Page Rendering for any reason, call the following in your processRequest method:

```
hideShow.setPartialRenderMode(OAWebBeanConstants.PARTIAL_RENDER_MODE_NONE);
```

You can also change the bound value if you're using a view object attribute to determine the disclosure state.

```
hideShow.setViewUsageName(<"viewObjectInstanceName">);
hideShow.setViewAttributeName(<"viewObjectAttributeName">;
```

Handle Hide/Show Events

When using the OADefaultHideShowBean, the OA Framework automatically handles the user selection events to hide and show supplemental content.

If you need to write code to handle these events yourself for any reason, add the following code to your controller's processFormRequest method:

Warning: You should not interfere with the OA Framework's handling of these events (in particular, do not make any changes involving the hide/show state management or the associated view instance if one is attached to the hide/show bean).

```
// Get the name of the event currently being raised.
String hideShowEvent = pageContext.getParameter(OAWebBeanConstants.EVENT_PARAM);

if ((OAWebBeanConstants.SHOW_EVENT.equals(hideShowEvent)) ||
    (OAWebBeanConstants.HIDE_EVENT.equals(hideShowEvent)))
{
...
```

If you have multiple Hide/Show components on the page, you can also get the name of the bean that raised the event by getting the value of the source parameter:

```
// Get the component ID for the bean that raised this event
String hideShowId = pageContext.getParameter(OAWebBeanConstants.SOURCE_PARAM);
```

Note that if you associate a controller with your Hide/Show control (so the webBean parameter in the processFormRequest method is referencing this component), then the source parameter value will equal the webBean's component ID.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  String hideShowEvent = pageContext.getParameter(OAWebBeanConstants.EVENT_PARAM);

  // Get the component ID for the bean that raised this event
  String hideShowId = pageContext.getParameter(OAWebBeanConstants.SOURCE_PARAM);

  // Get the component ID of the current webBean
  String beanId = webBean.getUINodeName();

  if (beanId.equals(hideShowId))...
```

Back Button / Refresh Considerations

The OA Framework automatically handles user selection of the browser Back and Refresh buttons for this component. That said, however, if you're using a view object to manage the disclosure state, you should consider review Supporting the Browser Back Button before proceeding.

Personalization Considerations

- See a summary of Hide/Show personalization considerations in the Oracle Application Framework Personalization Guide.

Hide/Show in a Table Row

You can also use a Hide/Show bean to control whether additional information is displayed for a table row.

See the table [classic | advanced] documentation for information about this.

Declarative Implementation

To implement this as shown in a column in a Classic table (*without* prompts for the data), follow the same steps that are listed in the Hide/Show in Page Content section with the following difference:

- You must specify the `Sort Allowed, Sort by View Attr` and `Initial Sort Sequence` properties to be able to sort the table columns.

If you want the data in the Hide/Show column to include prompts, you must add a **tableLayout** to the column first, and then add your **hideShow** region as described above.

Runtime Control

See the Hide/Show in Page Content section for runtime control instructions.

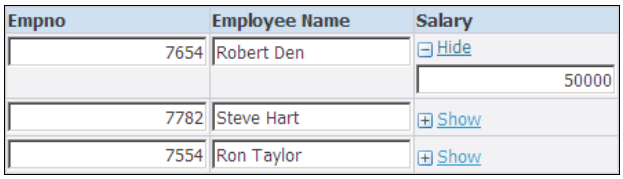
Personalization Considerations

- See a summary of Hide/Show personalization considerations in the Oracle Application Framework Personalization Guide.

Hide/Show in a Table Cell

In this context, control whether supplemental information is shown in a single table cell.

Figure 5: Example of a Hide/Show control for a table cell



Declarative Implementation

To implement this as shown in a table column (*without* prompts for the data), follow the same steps that are listed in the Hide/Show in Page Content section with the following difference:

- You must specify the `View Instance` and `View Attribute` names. The `View Instance` should reference the same view object that the table is using, and the `View Attribute` must be a `Boolean` type (in the SQL statement that maps to this attribute, use a `DECODE` to return 0 for false and 1 for true), and it must be updateable so the correct state can be set when the user toggles the control.

If you want the data in the Hide/Show column to include prompts, you must add a **tableLayout** to the column first, and then add your **hideShow** region as described above.

Runtime Control

See the Hide/Show in Page Content section for runtime control instructions.

Personalization Considerations

- See a summary of Hide/Show personalization considerations in the Oracle Application Framework Personalization Guide.

Hide/Show in Side Navigation

The OA Framework does not support Hide/Show in a side navigation as shown in Figure 6.

Figure 6: Example of a Hide/Show control in a Side Navigation



Hide/Show Headers

If you want to hide or show all the contents beneath a subheader as shown in Figures 6 and 7, use the Hide/Show Header bean.

Note: This bean does not have associated disclosed and hidden text properties since only the Hide/Show icon toggles when the user selects it. The header text remains the same in both states. Furthermore, as of Release 12.2.6, users of iOS touch devices may tap on a Hide/Show header to expand or collapse its content.

Figure 7: Example of a Hide/Show Header in the disclosed state

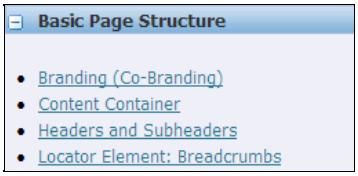


Figure 8: Example of a Hide/Show Header in the hidden (undisclosed) state



Declarative Implementation

To add a Hide/Show Header to your page, create a region and set its style to **hideShowHeader**. At runtime, the OA Framework will instantiate an `OAHideShowHeaderBean`.

The only other region property that you must set is the `Text` to display. You can also change the default `Initially Disclosed` value from **False** to **True**.

Finally, add contents to the **hideShowHeader** as you would for any other header region.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or easily extended.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

Instantiate

See the `OAControllerImpl` Javadoc for other createWebBean signatures.

```
Import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.layout.OAHideShowHeaderBean;
...
 OAHideShowHeaderBean header =
   (OAHideShowHeaderBean)createWebBean(pageContext, OAWebBeanConstants.HIDE_SHOW_HEADER_BEAN, null, "aName");
```

Control Visual Properties

The only visual property that you can set for this bean is the header's text as shown:

```
// Always remember to obtain a translated text value from Message Dictionary
// NEVER set a hard-coded text value.
header.setText(pageContext, "<some value>");
```

Control Behavior and Data

As described in the Hide/Show in Page Content section above, you can change the bound value if you're using a view object attribute to determine the disclosure state. You can also turn off client-side Javascript validation if needed.

Handle Hide/Show Events

The OA Framework automatically handles hiding and showing the header's content. There is no reason to write code to handle this yourself.

Back Button / Refresh Considerations

The OA Framework automatically handles user selection of the browser Back and Refresh buttons for this component. That said, however, if you're using a view object to manage the disclosure state, you should review Supporting the Browser Back Button before proceeding.

Personalization Considerations

- See a summary of Hide/Show personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

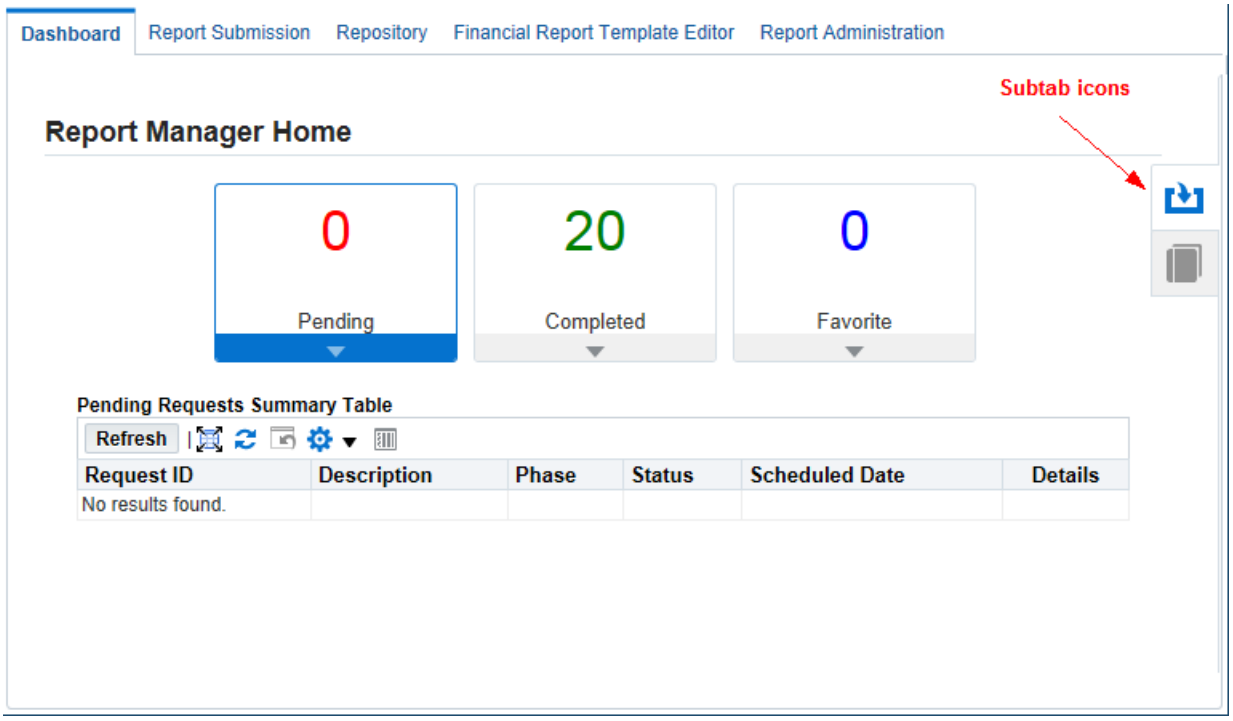
- BLAF UI Guidelines
  - Hide/Show
- OA Framework Developer's Guide
  - Tables [Classic | Advanced]
  - Testing OA Framework Applications
  - OA Framework Controller Coding Standards
  - Partial Page Rendering
  - Supporting the Browser Back Button
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAHideShowBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OADefaultHideShowBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAHideShowHeaderBean`
  - `oracle.apps.fnd.framework.webui.OAControllerImpl`
- OA Framework ToolBox Tutorial / Sample Library
  - `oracle.apps.fnd.framework.toolbox.tutorial.webui.SupplierSearchPG`
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.BasicStructPG`

Hide/Show SubTab Layout

Overview

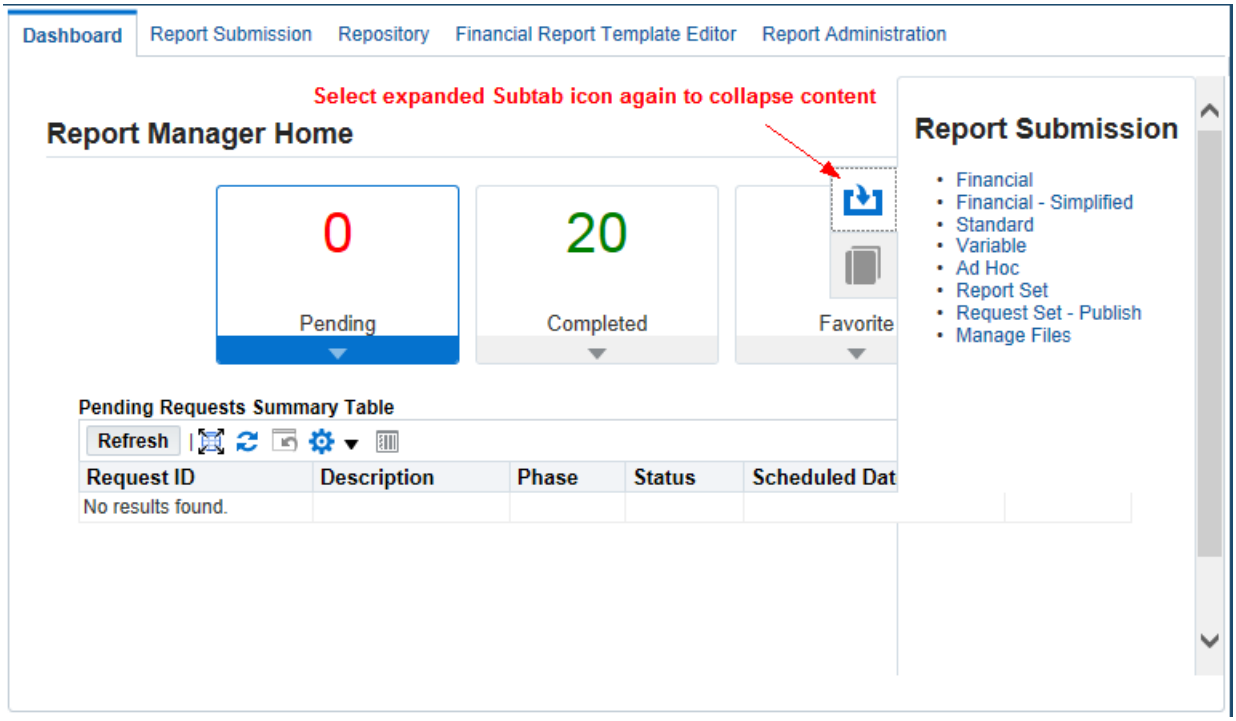
Beginning in Release 12.2.5, you can create a Hide/Show subtab layout to render additional Hide/Show content that is relevant to a page. This new feature renders a Hide/Show region containing a subtab layout that renders vertically on the page. Each tab of the Hide/Show subtab layout displays an icon, that when selected, slides out to display content. By default, the Hide/Show subtab layout renders in collapsed mode displaying an icon for each subtab at the right edge of the page layout.

Figure 1: Example of a Hide/Show subtab layout region in "collapsed" mode.



When a user selects one of the subtab icons, the associated content region slides out left (expands) to display the subtab content. Upon selecting the same icon in expanded mode, the Hide/Show subtab layout slides back to hide its content.

Figure 2: Example of a Hide/Show subtab layout region in "expanded" mode.



Note: Hide/Show subtab layout is not a suitable layout choice if you wish to design a page for performing transactions. It is better suited for displaying navigable content that is relevant to a page, as in the Ancillary Content links shown in the ToolBox Tutorial Home page.

Note: A single subtab for the Hide/Show subtab layout requires a set of two icons: one in color and the other in shaded grey. When a user selects a subtab, the colored icon appears. When a subtab is not selected, the grey-shaded icon appears. You only need to associate the colored icon with the subtab's link.

Declarative Implementation

To add a Hide/Show subtab layout to your page:

Step 1: Create a **pageLayout** region as you normally would.

Step 2: Select the **pageLayout** region in the JDeveloper Structure pane, right-click and select **HideShowSubTabLayout** from the context menu. This creates **hideShowSubTabLayout** as a named child under the **pageLayout** region.

Step 3: Under **pageLayout Components**, select the region under the **hideShowSubTabLayout** region. From the context menu, select **New > subTabLayout**. This creates a **SubTabLayout** region.

Step 4: Select the region under the **subTabLayout** region and from the context menu, select **New > subTabs**. Refer to the steps in the SubTab topic to complete the creation of your subtab navigation.

As of Release 12.2.6, the **hideShowSubTabLayout** component implicitly sets default values for the named child **SubTabLayout** region and its SubTabBar items. The following default values for the properties of the **SubTabLayout** region and its children ensure proper configuration of your Hide/Show subtab layout:

1. `Display Type` - defaults to **iconOnly.**
2. `Orientation` - defaults to **vertical**.
3. `Action Type` - defaults to **firePartialAction** (for each **link** item under the named child SubTabBar region of the containing SubTabLayout region).

Note: Any values that you set for these properties in OA Extension will not take effect as they will be overridden by the default values listed above.

Step 5: Associate an icon with each subtab **link** item you create in Step 6 of the SubTab topic. For each **link** item, specify the URI of the icon in the `Icon URI` property and set the `Text` property.

Step 6: Save your work.

Runtime Control

When the OA Framework instantiates a page with an `OAHideShowSubTabLayoutBean`, it creates an `oracle.apps.fnd.framework.webui.beans.layout.OAHideShowSubTabLayoutBean` containing an `oracle.apps.fnd.framework.webui.beans.layout.OASubTabLayoutBean` which internally contains an `oracle.apps.fnd.framework.webui.beans.layout.OASubtabBarBean` web bean hierarchy as follows:

```
----->OAHideShowSubTabLayoutBean (named child of page layout)
-------> OASubTabLayoutBean
---------> OASubtabBarBean(Named child of sub tab layout bean)
-----------> OALinkBean (Tab One)
-----------> OALinkBean (Tab Two)
```

Instantiate

There is no reason to instantiate an `OAHideShowSubTabLayoutBean` and its components yourself; you should create this declaratively.

Usage Restrictions

- You may add only one **HideShowSubtabLayout** region to a page. Declaratively, OA Extension reinforces this by not allowing you to add multiple **HideShowSubTabLayouts** regions to a page.
- The **HideShowSubTabLayout** region does not render in the printable facet of a page.
- Classic or advanced tables are not supported within a **HideShowSubTabLayout** region.

Accessibility Considerations

- Using the keyboard, you can access the Hide/Show sub tab layout by pressing **[Tab]** until focus reaches the first subtab of the layout. Pressing **[Tab]** again focuses on the subsequent component within that subtab.
- To navigate between subtabs, use the navigation keys. Use **[Down Arrow]/[Right Arrow]** to focus on the next subtab. Use **[Up Arrow]/[Left Arrow]** to focus on the previous subtab. Press **[Enter]** to show or expand the focused subtab.

Personalization Considerations

- See a summary of Hide/Show Sub Tab Layout personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAHideShowSubTabLayoutBean`
  - `oracle.apps.fnd.framework.webui.OAHideShowSubTabLayoutHelper`
- OA Framework ToolBox Tutorial / Sample Library

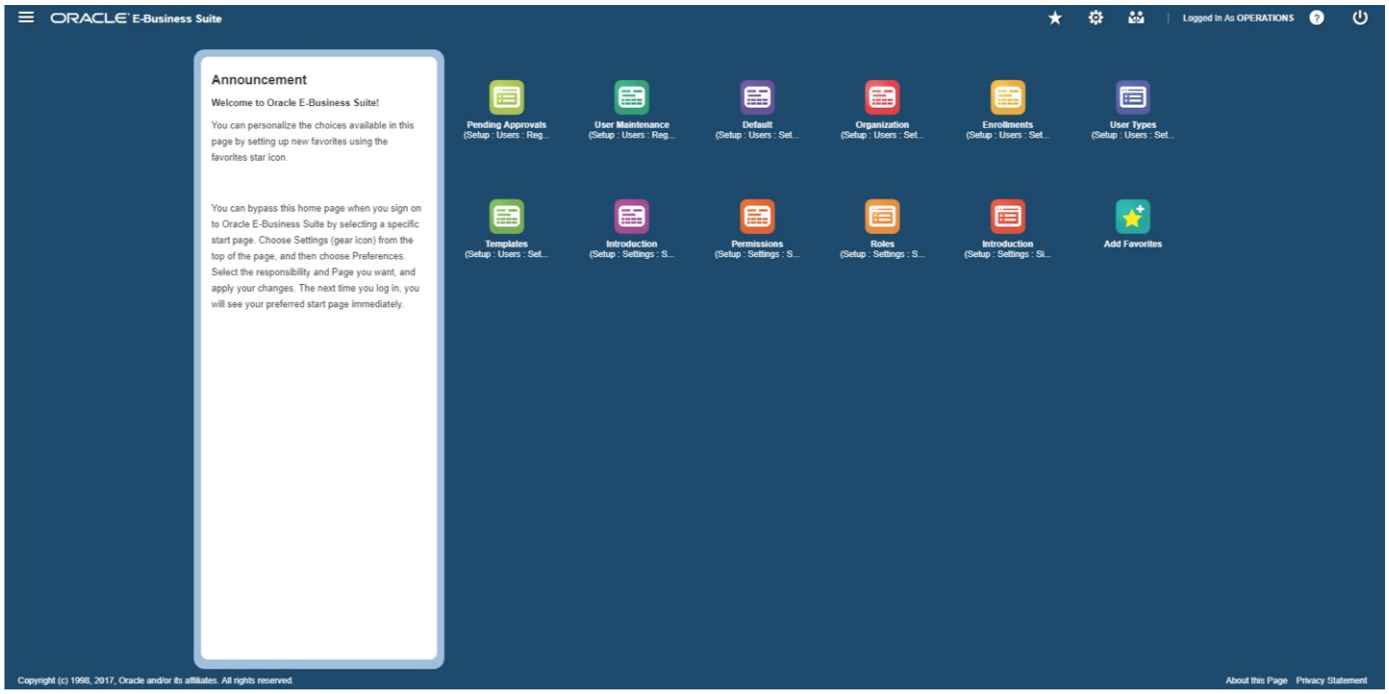
Home Page

Overview

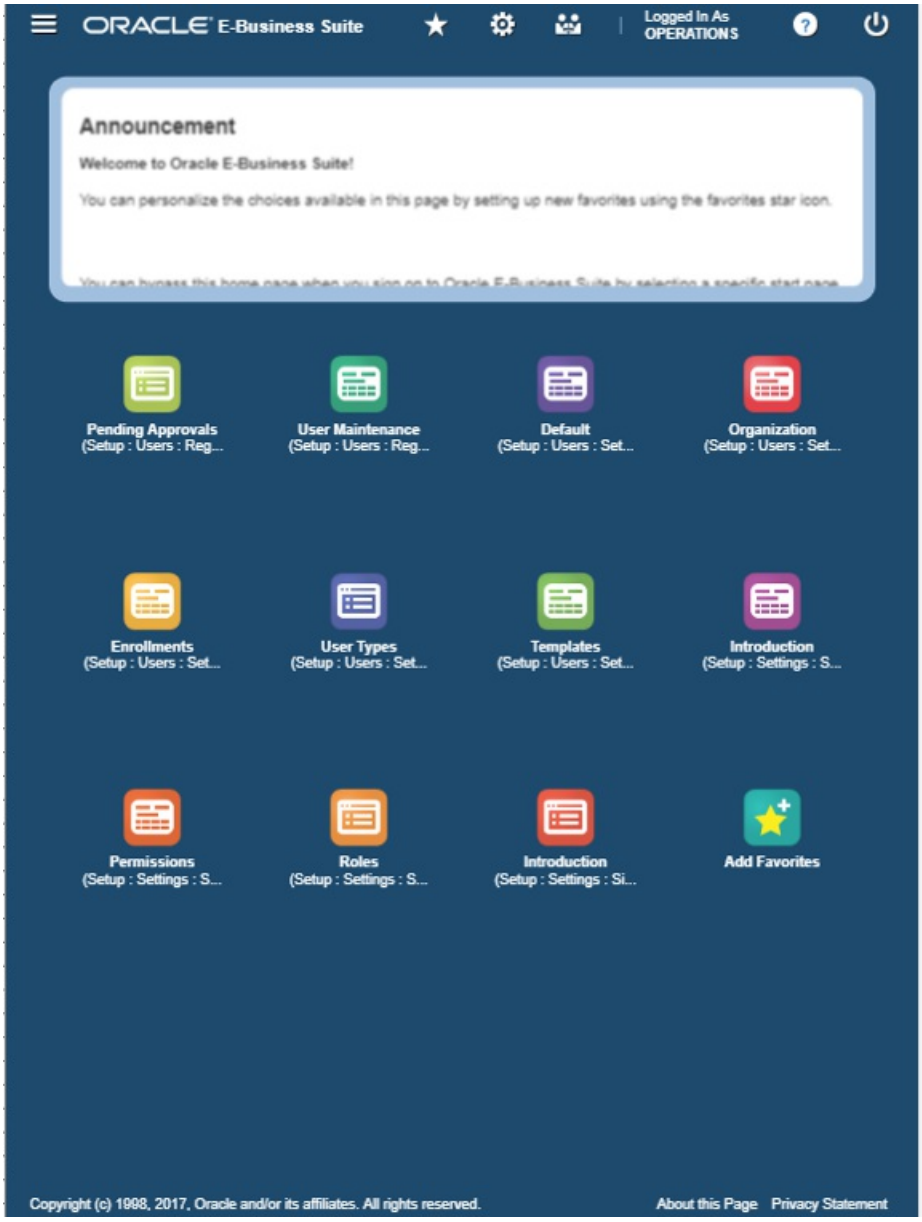
Release 12.2.9 introduced several important Home Page enhancements, including the following:

- Simplified Home Page display of the announcement text and responsibility tiles in a responsive pattern using the new Masonry layout.
- Redirection to Home Page (Framework only, Framework tree, simplified) based on user preference value rather than profile value settings.
- Dedicated Home Page layout options for desktop, tablet, and mobile phone. These are illustrated by the three screenshots below.

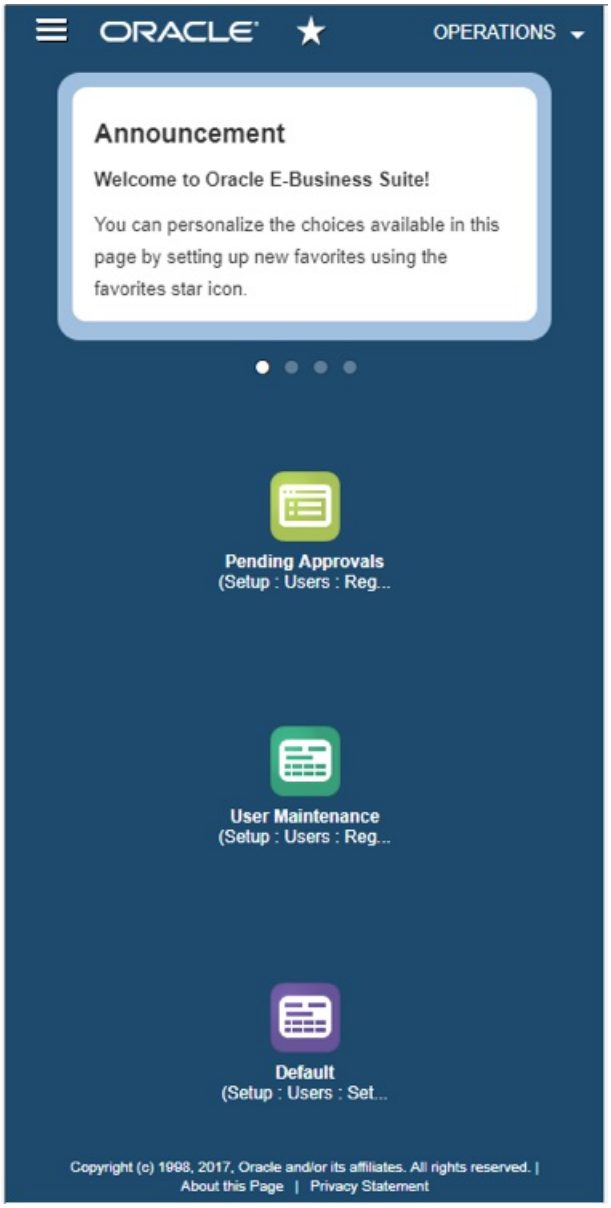
Simplified Home Page with Masonry Layout - Desktop



Simplified Home Page with Masonry Layout - Tablet (iPad)



Simplified Home Page with Masonry Layout - Mobile Phone



This section describes how to set the OA Framework-based Home Page style in Oracle E-Business Suite Release 12.2.4 (and higher), and use the Simplified Home Page.

Note: Be aware of the following important points.

- A "Warning" message displays at the top of the Home Page if the current Look-and-Feel (LAF) is not or does not extend the Alta Look and Feel, or if you are using an unsupported browser version. Currently, OA Framework performs version checks only for the Microsoft Internet Explorer (IE) and Firefox browsers. For information about supported browsers, refer to My Oracle Support Knowledge Document 389422.1, *Recommended Browsers for Oracle E-Business Suite Release 12*.
- From Release 12.2.6, all Home Page styles can now display the name of the instance, in the format "Instance: <*sid*>", below the In-Context Branding region of the Home Page's global header.
  - Display of the instance name on the Home Page is enabled by setting the profile option FND_SHOW_INSTANCE_NAME to "True".

Contents

- Simplified Home Page
- Home Page Profile Options
- Favorite Icons
- Accessibility Support
- Personalization Considerations
- Touch Device Considerations
- Known Issues
- Related Information

Simplified Home Page

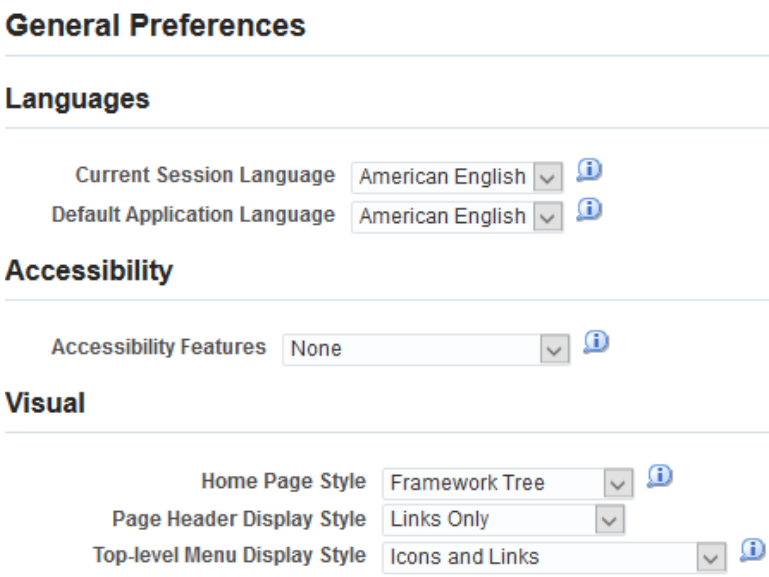
The new Simplified Home Page, as introduced in the Overview above, provides a straightforward user experience and lets you easily create a custom role-based Home Page. It consists of two major sections: a welcome **announcement,** and a list of iconized functions that provide fast and easy access to frequently-used favorites. Favorites appear in a grid-like pattern as labeled icons to the right of the **announcement**. Along with the streamlined features of the Universal Global Header, the Simplified Home Page provides uncluttered access to the following features:

- **Worklist notification and access** - Select the **Worklist** icon in the Universal Global Header to view or navigate to your Worklist page. Your Worklist notification appears to the right of the **Worklist** icon.
- **Function and Menu navigation** - Select the **Navigator** icon located in the Universal Global Header to navigate to functions and menus with an easy-to-use rich drop-down menu.
    - In Release 12.2.7, the following changes were made to navigation:
        - The Navigator icon was replaced with a "hamburger" icon that appears as the first global header icon before the Oracle logo.
        - If a page has a title, the "Add to Favorites" icon (white star) appears next to the page title of bookmarkable pages.
        - If a page does not have a title, it will appear in the "Favorites" popup (as was the case in Release 12.2.6).
        - If a bookmarkable page is already bookmarked, the "Manage Favorites" icon (orange star) appears next to the page title.
- **Favorites navigation** - If you prefer to navigate to your Favorites from a list, select the **Favorites** icon located in the Universal Global Header.
- **Favorites customization** - If you have no Favorites defined when you first log in, select the **Add to Favorites** icon to go to the **Manage Favorites** page where you can add new Favorites to your Home Page, reorder Favorites or remove existing Favorites. For more information on how to manage your Favorites, refer to the "Getting Started with Oracle E-Business Suite" chapter in the *Oracle E-Business Suite User's Guide*.

  Note: In a proxy session, the **Add to Favorites** icon does not appear on the Home Page for the proxy user.
- **Announcement personalization** - Select the **Personalize Page** option from the **Settings** icon in the Universal Global Header to personalize the Home Page's announcement.

Home Page Profile Options

An important associated change in Release 12.2.9 is the use of profile options. In previous releases, the profile option APPLICATIONS_HOME_PAGE determines the look-and-feel of the Oracle Self-Service Applications Personal Home Page. With the change to use Masonry, the user preference set in the Settings > Preferences page takes precedence over the profile option value. The values set in this page for both home page and icon style are stored in the FND_USER_PREFERENCES table. If no value is set here, the value from APPLICATIONS_HOME_PAGE profile option is used. This is illustrated by the following screenshot:



The following system profile options control the behavior of the Home Page in Release 12.2.4 and above:

- **Self Service Personal Home Page Mode** - From Release 12.2.4, this profile option supports the following values:
  - **Framework only** - displays the Home Page from Release 12.2.3 and prior, based on the value of profile option **FND: Disable Configurable Home Page**.
  - **Framework Tree** - displays the Home Page from Release 12.2.3 and prior, based on the value of profile option **FND: Disable Configurable Home Page**.
  - **Framework Simplified** - displays the Simple Home Page from Release 12.2.4 and above.
- **FND: Disable Configurable Home Page** - This profile accepts a value of **False** or **True** to determine whether to display the Configurable Home Page with the Tree-based Navigator or Home Page with the flat list Navigator, respectively, when the **Self Service Personal Home Page Mode** profile is set to **Framework Only** or **Framework Tree.**

To display the new Simplified Home Page, set the **Self Service Personal Home Page Mode** profile option to **Framework Simplified**. The table below summarizes the Home Page behavior that results from the various profile option combinations.

Profile Option		Resulting Home Page in Release 12.2.4 and above	Comments
Self Service Personal Home Page Mode	**FND: Disable Configurable Home Page**		
Framework Only (default)	**True**	Home Page with flat list Navigator (as rendered in Release 12.1)	Backward compatible with pre-12.2.4
Framework Only (default)	**False** (default)	Configurable Home Page with Tree-based Navigator (as rendered in Release 12.2)	Backward compatibility with pre-12.2.4
Framework Tree (Configurable Navigator)	**True**	Home Page with flat list Navigator (as rendered in Release 12.1)	Backward compatible with pre-12.2.4
Framework Tree (Configurable Navigator)	**False** (default)	Configurable Home Page with Tree-based Navigator (as rendered in Release 12.2)	Backward compatible with pre-12.2.4
Framework Simplified	**True**	Simple Home Page	"FND: Disable Configurable Home Page" value is overridden
Framework Simplified	**False** (default)	Simple Home Page	"FND: Disable Configurable Home Page" value is overridden

Favorite Icons

When a user first selects a function as a Favorite, OA Framework checks for any seeded icon assigned to that function and uses it for the Favorite icon on the Simple Home Page, If a seeded icon is not available, OA Framework randomly assigns an icon to the Favorite from a pool of default icons. The icon initially associated with that Favorite then becomes its permanent icon and will appear with the Favorite each time the user logs in.

Each Favorite icon appears with a label consisting of the function name, followed by an ellipsis that displays its corresponding responsibility, if any, as a tool-tip upon hover. The user can change the text using the Manage Favorites page.

Accessibility Support

You may use the keyboard to access the Simple Home Page as follows:

Keyboard Support

- **[Tab]** - moves the focus to the next page element. Note that only the first Favorites icon is in the tab order. If the current focus is on any Favorite icon, pressing **[Tab]** will shift the focus to the first tab-able element on the page.

- **[Home]** - with focus on any Favorites icon, pressing **[Home]** will shift the focus to the first Favorite icon in the grid.
- **[End]** - with the focus on any Favorite icon, pressing **[End]** will shift the focus to the last Favorite icon in the grid.
- **[Left/Right Arrow]** - with the focus on any Favorite icon, pressing **[Left/Right Arrow]** will shift the focus to the previous/next (or next/previous in a Right to Left (RTL) session) Favorite icon in the grid. If the currently selected Favorite is the last item in the grid, pressing **[Right Arrow]** (**[Left Arrow]** in a RTL session) will shift the focus to the first Favorite in the grid and vice-versa.
- **[Up/Down Arrow]** - with the focus on any Favorite icon, pressing **[Up/Down Arrow]** will shift the focus to the next Favorite in the same column. If the selected Favorite is last in it's column then the first Favorite in the next column will be highlighted and vice-versa.
- **[Spacebar]** - launches the related form/page for the selected Favorite icon in the grid.

Personalization Considerations

- See a summary of Home Page personalization considerations in the Oracle Application Framework Personalization Guide.

Touch Device Considerations

- See a summary of considerations for the Home Page feature on touch devices in the Mobile Applications topic.

Known Issues

- None

Related Information

- OA Framework Developer's Guide
  - Branding
  - Buttons (Global)
  - Masonry
  - Oracle Application Framework Profile Options

Images in Your Pages

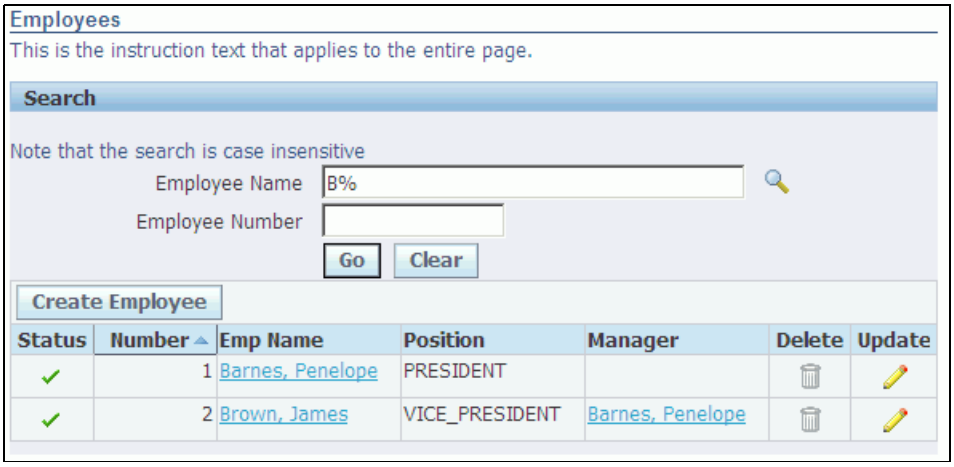
Overview

For the purposes of discussing how to incorporateimages into your pages, we classify Oracle Application Framework application images into the following three categories:

Category	Description	Images Added By
1	Core component images (for example: buttons, tabs, the List of Values (LOV) icon, train steps, table column sorting icons, the Hide/Show icons, message icons, the 1-pixel spacer image and so on)	UIX
2	Corporate and product branding images.	Page Developers
3	Images added to support a page's functionality as illustrated in Figure 1 (see the Status, Delete and Update icons in the search results table).	Page Developers

This document describes how to add images in the third category. See the Branding document for additional information about the second category. UIX renders images in the first category automatically.

Figure 1: Typical OA Framework page showing all three categories of images



Contents

- Declarative Implementation
- Runtime Control
- Using Image Beans to Fetch Images From the Database
- Personalization Considerations
- Known Issues
- Related Information

Declarative Implementation

Step 1: Locate the image that you want to add to your page using the Ancillary Graphic Repository (external link) or the Oracle E-Business Suite Icon Repository (external link). You'll need the image's name and its size for Step 2.

Step 2: Create a new item with a standards-compliant `ID` and set the following properties for the item:

- Set the `Style` property to **image**.
- Set the `Image URI` to the image's name. For example, to add the Update (enabled) icon shown in Figure 1 above, you would set this value to **updateicon_enabled.gif**.
- Set the `Additional Text` property to the value listed in the image's source repository. This value is displayed as tooltip text, and used for accessible component access (and as such is required by the Accessibility coding standards). If an image is purely decorative, then set this to **DECORATION** to generate an empty string for its value (which appears in the HTML source as alt="" for the image tag).
- Set the `Height` to the image's published height.
- Set the `Width` to the image's published width.

Note: The `Height` and `Width` properties must be set for any images that you add to a page. See the View Coding Standard V30 for additional information.

- If selecting the image should perform an HTTP GET, set the `Destination URI` to a target page (for example: `OA.jsp?page=/oracle/apps/fnd/framework/toolbox/samplelib/webui/SampleBrowserPG&retainAM=Y`). If selecting the image should perform an HTTP POST, see Submitting the Form for special configuration instructions.

At runtime, OA Framework instantiates an `oracle.apps.fnd.framework.webui.beans.OAImageBean`.

Tip: If you need to conditionally display different images in a table (for example, a Delete image is enabled or disabled on a row-level basis), use a Table Switcher. You can also use a bound value as shown in the Runtime Control example below.

Storing and Deleting Product Specific Dynamic Images

OA Framework provides the following APIs for storing and deleting product specific temporary and permanent dynamic images under /OA_HTML/fwk/t and /OA_HTML/fwk/p.

Note: These directories are not striped by product, therefore we recommend that all image names be prefixed with the product code.

Use these APIs to return the physical path to where the images can be stored:

```
public String getTemporaryImageLocation();
public String getPermanentImageLocation();
```

Use these APIs to return the virtual path to where the images can be stored:

```
public String getTemporaryImageSource();
public String getPermanentImageSource();
```

Note: This path can be used to build URLs for your images.

Use this API to delete a permanent or temporary image:

```
Public void deleteImage(String imageName, String imageType);
```

Use this API to ensure that your image name is unique:

```
Public String generateUniqueImageName(String imageName, String imageType);
```

Note: Image type can be `OAWebBeanConstants.OA_TEMPORARY_IMAGE` or `OAWebBeanConstants.OA_PERMANENT_IMAGE`.

In a typical customer environment, multiple users and JVMs access the same physical directory. In order to guarantee a unique image name, an empty file corresponding to the image name is created. File extensions provided in the image name will be respected. If you are attempting to create a gif file, make sure your image name is `someImageName.gif`. It is the responsibility of the application using the temporary images, to delete such images at the end of the transaction. See the javadoc for more information.

How to use these APIs is shown in the Runtime Control example below.

Runtime Control

To add an image to your page programmatically, you *must* specify the relative image path in addition to setting the height, width, and alternative text. However, (when adding an image declaratively, it is sufficient to specify just the image's name and the alternative text, as shown in this example.

```
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.OAImageBean;
...
public void processRequest(OAPageContext pageContext, OAWebBean webBean);
{
  super.processRequest(pageContext, webBean);
  ...
  OAImageBean dots =
    (OAImageBean)createWebBean(pageContext, OAWebBeanConstants.IMAGE_BEAN, null, "dotsImage");
  dots.setSource(OAWebBeanConstants.APPS_MEDIA_DIRECTORY + "cc_dotsprocess.gif");
  dots.setShortDesc(pageContext.getMessage("FND","FND_DOTS_SHORT_DESC"));
  dots.setHeight("35");
  dots.setWidth("8");
  webBean.addIndexedChild(dots);
}
```

You can also add an image to your page using bound values, as shown below. (See the Bound Values topic for additional information). The following example (from the `PoSummaryCO` class in the ToolBox Tutorial application) uses a view object attribute value to set the correct status image on a row-level basis).

```
import oracle.cabo.ui.data.BoundValue;
```

```
import oracle.cabo.ui.data.bind.ConcatBoundValue;
import oracle.cabo.ui.data.bind.FixedBoundValue;
import oracle.apps.fnd.framework.webui.OADataBoundValueViewObject;
import oracle.apps.fnd.framework.webui.beans.OAImageBean;
...
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
  OAImageBean statusImageBean =
     (OAImageBean)table.findIndexedChildRecursive("StatusImage");
  /*
  ** Note that you may define an image bound value without specifying the APPS_MEDIA_DIRECTORY,
  ** however, we include this example to show you how to concatenate a fixed bound    value
  ** with a data bound value.
  */
  // Define the OA Framework image directory
  FixedBoundValue imageDirectory = new FixedBoundValue(APPS_MEDIA_DIRECTORY);

  // Define a binding between the image bean and the view object attribute that it
  // will reference to get the appropriate .gif image value name.
  // Note that the corresponding attribute values are obtained using a decode() in the
  // POSimpleSummaryVO view object. Another attribute must be defined
  // to return the text to be displayed for the shortDesc.
  OADataBoundValueViewObject statusBinding =
    new OADataBoundValueViewObject(statusImageBean, "StatusImage");
  OADataBoundValueViewObject statusTextBinding =
    new OADataBoundValueViewObject(statusImageBean, "StatusText");

  statusImageBean.setAttributeValue(SHORT_DESC_ATTR, statusTextBinding);
  // Concatenate the image directory with the actual image name (as retrieved
  // from the view object attribute decode() statement)

  ConcatBoundValue statusCBV
    = new ConcatBoundValue(new BoundValue[] {imageDirectory, statusBinding});

  // Finally tell the image bean where to get the image source attribute

  statusImageBean.setAttributeValue(SOURCE_ATTR, statusCBV);

}
```

The following is an example of how the APIs for storing and deleting product specific dynamic images may be used:

```
...

    //At the beginning of a transaction
    //Image returned would look like HRCar2.gif, if say another image
    //with the prefix HRCar.gif already exists in that directory.

    String imageName = pageContext.generateUniqueImageName("HRCar.gif",
      OAWebBeanConstants.OA_TEMPORARY_IMAGE);
    //assuming your image generation program takes full path to the image being created
    someAPIForCreatingTheGif(getTemporaryImageLocation() + imageName);
    //create the image bean
    OAWebBeanFactory factory = pageContext.getWebBeanFactory();
    OAImageBean image = (OAImageBean)factory.createWebBean(pageContext, IMAGE_BEAN);
    image.setSource(getTemporaryImageSource() + "infoicon_active.gif");
...

    //At the end of the transaction
    pageContext.deleteImage(imageName, OAWebBeanConstants.OA_TEMPORARY_IMAGE);
```

Using Image Beans

From Release 12.2.10, users can store images in the database and easily display those images by providing the view name and attribute name. Users can also provide a target frame attribute. This is all done via the use of image beans.

Two aspects of this will be considered in turn:

- Setting up image beans to fetch images from the database
- Optionally setting up image beans with a target frame

Setting Up Image Beans to Fetch Images from Database

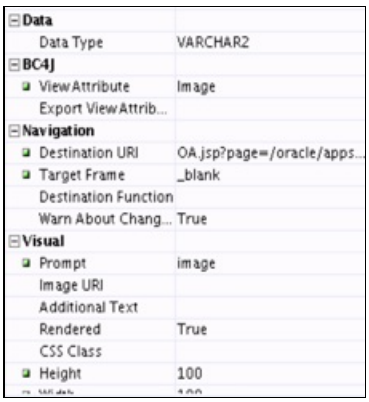
To set up the Image Bean item, a container such as a header ot table needs to be created. An imagebBean can be displayed as a single view image, or inside a table or list. To implement this, the developer has to make the following changes to the image bean.

- The view instance and view attribute must be defined with corresponding entries.
- The image URI should be left empty (if the developer defines "Image URI", it will take precedence over the view instance and view attribute).

There are two scenarios that need to be validated, inspected and tested.

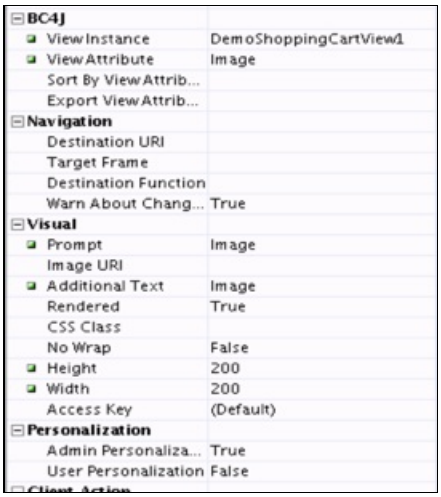
- Image Bean inside a Table or Advanced Table region.
- Image Bean as a single image view.

Including Image Bean inside search result table region



- Define image column in the table.
- Runtime control:
  - Setters:
    - `public void setViewName(String viewName)`
    - `public void setViewAttributeName(String viewAttributeName)`
  - Getters:
    - `public String getViewName()`
    - `public String getViewAttributeName()`

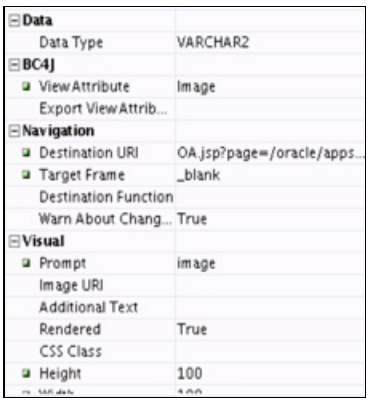
Displaying Image Bean as a Single View



- Define image item.
- Runtime control:
  - Setters:
    - `public void setViewName(String viewName)`
    - `public void setViewAttributeName(String viewAttributeName)`
  - Getters:
    - `public String getViewName()`
    - `public String getViewAttributeName()`

Setting Up Image Beans with a Target Frame

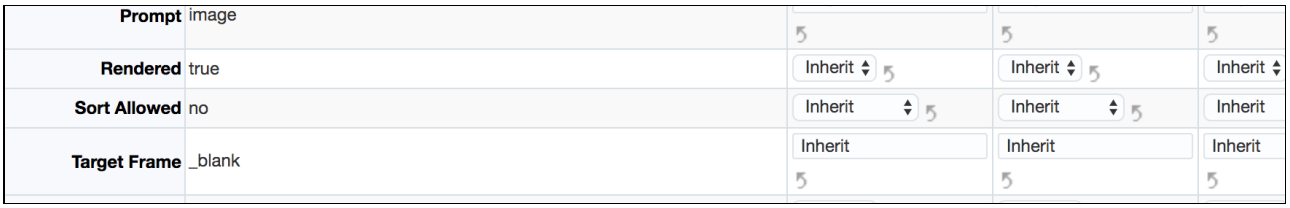
A target frame defines the target window where the provided link should open. This property is enabled and available under the Navigation group.



Properties the user can define for the target frame include the following.

Property	Description
_blank	The linked page opens in a new window.
_self or empty	The linked page opens in the same frame.
<frame name>	The linked page opens in the named frame.

The target frame attribute can be personalized at runtime.



Accessibility

Addition of dynamic additional text, or a short description, can be accomplished using a bound value:

```
OADataBoundValueViewObject itemNameBinding = new OADataBoundValueViewObject(image,"ItemName");
image.setAttributeValue(SHORT_DESC_ATTR, itemNameBinding);
```

Personalization Considerations

- See a summary of Images in Your Pages personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guideline(s)
  - Ancillary Graphic List (Repository)
  - Icon Repository
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.OAImageBean`

Include Content (URL and Servlet)

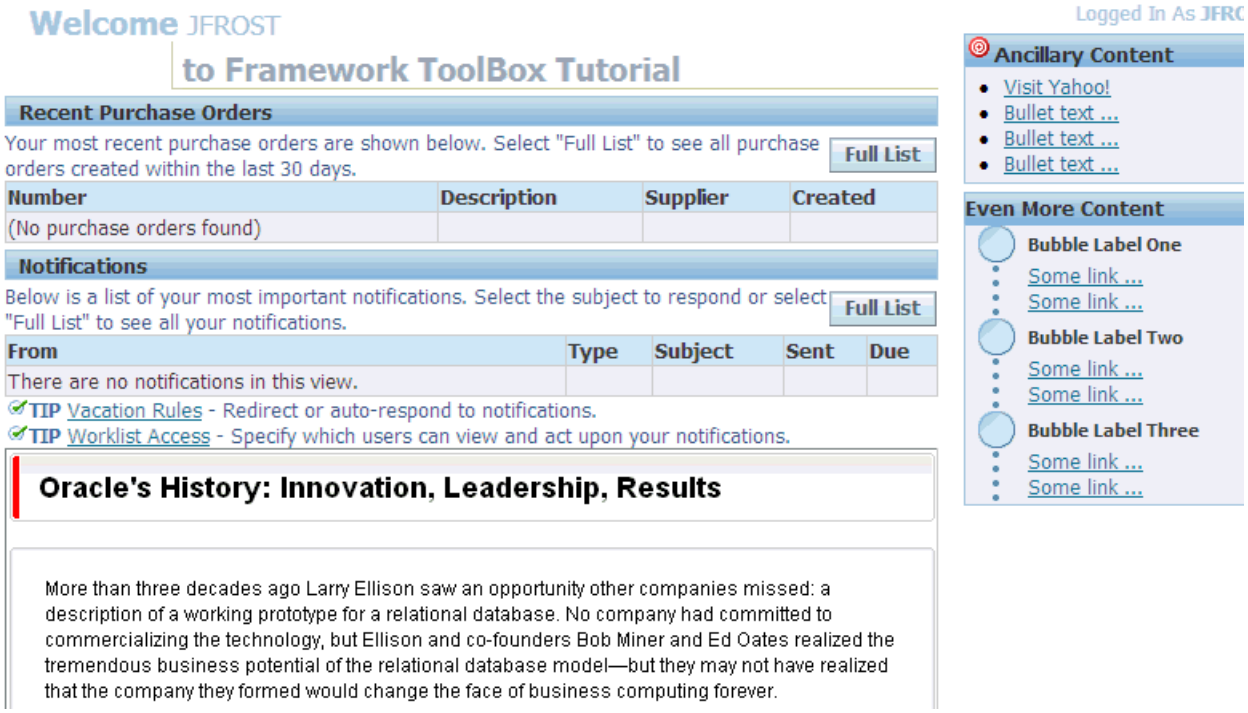
Overview

You can use the URL Include or Servlet Include web beans to include HTML content loaded from an external source under a parent region. This allows you to easily modify the HTML content without disturbing your E-Business Suite application page.

- URL Include - use this item style to display a block of very simple HTML content on an existing page. For example, you may use this item style to display recent announcements on a home page.
- Servlet Include - use this item style to display a block of HTML content loaded from a local Servlet or JSP.

URL Include

Figure 1: Example of a "Home" page including a simple HTML announcement.



Declarative Implementation

Step 1: Create a new Item in the region where you want to include content.

Step 2: In the **Property Inspector**, set the `Item Style` property to **urlInclude**.

Step 3: Enter a value for the `Source URI` property to indicate the source of the HTML content. Note the following:

- The value should be an absolute URL such as, `http://www.example.com/products/product_list.html`, and not a relative URL, such as `/products/product_list.html`. Relative links behave as relative to the surrounding page and not relative to the source of the HTML content.
- No post-processing on the HTML is performed, so the included HTML should not include HTML tags like `<html>` or `<body>`.
- Do not use forms or input fields in your HTML content, because anything submitted in the form (POST) is sent in the context of the surrounding page, not the source of the HTML content.

Note: If the included HTML content contains a link that navigates out of that content, you should include a target="_blank" designation so that the link opens in a new browser window instead of in the current browser window, unless that is what you desire. For example, the HTML link would look something like this:

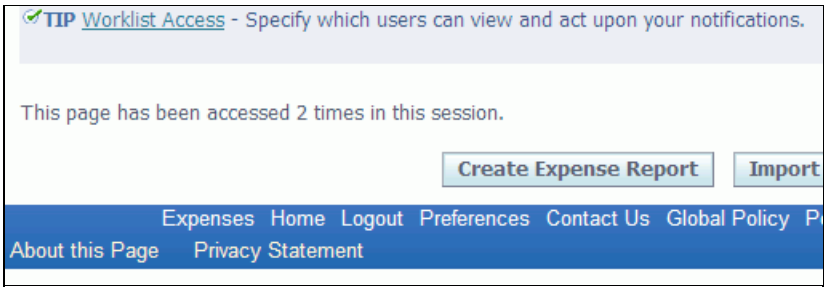
```
<a href="http://www.example.com/products/product_list.html"
  target="_blank">Oracle Products</a>
```

Runtime Control

No programmatic steps are required to implement the URL Include web bean.

Servlet Include

Figure 2: Example of a dynamic page displaying the number of times accessed.



Declarative Implementation

Step 1: Create a new Item in the region that you want to include content.

Step 2: In the **Property Inspector** and set the `Item Style` property to **servletInclude**.

Step 3: Enter a value for the `Source URI` property to indicate the source of the local Servlet or JSP. Note the following restrictions:

- The value should be an relative URL to the JSP or Servlet.
- Query parameters may be appended to the relative URL. These parameters are visible to the included servlet or JSP in its ServletRequest.

Runtime Control

No programmatic steps are required to implement the Servlet Include web bean.

Personalization Considerations

- None.

Known Issues

- None.

Infotile

Overview

Beginning in Release 12.2.5, you can use the new infotile component to present a single page where user-selected tile content may be displayed at any time. The infotile component contains one or more tile-based headers and corresponding detailed regions for content. The implementation of an infotile component appears as an enumeration of tiles, where each tile provides a brief description of its detailed region. When a user selects a given tile, the detailed content pertaining to that tile loads and displays in the infotile content region. You can set a default tile to display when the page initially loads.

The infotile component supports two modes of orientation: horizontal and vertical. The horizontal mode displays horizontally aligned tiles above the infotile content region, while the vertical mode displays vertically aligned tiles to the left of the infotile content region. Infotile also supports two tile sizes: Small and Regular.

For both small and regular tiles, each tile has a fixed dimension. If the title exceeds the width of the tile, the title will be truncated followed by an ellipses (...). In the figures below, the title appears as "Rejected" or "Approved" in each of the tiles. If the number of tiles exceeds the real estate of the viewport, overflow indicators (arrow head) appear, allowing users to navigate to the remaining tiles.

Note: Since the infotile feature relies on Partial Page Refresh (PPR) to load the infotile content region, you must enable PPR.

Note: If you are using the Microsoft Internet Explorer browser, the infotile feature requires IE10 and above.

Figure 1: Regular, horizontal infotiles.

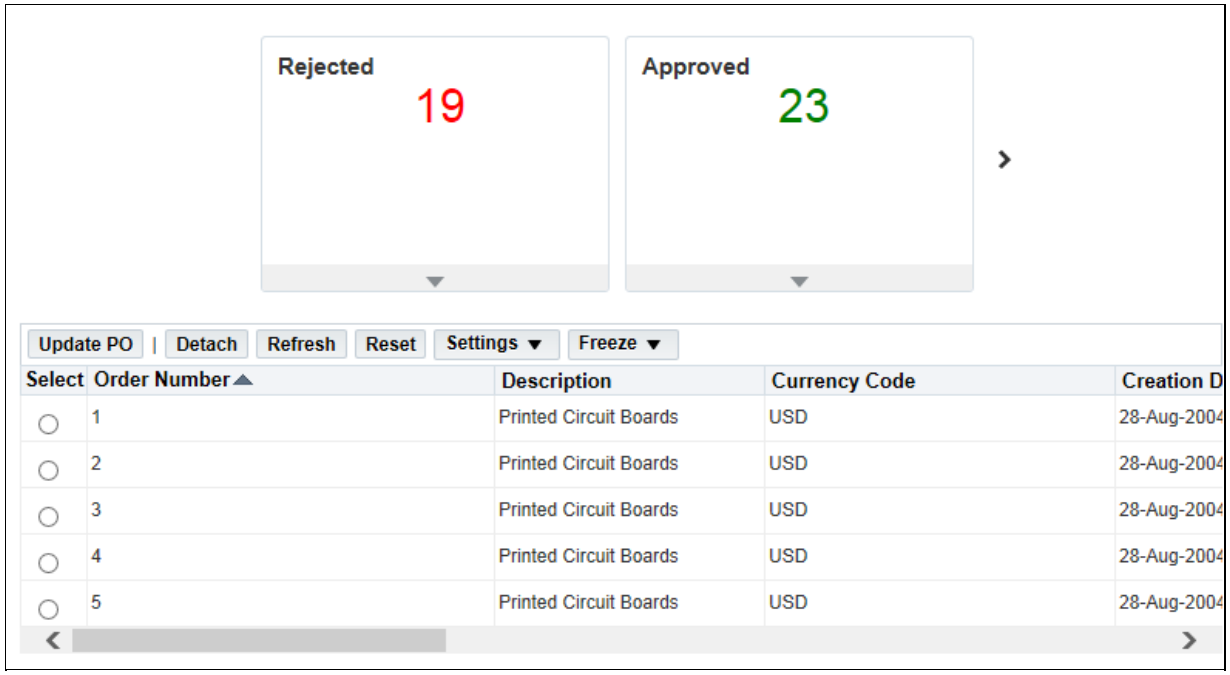


Figure 2: Regular, vertical infotiles.

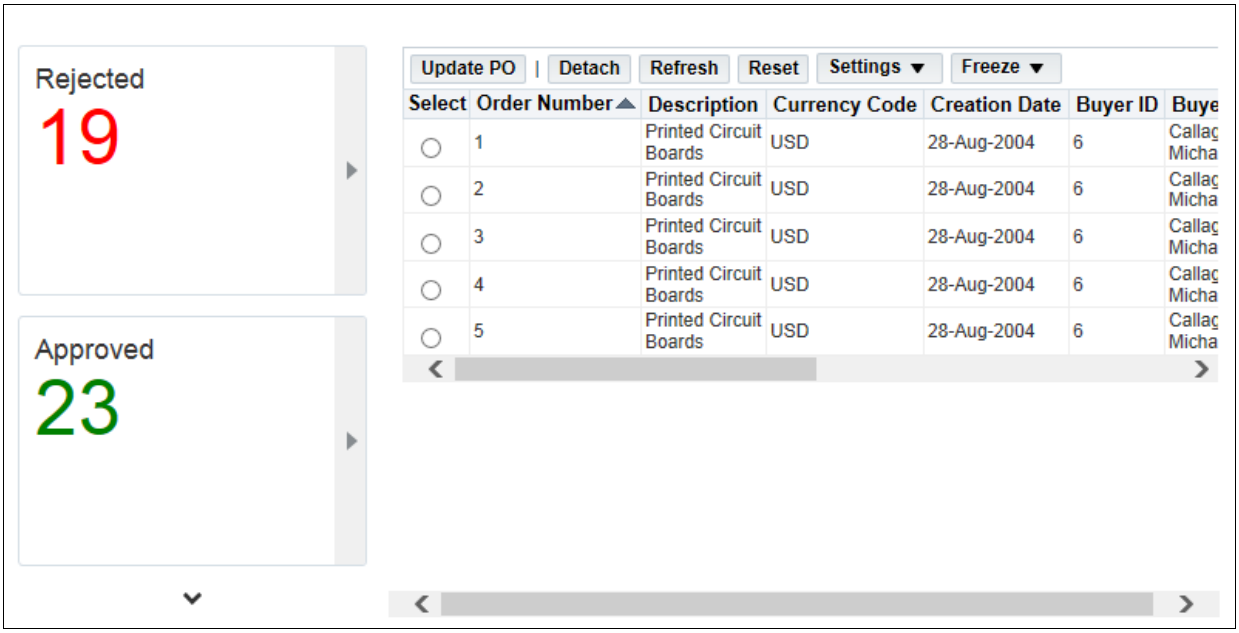


Figure 3: Small, horizontal infotiles.

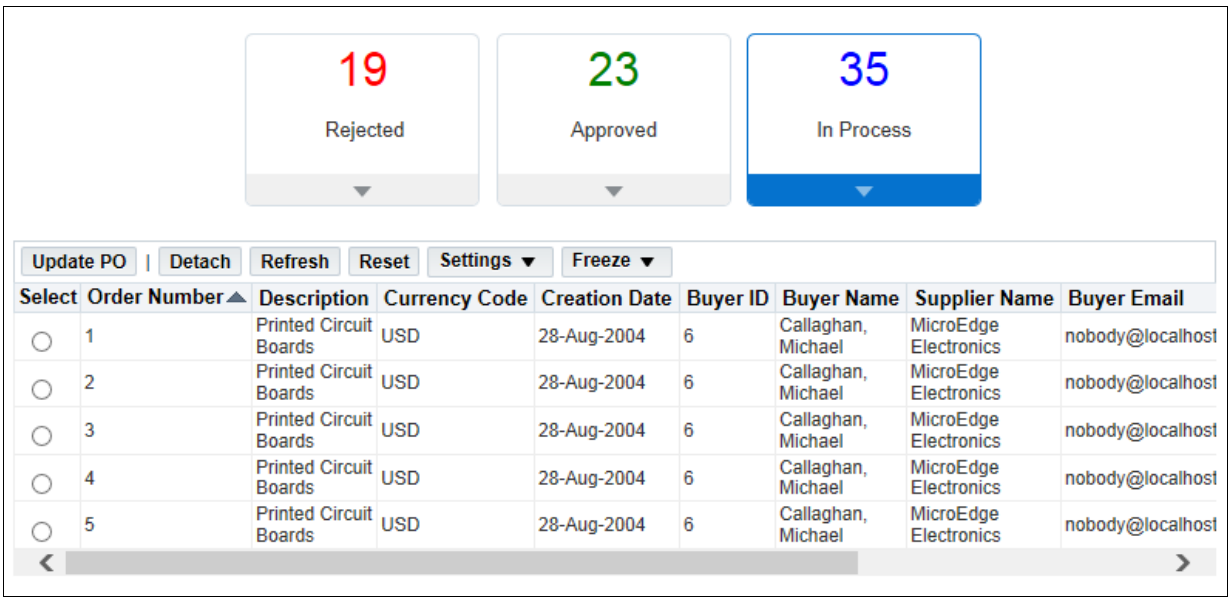
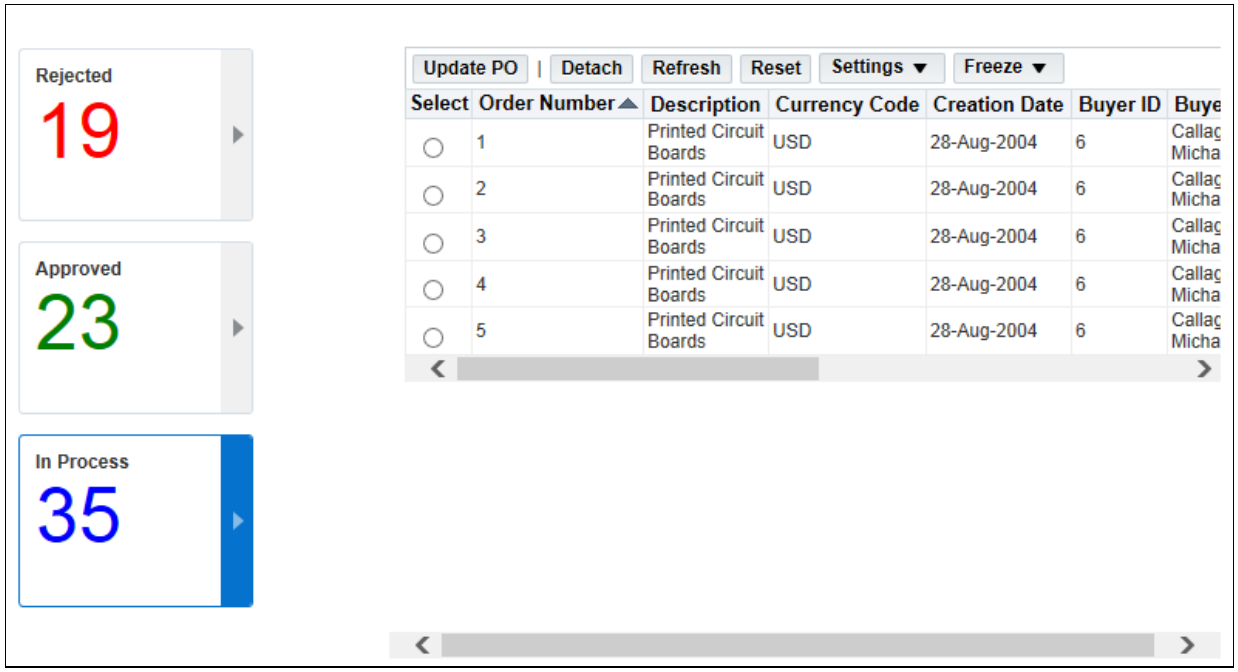


Figure 4: Small, vertical infotiles.



Contents

- Declarative Implementation
- Runtime Control
- Accessibility Support
- Personalization Considerations
- Known Issues
- Related Information

Declarative Implementation

Following is a brief outline of the steps to create an infotile region.

Step 1: Create a page with a pageLayout region using OA Extension. Make sure the `Form` property on the pageLayout region is set to **True**.

Step 2: Select the pageLayout region in the Structure pane, and choose **New** > **Region** from the context menu. Set the following properties on this new region (Required properties are marked with *):

- *`ID` - set the infotile's ID property in accordance with the OA Framework File / Package/ Directory Standards.
- *`Region Style` - set the region style to **infotile**.
- `*Orientation` - set this property to **horizontal** or **vertical** as desired.

Important: You should create only one infotile region per page.

Step 3: Create the container for the content as an indexed child of the **infotile** region. Select the **infotile** region and choose **New** > **Region** from the context menu. Set the `ID` on this new region in accordance with the OA Framework File / Package/ Directory Standards. This `ID` will later be used to map this container region to a tile. Set the `Region Style` property to a desired container region style (for example, stackLayout).

Step 4: Select the container region and choose **New** > **Region** from the context menu to add the desired contents under this container region. When you later map a tile to this container region, all the content added under this container will display when a user selects the tile.

Important: Do not add the infotile contents directly under the **infotile** region, always add the contents to a container region directly below the **infotile** region.

Step 5: Select the **infotile** region and choose **New** > **tiles** from the context menu. OA Extension automatically creates a **tileHeader** region named child under the **infotile** region. The **tileHeader** region is the container to which you will add more tiles.

Step 6: You can set a default tile to display when the page loads. Select the **tileHeader** region. Set the `Default Tile Id` property to the ID of the content container you wish to display by default. If the `Default Tile Id` property is not set, the first added tile becomes the default tile.

Step 7: For the **tileHeader** region, set the `Tile Size` property to **Small** or **Regular**.

Step 8: Select the **tileHeader** region and and choose **New** > **tile** from the context menu to create a new **tile** region indexed child under the **tileHeader** region.The **tile** region refers to the area that holds brief information about its content.

Important: You should create at least one tile region for the infotile component.

Step 9: Use the `contentRegionID` property to map the **tile** region to a specific content region. Specify the **ID** of a container region that you previously added as an indexed child of the **infotile** region. When a user selects the rendered tile, the content region corresponding to the tile displays.

Step 10: Optionally set the `Title` property on the **tile** region.

Step 11: Optionally create brief content for the **tile** region by selecting the **tile** region and and choosing **New** > **region** or **item** from the context menu and style the tile content as desired.

Runtime Control

There are no programmatic steps necessary to create an infotile component. Adding an infotile region to a page instantiates the `OAInfotileBean`. A tile is an object of `OATileBean` and an indexed child of `OATileHeaderBean`. `OATileHeaderBean` is added as a named child under the infotile region.

When a user selects a tile, the entire infotile region refreshes using partial page refresh and responds with content corresponding to the selected tile. The controller `processRequest ()` method for all the tiles and content areas is called during the page load. Subsequent tile selections by the user then trigger PPR events during which the `processFormRequest ()` method is called for all the tiles and for the current content container and its content. The remaining content containers are not processed during the PPR cycle.

You may use the `oracle.apps.fnd.framework.webui.beans.layout.OATileHeaderBean` method `setInitialTileId(Object tileID)` to set the default tile when the infotile component first loads.

The following example sets a tile with a tile ID of "tile4" as the default tile when the infotile component loads:

```
  OATileHeaderBean tileList=(OATileHeaderBean)(((OAInfotileBean)webBean).getTileHeader());
  tileList.setInitialTileId("tile4");
```

The `setInitialTileId(Object tileID)` API, which supports data binding, actually sets the attribute `INITIAL_TILE_ID`. Suppose a user drills down from an infotile content region of a given tile and then navigates back to that infotile content page after performing some action. If you retain the tile selection from where the user drill-down navigation began as a view attribute, you can then fetch the view attribute to set the `INITIAL_TILE_ID` when the user navigates back, as shown in the following code example:

```
OADataBoundValueViewObject  dataBound = new OADataBoundValueViewObject(webBean, viewAttrbuteName);
 tileList.setInitialTileId(dataBound);
 // this sets the required attribute to display selected tile based on the
 // view attribute value when navigating back
```

In the example above, `tileList` is `OATileHeaderBean`, while `viewAttributeName` is the view attribute where the current **selected tile Id** is saved while redirecting to the drill down page.

You may also identify which tile is currently selected by adding following code to your controller's `ProcessFormRequest()` method:

```
String selectedTileId=null;
   String event = pageContext.getParameter(EVENT_PARAM);
   if (event.equals(TILE_SELECT_EVENT))
{
   selectedTileId = pageContext.getParameter(SOURCE_PARAM);
}
```

Small Infotile Display

The small infotile is optimized to display only a numeric value, with an optional image adjacent to the number. While you may control the small infotile display from your controller code by adding font-color to the number, you must always set the font-size to be a constant of 30 pixels. By adding the following inline styles to your controller code, you ensure that the number properly fits the small tile space allocated to it:

```
color: purple; // Desired color for the metric
font-size: 30px; // Standard size for the metric
```

However, if you use a layout that appends additional padding or margin to the number, the metric will occupy more space than is available and be partially hidden towards the bottom. If this happens, inspect the number using a developer tool in your browser. If any padding or margin is applied to the number, then nullify its effect by adding the following "margin:0px" inline style on the metric in your controller code:

```
color: purple; // Desired color for metric
font-size: 30px; // Standard size for metric
margin:0px // Nullify effect of margin introduced by layout
```

Note that the above anomaly may be avoided by adding a formattedText item under the tile to display the tile metric and binding the formattedText item to a desired view attribute.This ensures a consistent UI after adding the following inline-styles to your controller code:

```
color: purple;  // Desired color for metric
font-size: 30px; // Standard size for metric
```

Accessibility Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still use the infotile component as follows. In general, keyboard interaction for Infotile is similar to subtabs. The keyboard interactions common to both vertical and horizontal display mode of infotile are:

- **[Tab]** - moves focus to next page element. Note that only the active tile is in the tab order. If the current focus is on the active tile itself, pressing **[Tab]** shifts focus to the first "tab-able" element in the active tile's content region.
- **[Home]** - with focus on active tile, pressing **[Home]** shifts focus to the first tile of the tile list and activates that tile.
- **[End]** - with focus on active tile, pressing **[End]** shifts focus to last tile of the tile list and activates that tile.
- **[Control]** + **[Page Up]** - with focus anywhere within the infotile content region, pressing **[Control]** + **[Page Up]** shifts focus from the current tile to the previous tile and activates it. If the currently selected tile is the first tile, then this action shifts the focus to the last tile and activates it.
- **[Control]** + **[Page Down]** - with focus anywhere within the infotile content region, pressing **[Control]** + **[Page Down]** shifts focus from the current tile to the next tile and activates it. If the currently selected tile is the last tile then this action shifts the focus to the first tile and activates it.

Keyboard interactions for **horizontal** display mode:

- **[Left/Right Arrow]** - with focus on the active tile, pressing **[Left/Right Arrow]** shifts focus to the previous/next (next/previous in RTL session) tile in the tile list and activates it. If the currently selected tile is the last tile, pressing **[Right Arrow]** (**[Left Arrow]** for a RTL session) shifts focus to the first tile. If the currently selected tile is the first tile, pressing **[Left Arrow]** (**[Right Arrow]** for a RTL session) shifts focus to the last tile.
- **[Control]** + **[Up Arrow]** - with focus anywhere within the infotile content region, pressing **[Control]** + **[Up Arrow]** shifts focus to the currently selected tile.

Keyboard interactions for **vertical** display mode:

- **[Up/Down Arrow]** - with focus on the active tile, pressing **[Up/Down Arrow]** shifts focus to the previous/next tile in tile list and activates it. If the currently selected tile is the last tile, pressing **[Down Arrow]** shifts focus to the first tile. If the currently selected tile is the first tile, pressing **[Up Arrow]** shift focus to the last tile.
- **[Control]** + **[Left Arrow]** (**[Right Arrow]** in a RTL session) - with focus anywhere within the infotile content region, pressing **[Control]** + **[Left Arrow]** (**[Right Arrow]** in a RTL session) shifts focus to the currently selected tile.

Personalization Considerations

See a summary of Infotile personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- Infotiles are not supported in JTT pages.
- Infotiles are only supported in the Release 12.2.5 Look and Feel.

Related Information

- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAInfotileBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OATileHeaderBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OATileBean`

Inline Messaging, Tips, Hints and Bubble Text

Overview

Per the Oracle Browser Look-and-Feel (BLAF) UI Guideline Inline Messaging, Tips, Hints and Bubble Text, you can add supplemental helpful text to your pages to help the user complete the task at hand. Within this general category of help methods (see the UI Guideline for information on how to choose the appropriate help method(s) for your design), this document describes how to add field-level hints, tips and bubble text. Figure 1 shows instruction text set for the page and region-levels, field-level hints and a tip. Figure 2 shows bubble text examples.

- See the Instruction Text document for information on adding primary task help text at the page and region levels.
- See the Buttons (Global) document for information on adding page-level context sensitive help to the Help global button.

Figure 1: Example of page and region-level instruction text, field-level hints and a tip.

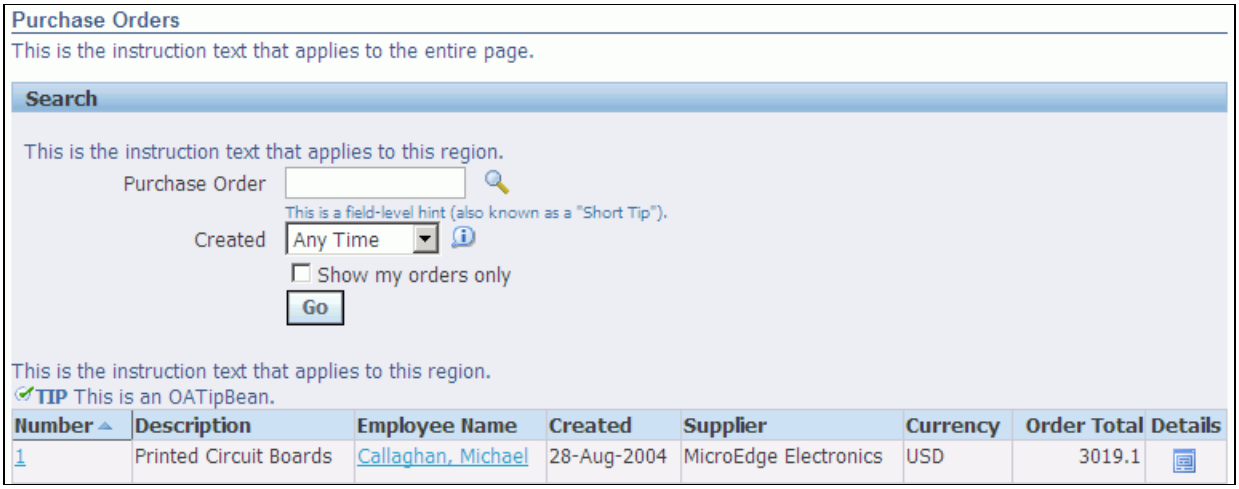
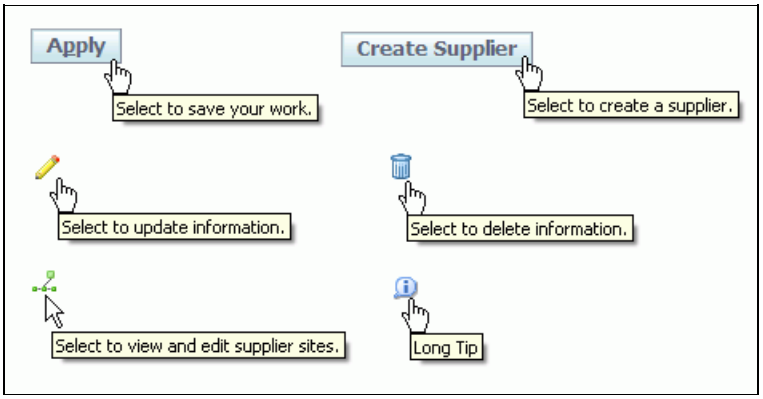


Figure 2: Bubble text examples.



Inline Messages / Field-Level Hints

You can declaratively create either short or long field-level hints:

- The short hints render immediately below the item as shown in Figure 1 (see the Purchase Order field).
- The long hints render as a selectable information icon next to the item (see the Created poplist in Figure 1). When the user selects the information icon, a dialog window opens as shown in Figure 3 below.

Additional usage information and restrictions:

- In Screen Reader Optimized mode, tip text is displayed on the left side of the input field, depending on language session being used. In None or Standard mode, tip text is displayed below the input field.
- You cannot configure a static error message as shown in the UI Guidelines (this is displayed automatically for you when field-level validation fails; see Error Handling for additional information about this).
- You cannot associate a warning icon with an item as shown in the UI Guidelines.

Field-Level Hints

To configure a field-level hint for any field *that is not displaying a date* (see the date format instructions below for this case):

Step 1: In the JDeveloper structure pane, select the item with which you want to associate a hint.

Note: Field-level hints can be configured only for those items whose name begins with **message** (for example, a **messageChoice** or a **messageTextInput**).

Step 2: Set the `Tip Type` property to **shortTip**.

Step 3: Define a message in the Applications Message Dictionary for the text you want to display. Set the `Tip Message Appl Short Name` and the `Tip Message Name` properties accordingly.

The OA Framework displays your text in the current session language immediately beneath the related item.

To configure the standard date format example inside a date field as Placeholder text:

Step 1: In the JDeveloper structure pane, select the date item that needs a format example.

Step 2: Set the `Tip Type` property to **dateFormat**.

The OA Framework automatically renders a standard date format example using the user's date format preference.

Long Hints (with Information Icon)

To add a long hint to an item:

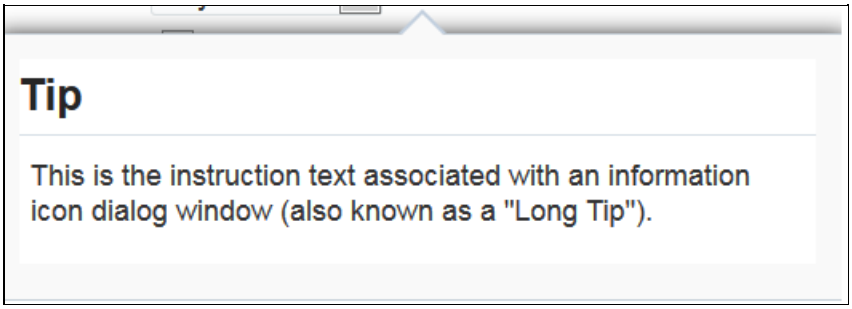
Step 1: In the JDeveloper structure pane, select the item with which you want to associate a hint.

Step 2: Set the `Tip Type` property to **longMessage**.

Step 3: Configure the long message content:

- (Option 1) Define a message in the Applications Message Dictionary for the text you want to display. Set the `Tip Message Appl Short Name` and the `Tip Message Name` properties accordingly, and the OA Framework displays your message in the dialog shown below when the user selects the item's information icon.
- (Option 2) If you need complete control of the contents in the dialog window, do not specify the `Tip Message Appl Short Name` and `Tip Message Name` properties. Instead, design your own region and specify its fully qualified name for the `Long Tip Region` property (for example, `/oracle/apps/fnd/framework/toolbox/webui/PurchaseOrderLongTipRN`). At runtime, the OA Framework displays your region in the dialog.

Figure 3: Popup window that appears when the information icon is selected.



Tips

Although the field-level hint properties in JDeveloper are called "tips," a tip in BLAF UI Guidelines parlance is a special component with an icon, a standard "TIP" prefix and the tip text that you define (an example is shown immediately above the search results table in Figure 1). When you define a tip for your page, the OA Framework instantiates an `oracle.apps.fnd.framework.webui.beans.OATipBean` and renders the tip text in the current session language.

Tip: If you want to display tip text that includes HTML tags, see the Custom HTML document.

Declarative Implementation

To add a tip to your document:

Step 1: In the JDeveloper structure pane, select the region where you want to add a tip, right-click and select **New > Item**. Note that, unlike field-level hints which are associated with directly with existing items, tips are like any other item; you can simply add them wherever you need them.

Step 2: Specify the item `ID` in accordance with the OA Framework Naming Standards and set the `Item Style` to **tip**.

Step 3: If your tip text is very brief, you may enter it directly in the `Text` property. Otherwise, define a message in the Applications Message Dictionary, and then set the `Tip Message Appl Short Name` and the `Tip Message Name` accordingly.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

If you need to create a tip programmatically, follow these steps:

Step 1: Create a message in the Applications Message Dictionary.

Step 2: Instantiate the tip as shown below. Then, instantiate an `oracle.apps.fnd.framework.webui.beans.OAStaticStyledTextBean` to hold your tip text and add it as an indexed child of the tip. Note that UIX automatically sets the CSS style to **OraTipText** on your behalf.

```
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.OAStaticStyledTextBean;
import oracle.apps.fnd.framework.webui.beans.OATipBean;
...
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
 // Always call this first.
 super.processRequest(pageContext, webBean);

 // Instantiate the tip bean using the factory mechanism (do not use "new").
 OATipBean tip = (OATipBean)createWebBean(pageContext,
                                          OAWebBeanConstants.TIP_BEAN,
                                          null,"aName");

 // Instantiate the text portion of the tip.
 OAStaticStyledTextBean tipText = (OAStaticStyledTextBean)createWebBean(pageContext,
                                          OAWebBeanConstants.STATIC_STYLED_TEXT_BEAN,
                                          null,"anotherName");

 // Obtain the translated text value from the Applications Message Dictionary
 // and set it as the text value in the static styled text bean.
 String tipTextValue = pageContext.getMessage("AK", "FWK_TBX_T_TIP_BEAN", null);
 tipText.setText(tipTextValue);

 // Add the tip text to the tip bean.
 tip.addIndexedChildren(tipText);

}
```

Bubble Text

Bubble text, otherwise known as "ALT" or "rollover" text, should be added to buttons and images as described in the OA Framework View Coding Standards Accessibility section. See this document for instructions on when and how to specify the ALT text. See the UI Guideline for information on appropriate verbiage.

Related Information

- BLAF UI Guidelines
  - Inline Messaging, Tips, Hints and Bubble Text
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.OATipBean`
  - `oracle.apps.fnd.framework.webui.beans.OAStaticStyledTextBean`
- Developer's Guide
  - Instruction Text
  - Buttons (Global)
  - Custom HTML

    - OA Framework View Coding Standards
    - OA Framework File Standards
    - Error Handling
- OA Framework ToolBox Tutorial / Sample Library
    - `oracle.apps.fnd.framework.toolbox.tutorial.webui.PoSearchPG.xml`

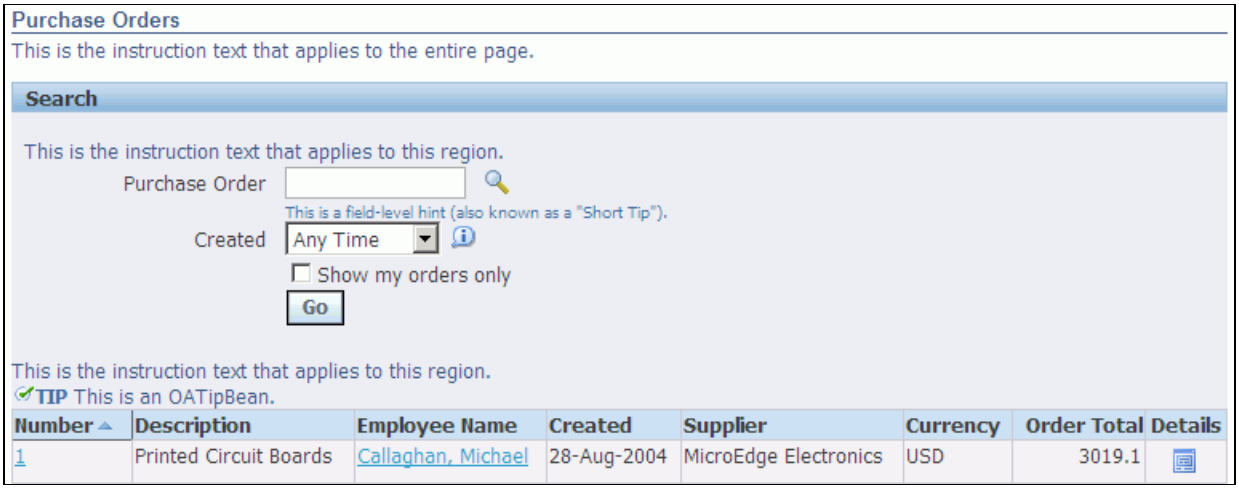
Instruction Text

Overview

Per the Oracle Browser Look-and-Feel (BLAF) UI Guideline Instruction Text, instruction text is the primary method for directing and helping users to perform specific tasks. Instruction text can be specified in relation to the following as illustrated in Figure 1 below.

- the entire page
- a section of content (a subheader or a subsubheader)
- a table
- a group of components within a section of content (rare, not illustrated in Figure 1)

Figure 1: Example of instruction text in different parts of the page.



See Inline Messages, Tips, Hints and Bubble Text for information on creating field-level hints and tips as shown above the table in Figure 1.

Declarative Implementation

To add plain instruction text (without any links or HTML formatting) in any of the valid page areas, following these steps and the OA Framework will create an `oracle.apps.fnd.framework.webui.beans.OAStaticStyledTextBean`.

Tip: If you want to display tip text that includes HTML tags, see the Custom HTML document.

Step 1: Create a message in the Applications Message Dictionary.

Step 2: Create a region item and set its `Style` to **staticStyledText**.

Step 3: Set the region item's `ID` property in accordance the OA Framework File Standards.

Step 4: Set the `CSS Class` property to **OraInstructionText**.

Step 5: Associate the message you created in Step 1 with the region item you created in Step 2 by setting its `Message Name` and `Message Appl Short Name` properties as appropriate for your message.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or extended easily.

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

If you need to create a tip programmatically, follow these steps:

Step 1: Create a message in the Applications Message Dictionary.

Step 2: Instantiate the static styled text and configure its key properties.

```
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.OAStaticStyledTextBean;
```

```
...
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
 // Always call this first.
 super.processRequest(pageContext, webBean);

 // Instantiate the instruction text bean using the createWebBean() factory.
 OAStaticStyledTextBean helpText = (OAStaticStyledTextBean)createWebBean(pageContext,
                                        OAWebBeanConstants.STATIC_STYLED_TEXT_BEAN,
                                        null,"aName");

 // Obtain the translated message from Message Dictionary and set it
 // as the bean's value.
 String helpTextValue = pageContext.getMessage("AK", "FWK_TBX_T_REGION_GENERAL", null);
 helpText.setText(helpTextValue);

 // Set the CSS Style to OraInstructionText.
 helpText.setCSSClass("OraInstructionText");

}
```

Personalization Considerations

- None

Known Issues

- None

Related Information

- BLAF UI Guidelines:
  - Instruction Text
- Developer's Guide
  - Custom HTML
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.OAStaticStyledTextBean`

Keyboard Shortcuts

Keyboard shortcuts provide users with an alternative to pointing devices for navigating within a page. OA Framework supports the use of accelerator keys and the `Enter` key, in certain contexts, as keyboard shortcuts.

Contents

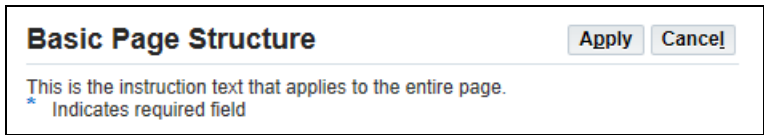
- Accelerator Keys
- Enter Key

Accelerator Keys

As described in the Oracle Browser Look and Feel (BLAF) UI Guideline: Keyboard Shortcuts, OA Framework pages provide support for two kinds of "hot keys" for quickly performing selected actions/navigation using the keyboard:

- Mnemonic (Common) Accelerator Keys - centrally defined mnemonic letter and symbol assignments for commonly accessed buttons as shown in Figure 1 below.
- Numeric (Application-Specific) Accelerator Keys - numeric assignments which may be selectively assigned to common actions within a given application page.

Figure 1: Accelerator Keys in Common Cancel and Apply buttons.



In Windows Internet Explorer, users exercise the accelerator keys by selecting **`Alt + <char>`** from the keyboard, while in Firefox, users exercise the accelerator keys by selecting **`Shift + Alt + <char>`**.

- In Windows Internet Explorer, for buttons, the accelerator key "activates" the widget. For example, typing **`Alt + p`** on the page shown in Figure 1 selects the **Apply** button and submits the form. In Firefox, for buttons, the accelerator key just puts focus on the button, but does not activate it, that is, it does not submit the form.
- For message beans, the accelerator key sets the focus to that bean. For example, if you designate "2" as the numeric accelerator key for a `messageTextInput`, then typing **`Alt + 2`** moves your cursor to that text field.
- Accelerator keys are automatically provided for subtabs. At runtime, users can quickly move focus to the next or previous subtab by selecting **`Alt + >`** and **`Alt + <`** from the keyboard. Selecting the `Enter` key displays the subtab contents.

Attention: In Chrome, if an accelerator key is defined within an inline frame (iframe), but the current focus is outside of the iframe, the accelerator key does not work.

Mnemonic (Common) Accelerator Keys

Declarative Implementation

The centralized list of mnemonic accelerator keys is maintained in the Keyboard Shortcuts UI guideline, and implemented by the OA Framework team in the standard attribute sets contained in the package `/oracle/apps/fnd/attributesets/Buttons`. To ensure that any instances of buttons that you have on this list inherit the correct accelerator key value, simply assign the button the appropriate attribute set in this package.

The following process is the easiest way to ensure that you specify the correct attribute set in JDeveloper:

1. Select your button to access the **Property Inspector** and select the `Attribute Set` property's LOV icon.
2. In the **Attribute Set** window, enter the button label that you're seeking in the **Attribute Set** field (for example, Go, Apply, Cancel and so on).
3. Opt to perform a search in the **Entire MDS XML Path**.
4. Select the **Search** button to find the matching attribute set in `/oracle/apps/fnd/attributesets/Buttons`. For example, Searching for "Apply" should return the attribute set `/oracle/apps/fnd/attributesets/Buttons/Apply`.

Tip: A Perl upgrade script is provided to help you quickly set the correct attribute sets for preexisting code (if needed).

Runtime Control

If you need to programmatically set the prompt for a common button (there should be little or no reason to do this), you may call the `setText(String)` method with an ampersand character (**&**) immediately before the character to serve as an accelerator key. For

example, UIX will assign"r" as the accelerator key for the String value **Sea&rch**. Note that you must also explicitly set the short description to an appropriate value by calling the `setShortDesc(String)`. For example, if the prompt is set to "Sea&rch", then the short description should be set to "Search" or something similar.

Tip: The character **&** itself can be included in the prompt by using the value **&&**.

Any values that you set when calling `setText` or `setShortDesc` must be translated (you cannot use untranslated, static Strings). Although you could obtain the values from message dictionary for this, a better approach would be to use the values set in the corresponding attribute sets. For example, you could reuse the prompt definition from the `/oracle/apps/fnd/attributesets/Buttons/Apply` attribute set. See Programmatic Access to Attribute Sets in Implementing the View.

Numeric (Application-Specific) Accelerator Keys

You may -- in exceptional cases where rapid navigation/action execution is essential for frequent users -- assign numeric accelerator keys for selected items in a page.

At runtime, numeric access keys appear underlined if present in the component's prompt. If not, UIX appends the underlined access key to the end of the prompt and encloses it in parentheses. For example: **Some Button (9)**.

Declarative Implementation

Step 1: Create an item with one of the following styles:

- button
- submitButton
- any `message*` style

Step 2: Set the `Access Key` property to a value of 0 - 9 by selecting it from the property's poplist.

Note: Limit your access key implementation to product-specific buttons where execution speed is absolutely essential for high-frequency users. So, for example, you don't need product-specific button access keys in a self-service application that is likely to be used once a year.

Runtime Control

If you need to set the access key programmatically, call `setAccessKey(char)` on any of the supported beans.

Warning: You **MUST** use a value of 0 - 9. Do not use letters as these are reserved for use by UIX for the common accelerator keys.

Enter Key

As described in the Oracle Browser Look and Feel (BLAF) UI Guideline: Keyboard Shortcuts, the `Enter` key is used in the following ways:

- When the cursor is in a field, the `Enter` key triggers the action/navigation button associated with the field, such as **Go** and **Login**. This behavior requires programming, and is supported for two types of fields:
    - **Login fields** - If the cursor is in a login field, such as **User Name** or **Password**, the `Enter` key triggers the action of the **Login** button.
    - **Search fields** - (for Simple Search only) If the cursor is in a **Search** field the `Enter` key triggers the search. If the page contains other updateable fields, the application must validate the data before the search is performed.
- On a LOV secondary window (not Look Ahead LOV window), the `Enter` key triggers the **Select** button.
- When the focus is on a link, an image, or a button with an associated URL, the `Enter` key triggers the associated action. This is default browser behavior.
- In Release 12.0 and above, when the cursor is in a text input field and the page contains at least one Submit button, pressing the `Enter` key submits the form that encloses the `Enter` key. This is standard browser behavior. Since an OA Framework page uses a single HTML form, pressing the `Enter` key triggers the submission of the entire page, and for this to happen, the HTML form must contain a submit control such as a Submit button on the page.

    Note: The form submit event does not require any submit control to fire, that is, the event is not associated with any Submit button on the page. As a result, if you do not write your server-side code handling events properly (for example, it does not explicitly check for event parameters before executing) the application may behave in an unpredictable manner when a user presses the `Enter` key.

Personalization Considerations

- None

Known Issues

- None

Related Information

- BLAF UI Guideline(s)
  - Keyboard Shortcut

Layered Layout

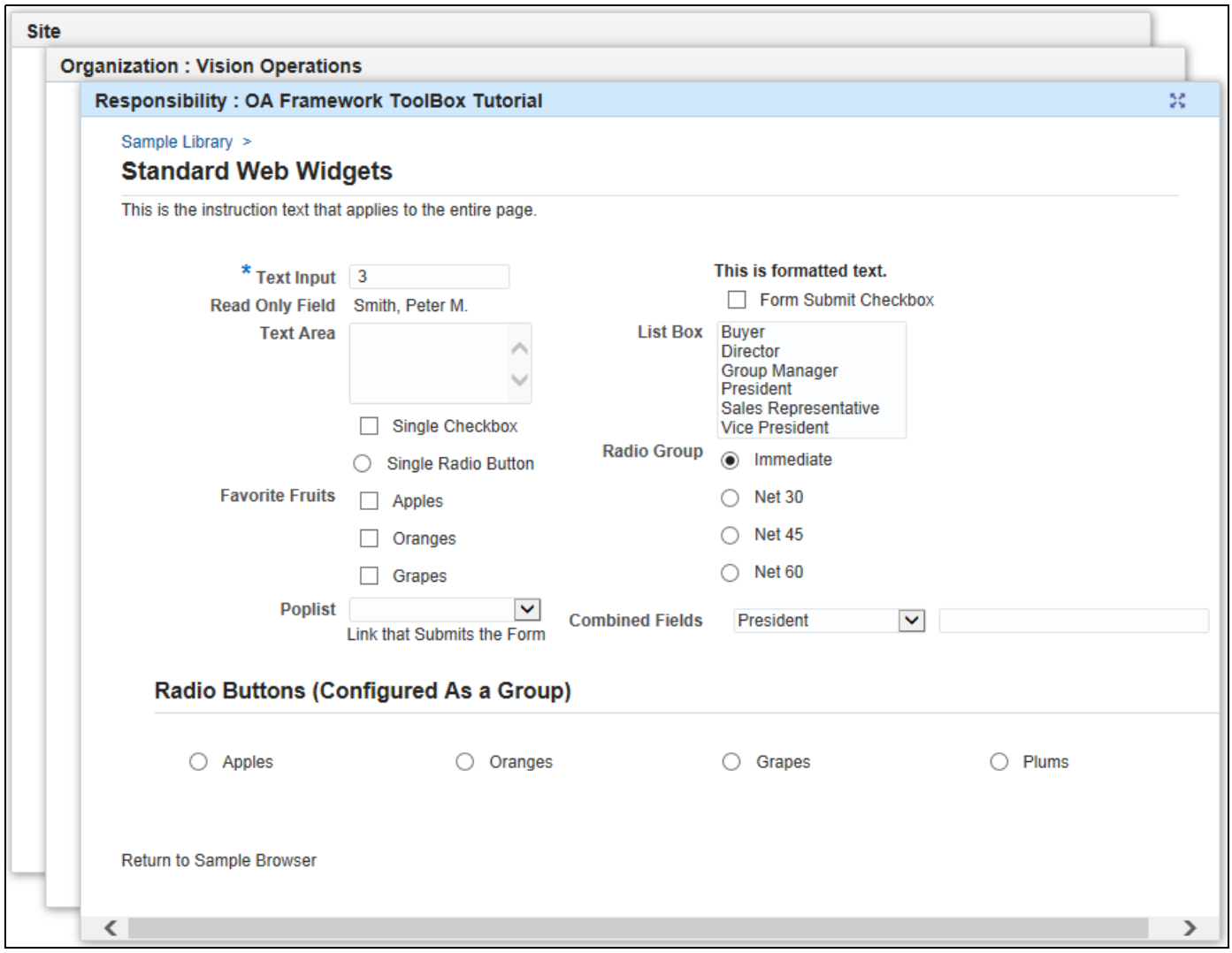
Overview

From Release 12.2.6, you can create a component that allows you to display content in a *layered layout* rather than a subtab layout.

Each layer has a title bar area users can select to switch between layers. Users may only select one layer at a time, and only upon selection will the layer's content render. The layers themselves are not movable, however, and users may select the maximize icon to increase the size of a selected layer or the restore icon to revert the layer back to its original size. As a developer, you can also add icons to the title bar of a layer to perform actions as needed.

Unlike a subtab layout, a layered layout does not use padding. To add spacing to the components in a layered layout, developers need to explicitly add paddiing/margin to the inner component, such as has been done in the personalization page.

Figure 1: Example of the layered layout component as implemented in Personalization Workbench



Contents

- Declarative Implementation
- Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

Declarative Implementation

You may use the Oracle JDeveloper OA Extension Property Inspector to add a layered layout component to a region, such as to a

header region.

Step 1: In your pageLayout region, select the region to which you want to add a layered layout component. Select **New** > **Region** from the context menu.

Step 2: Set the `Region Style` property for this new region to **layeredLayout**.

Step 3: Use the Property Inspector to set the properties on the **layeredLayout** region, specifically (where * indicates a required property):

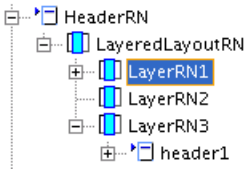
- `ID`* - Specify an identifier that uniquely identifies the region in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Selected Layer ID` - ID of the initially selected layer. This ID must match the ID of one of the layers contained in the layered layout region that you create in Step 5. If this property is not set, the last rendered layer is selected by default.
- `Layer Selection Event` - specify the event you wish to trigger when a layer is selected. Choose between submitAndRedirect or submit. The default is submitAndRedirect.
- `Controller Class` - specify the fully-qualified class name of the controller object.
- `Rendered` - specify whether to render the layered layout component. The default is True.

Step 4: Select the layered layout region, then select **New** > **layer** from the context menu.

Step 5: Use the Property Inspector to set the properties on the **layer** region, specifically (where * indicates a required property):

- `ID`* - Specify an identifier that uniquely identifies the region in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Text` - specify the text for the title bar of the layer.
- `Rendered` - specify whether to render the layer. The default is True.

Figure 2. Example of a layered layout region structure as displayed in Oracle JDeveloper.



Runtime Control

At runtime, OA Framework instantiates an

`oracle.apps.fnd.framework.webui.beans.layout.OALayeredLayoutBean.`

You may also programmatically create a layered layout web bean inside a region by including code similar to the example below in your controller's `processRequest` method. The code creates an instance of the OALayeredLayoutBean and adds OALayerBean objects as indexed children.

```
OALayeredLayoutBean layeredLayoutBean = (OALayeredLayoutBean)createWebBean(pageContext,
                                         OAWebBeanConstants.LAYERED_LAYOUT_BEAN,
                                         null, "LayeredLayoutRN");
for( int i = 0; i < 3; i++)
{
 OALayerBean layerBean =  (OALayerBean) createWebBean(pageContext,
                          OAWebBeanConstants.LAYER_BEAN, null,
                          "LayerRN" + i)
 layerBean.setText("Layer " + i);
 // text on title bar
 layeredLayoutBean.addIndexedChild(layerBean);
}
```

To explicitly select a specific layer, add code similar to the following:

```
layeredLayoutBean.setSelectedLayerId("layer2");
```

To add one or several icons to a layer, add code similar to the following:

```
OAPageButtonBarBean bar = new OAPageButtonBarBean();
OAImageBean i1 = new OAImageBean();
i1.setSource("/OA_MEDIA/undo_pr.png");
i1.setFireActionForSubmit("deletePersonalization",param,null,true);
i1.setCSSClass("DeleteIcon");
bar.addIndexedChild(i1);
```

```
layer.setHeaderButtons(bar);
```

To set the redirect mode or the form submission mode, add one of the following code examples:

```
layeredLayoutBean.setLayerSelectionEvent(LAYER_SELECTION_EVENT_SUBMIT_AND_REDIRECT);
// set to redirect mode, which is the default

layeredLayoutBean.setLayerSelectionEvent(LAYER_SELECTION_EVENT_SUBMIT);
// set to form submission mode
```

Redirect Mode

In redirect mode, the page is redirected back to the same page when switching layers. As a result, a full page refresh is performed on the client side. If layers are associated with controllers, `processRequest` is called only on the controller of the selected layer and as a result, data on unselected layers are not created or initialized. When switching layers, `processRequest` is called on the layer that is to be selected after redirection. In other words, if “layer1” is currently selected, and you select “layer2”, `processRequest` is call on “layer2” after the page redirects.

The methods `processFormData` and `processFormRequest` are only called on the *currently* selected layer. That means that if “layer1” is selected, and you then select “layer2”, `processFormData` and `processFormRequest` will be called on “layer1”.

Form Submission Mode

If you set the layered layout to form submission mode, PPR is used to switch layers and no redirection occurs.

Also, when the page initially loads, `processRequest` is called on all layers if they are associated with controllers.

Methods `processFormData` and `processFormRequest`are only called on the *currently* selected layer. That means that if “layer1” is selected, and you then select “layer2”, `processFormData` and `processFormRequest` will be called on “layer1”.

Personalization Considerations

- See a summary of Layered Layout personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- OA Component Reference
  - For a description of all the possible properties that you can set for the **layeredLayout** region type, see the OA Component Reference in the JDeveloper online Help. To access, select **Help > Help Topics** from the main menu. In the **Help Navigator** window, expand the **OA Component Reference** and **OA Region Styles** nodes.
- OA Framework ToolBox Tutorial / Sample Library
  - None

List of Values (LOV)

Overview

As described in Oracle Browser Look-and-Feel (BLAF) UI Guideline: LOV (List of Values), a List of Values (LOV) is a special control that lets users choose a value from a predefined list of specific values for the purpose of populating one or more fields on a page.

Implementation details for the following LOV types are described below.

- Text Field LOV
- Dependent LOV Text Field
- LOV Choice List
- Multiselect LOV (For a Table)

LOV Primer

Before learning how to implement a List of Values, it's important to understand its key components and behavior. This section briefly describes each of the following; implementation-level details are provided below for the different LOV types.

- Base Page and LOV Window
- LOV Mappings
- Dependent LOVs
- Validation (Also Known as "Autocompletion")
- AutoClear
- Look Ahead LOV
- Multiselect Look Ahead LOV

Base Page and LOV Window

A base page includes an LOV control in its page layout. For example, Figure 1 illustrates a "Purchase Order" LOV field. When the user selects the magnifying glass icon, the modal LOV dialog shown in Figure 2 displays. The user optionally searches within the LOV using designated query criteria, and selects a row from the results list to populate the value(s) in the base page (you can configure your LOV to return multiple values to different base page items). Note that the modal LOV dialog can only be closed by choosing a Quick Select value, or by choosing Cancel or Select. The modal dialog cannot be minimized.mltiselect

Note: The Quick Select column renders with 1% width in the results table of all LOV windows.

Figure 1: Purchase Orders and LOV Search.

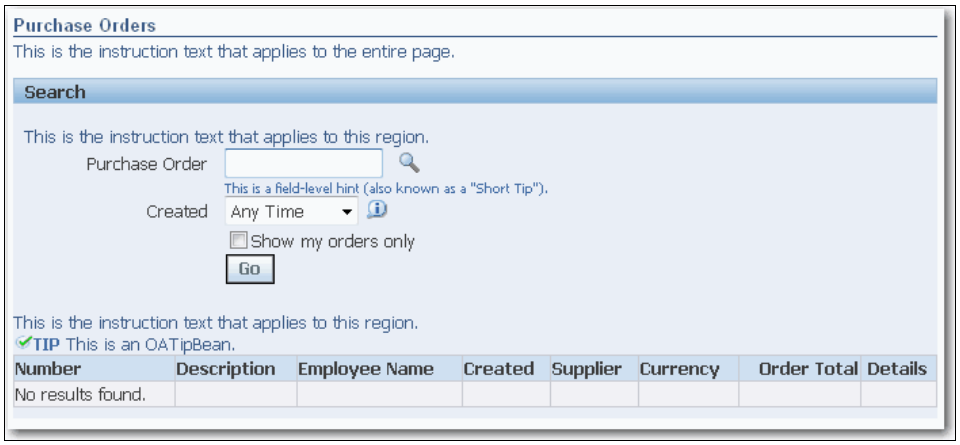
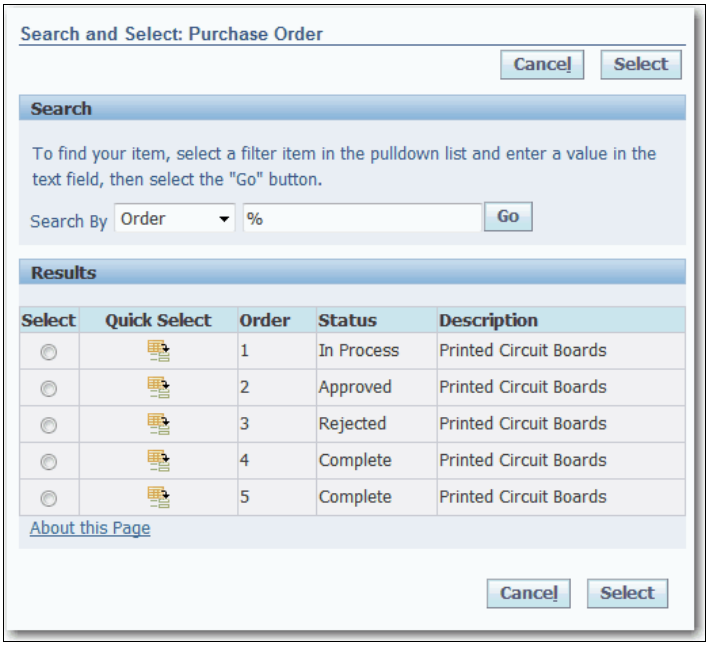
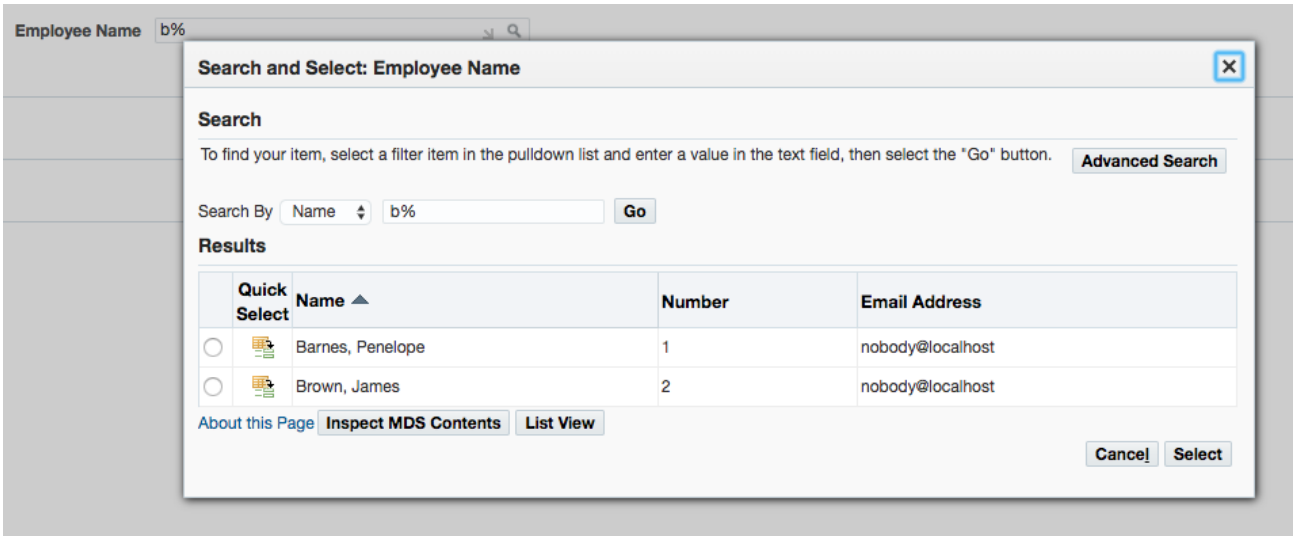


Figure 2: A LOV window with Advanced Search enabled



From Release 12.2.9, by default a LOV is opened in a popup, as shown in Figure 3 below. This behavior is controlled by setting the profile option "FND: Enable Open LOV In Popup" at the Site, Application, Responsibility or User level. The profile option affects all types of LOV, including Text Field LOV, LOV Choice List, and Multiselect LOV. If you need to open a LOV in a window as in previous releases, set the profile value to False.

Figure 3: A LOV Opened in a Popup



LOV Mappings

LOV Mappings define the data communication between the base page and the LOV window. An LOV map is comprised of the following participants:

LOV Item

The item in the LOV for which the mapping is defined.

(Base Page) Criteria

When the user invokes a List of Values, one or more field values from the base page can be passed to the LOV to be used as search criteria (note that the LOV field on the base page should *always* be configured as a search criteria item). When the LOV window renders, the query results are displayed. For example, in the purchase order example above, if the user enters "2" in the "Purchase Order" field and selects the LOV icon, the query will return all purchase orders whose number starts with "2."

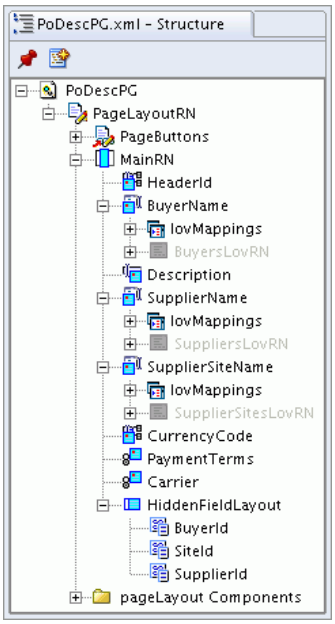
If you need to programmatically control the query criteria (for example, you need to intercept base page values to ascertain what the real query criteria should be), then you must configure the LOV to accept *passive criteria*, which is loosely defined to mean any LOV query criteria that you specify programmatically in an associated LOV controller.

(Base Page) Result

When the user selects a row in the LOV, one or more values are returned to the base page. In the ToolBox Tutorial Application example shown in Figure 1

above, each of the user-friendly "Buyer," "Supplier" and "Supplier Site" LOV fields also have hidden primary keys. When the user selects a value from the LOV, the OA Framework copies both the user-friendly name and the primary key values to the base page.

Figure 3: XML page structure showing the Buyer, Supplier and Supplier Site LOV fields and the corresponding hidden primary key fields.



Dependent LOVs

You can also configure an LOV value to depend on a value from another field(s). For example, in the ToolBox Tutorial Application example shown above, the user cannot select a supplier site value until a supplier is selected, so the "Supplier Site" LOV is configured to depend on the "Supplier" value.

Validation (Also Known As "Autocompletion")

LOVs can be configured to automatically validate data that the user enters without invoking the LOV window unless required. For example, if partial page rendering (PPR) is enabled and the user enters the value "Smi" in a Buyer name field and then tabs out, the OA Framework automatically queries all buyers whose name begins with "Smi."

In such a case, the following outcomes are possible:

- If a single match is found, the OA Framework automatically populates the result field(s).
- If multiple matches are found, the OA Framework displays the LOV window so the user can try different search criteria.
- If no matches are found, the OA Framework displays the LOV window so the user can try different search criteria.

  Note: PPR does not work with pages that have errors on them; the OA Framework resorts to a full page refresh in these cases.

Validate-On-Submit

For the successful validation of a LOV, you should map at least one more form element (preferably a hidden web bean) as a result item for the LOV web bean apart from the LOV web bean itself. Map this result item to a column in the LOV results table that has unique values.

The OA Framework also automatically validates hidden formValue LOV result fields when the page form is submitted (You can declaratively disable this for **formValue** fields and enable it for displayed LOV fields).

This second level of checking is important for the following reasons:

- If PPR is disabled, validation is simply not performed when the user enters a value and tabs out; no other result fields will be populated.
- Even if validation causes the LOV window to open, the result fields will still be empty if the user cancels out without making a selection.
- If the user enters a value in the LOV field and immediately selects a submit button, link or icon without tabbing out first, regular validation is not performed.

When validate-on-submit is enabled, the LOV behaves almost exactly the same as it does when performing the standard validation:

- If a single match is found, the OA Framework automatically populates the result field(s).
- If multiple matches are found, the OA Framework displays an error message at the top of the base page.
- If no matches are found, the OA Framework displays an error message at the top of the base page.

Note: If a user enters a value in a LOV field and selects the Submit button on the page without tabbing out of the LOV field first, validate-on-submit occurs, as described above. In this scenario, the `isLovEvent ()` event is not triggered hence the code that should be executed as a result of this event does not run. To work around this issue, identify all other cases when a piece of logic needs to be executed and include their required conditions in the same condition statement that also checks for `pageContext.isLovEvent ()`.

Passive Criteria

Previously, LOVs with passive criteria could not be properly validated. Even if a single valid match was found for the given criteria, the OA Framework displayed the LOV window so the user could select the single matching value. Now, even LOVs with passive criteria can be validated (assuming you enable validation on the LOV field as described in the implementation details below).

AutoClear

The OA Framework automatically clears dependent fields when the user changes the LOV value.

- First, if the user changes a value of the LOV field, or any criteria items for the LOV, and tabs out, the OA Framework clears all associated result field values.
- Second, if the user changes the value of an LOV criteria field and tabs out, the OA Framework clears the LOV field value.

Note that AutoClear cascades throughout LOV relationships. For example, assume the user changes the value of a criteria field and tabs out. The OA Framework clears the corresponding LOV input field, and all related results fields. If any of these result fields are designated as a criterion for different LOV field, that LOV input will also be cleared along with its result fields. This continues until all related LOV items have been cleared.

Warning: If an LOV result is written to a **messageStyledText** item, AutoClear cannot clear this field.

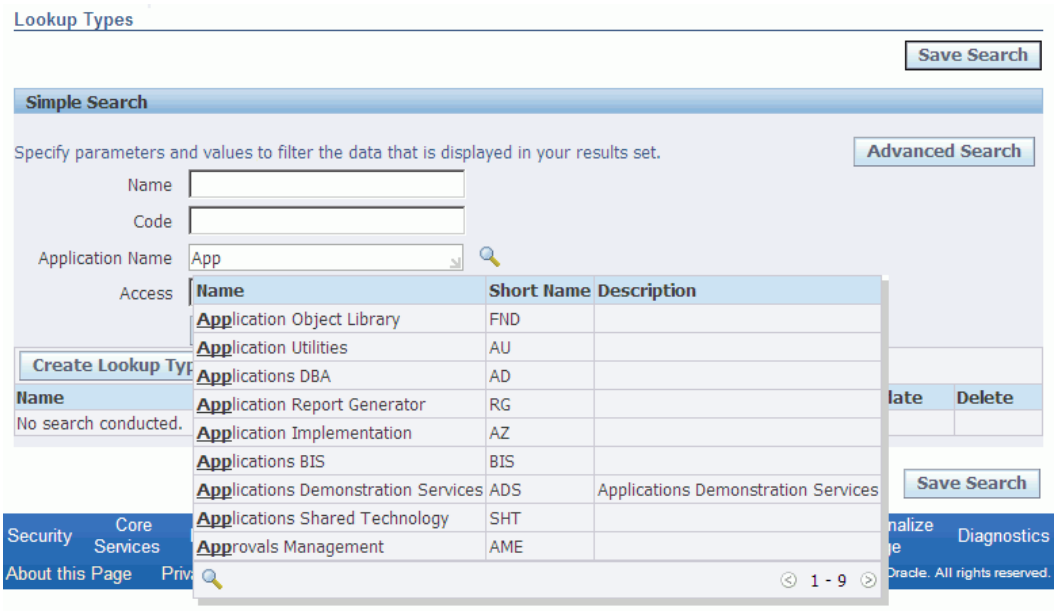
Look Ahead LOV

The List of Values component also has type-ahead search capabilities. As a user enters characters in the LOV search field, the results are fetched and displayed inline to the LOV component. The user can select a value from this look ahead window, just as in the classic LOV window. In most use cases, this capability eliminates the need to launch the LOV modal window, perform a search within it, and then navigate through the results using the **Next** / **Previous** links in the LOV window results table. The end result is a saving in the number of clicks and server-side requests, and a significant enhancement to user productivity.

Note: This feature is supported by Oracle E-Business Suite Release 12.1.2 and higher, and by a subset of the browsers certified with Release 12. For Microsoft Internet Explorer (IE), this feature is only supported in version 7.0 and higher.

The Look Ahead LOV is enabled by default. It is controlled by a property on the **messageLovInput** item and the profile option FND: Disable Look Ahead LOV at the Site or Application level. When Look Ahead LOV is enabled for a LOV component, a UI indicator of a dimmed downward arrow in the lower right corner of the LOV text input field appears.

Figure 4: The Look Ahead LOV window appears as a user enters data in the Application Name LOV.



An inline list of values appears only when a user enters 1 or more alpha-numeric characters into the LOV input field. You can specify the minimum number of characters required to initiate the Look Ahead LOV by setting a property on the **messageLovInput** item or by setting the profile option FND: Minimum Characters for Look Ahead at the Site or Application level.

Users can navigate the Look Ahead LOV window of matching records using the keyboard or mouse.

By default, the Look Ahead LOV performs a search for records that "start" with the alpha-numeric characters entered. You can change this behavior to search for records that "contain" the criteria entered, by setting a property on the **messageLovInput** item.

The matching records appear in the Look Ahead LOV window below the LOV text input field. The matching records update in real-time as the user enters or deletes characters in the LOV input field. By default, only a maximum of 50 records are fetched at a time, but you can change that maximum by setting a property on the **messageLOVInput** item. Users can click on the **Next/Previous** links in the window to fetch further or previous records.

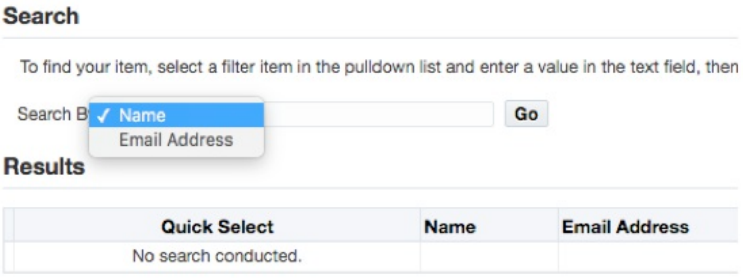
By default, the Look Ahead LOV displays 10 rows in its visible area. Depending on your page, you can change the number of rows to display in its visible area by setting a property on the **messageLOVInput** item.

By default, the Look Ahead LOV performs a search on the first searchable field. The **lookAheadMultiColumnSearchEnabled** property can be used to enable

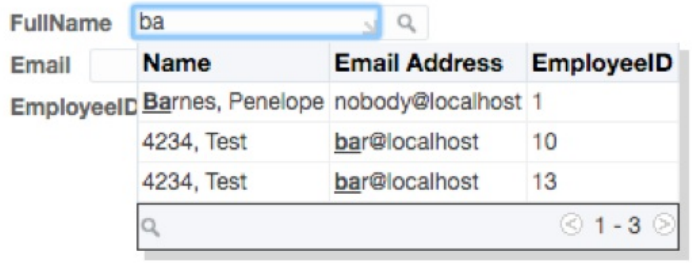
or disable look ahead LOV search on all searchable fields. The default setting of this property is 'false', in which case the look ahead LOV works as described above.

To use the **lookAheadMultiColumnSearchEnabled** property:

1. Set **lookAheadMultiColumnSearchEnabled** to 'true'.
2. Set the attributes of the LOV item under the LOV table (Search Allowed: True, Selective Search Criteria: True). For example, if you choose to use Name and EmailAddress:



3. This LOV item will be treated as a searchable field that will be shown in the "Search By" dropdown list in the LOV window as well
4. When the user enters a value such as 'ba' in the lookahead LOV input field, it will look for the value in all searchable fields. For example, an EmailAddress value such as 'ba%' or FullName value like 'ba%' will return all matching records for either field:



Keyboard Navigation

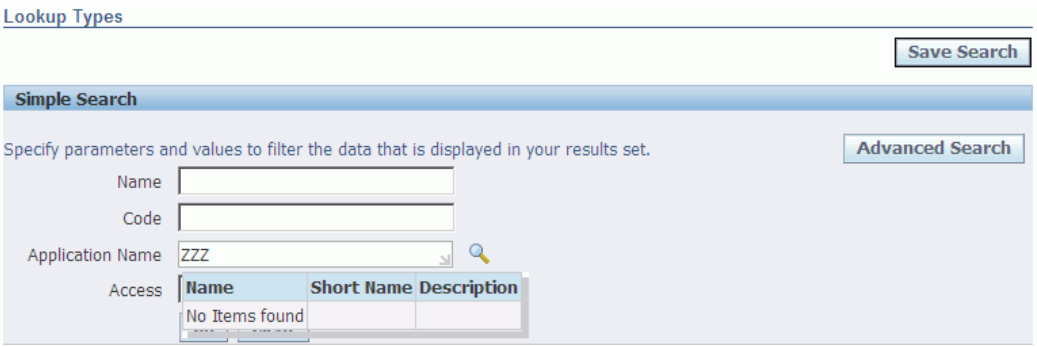
- **[Tab]** - Selects the highlighted record.
- **[Up Arrow]** - Highlights the previous record. If the currently highlighted record is the first record, the Up Arrow removes the highlighting from that record. If no record is currently highlighted, the Up Arrow has no effect.
- **[Down Arrow]** - Highlights the first record if no records are currently highlighted, or highlights the next record if a record is already highlighted.
- **[Esc]** - Dismisses the Look Ahead LOV window without selecting a record.
- **[Alt][<]** or **[Alt][Shift][,]** - Navigates to the previous page of records.
- **[Alt][>]** or **[Alt][Shift][.]** - Navigates to the next page of records.
- **[Alt][R]** - Invokes the LOV modal window where you can refine your search.
- **[Space bar]** - Selects the highlighted value.

Mouse Navigation

- Hovering over a particular record highlights the record.
- Left-clicking on the highlighted record selects the highlighted record and populates the record values into the result fields.
- Scrolling the mouse scrolls the LookAhead LOV window.

Note: If a user enters data into the LOV input field without first entering values for the LOV's criteria fields, the message about missing validation criteria appears, as in the traditional LOV modal window. If no records match the data that the user enters into the LOV input field, then the same message of 'No records found' appears as in the results table of the LOV modal window.

Figure 5: A 'No Items found' message displayed by the Look Ahead LOV feature if no record matches the data entered by the user.



Multiselect Look Ahead LOV

The Multiselect Look Ahead LOV feature is an enhancement to Look Ahead LOV. It is designed to allow display of multiple records, which can be selected one at a time. The LOV icon is removed, and only Look Ahead LOV is enabled for this purpose. It is based on the Faceted Search feature.

As the user enters the characters in the LOV search field (the default is 1), the results are displayed in the Look Ahead LOV dropdown. The user may select one value at a time, and the selected values are shown as checked checkbox values below the LOV field. Unchecking of the checkbox deletes the elements. Once an element is selected, it will be shown in the search results, but cannot be selected again unless deleted from the selected list.

Enabling Multiselect Look Ahead LOV

The Multiselect Look Ahead LOV feature is not enabled by default. There are two general ways to enable it:

- Declaratively, by setting the 'Enable Multiple Selection' property to 'True' at DT.
- Programmatically, by calling the setMultipleOptionEnabled API with the lovBean.

These are described in more detail below.

Declarative Implementation

To enable the multiselect Look Ahead LOV feature declaratively, you must:

1. Create the LOV and specify all mappings, ensuring the FormValues are correctly created.
2. In the Property Inspector for the LOV, set EnableMultipleSelection to 'True'.
3. Mark all FormValueBeans that participate in LOV mapping for enable multiple selection.

Programmatic Implementation

Enable multiple selection for the LOV and the formValueBeans that participate in the lovMapping:

```
OAMessageLovInputBean lovBean = (OAMessageLovInputBean)webBean.findChildRecursive("SearchEmpName");
  OAFormValueBean fv1 = (OAFormValueBean)webBean.findChildRecursive("SearchEmpNum ");
    ((OAMessageLovInputBean)lovBean).setMultipleOptionsEnabled(true);
((OAFormValueBean)fv1).setMultipleOptionsEnabled(true);
```

Controller Level Changes for Retrieving MultiSelect List

```
OAMessageLovInputBean lovBean = (OAMessageLovInputBean) webBean.findChildRecursive("SearchEmpName");

Hashtable results = lovBean.getMultiSelectLovResults(pageContext);
```

The results will be a hash table. Keys are the resultant fields in the lov mappings. In this case, there are two keys, SearchEmpName and SearchEmpNum. Each key will have a vector value that is nothing but the selected values. For example:

```
SearchEmpName  <barnes, joseph, test1, test2>
SearchEmpNum <110,113,456,567>
```

Initializing a MultiSelect LOV on Page Load Via the Controller

```
 OAMessageLovInputBean lovBean = (OAMessageLovInputBean) webBean.findChildRecursive("SearchEmpName");
 lovBean.setValue(pageContext, "Barnes,Penelope;Barnes,Penelope"
 OAFormValueBean formValueBean = (OAFormValueBean)     webBean.findChildRecursive("SearchEmpNum");
formValueBean.setValue(pageContext,"234;558");
```

Initializing a MultiSelect LOV with a Delimited String of Values

There may sometimes be a need to initialize a MultiSelect LOV with a string of two or more values, such as:

```
Val1;Val2
```

In such a case, ensure you separate Val1 and Val2 with the MultiSelect LOV delimiter. The default delimiter is the semicolon (;) as shown above. If necessary, you can use some other delimiter.

Using Placeholder Text With Multiselect LOV

Developers should provide placeholder text when using Multiselect LOV. This is because it does not have the LOV icon, but only the lookahead arrow symbol. The placeholder will help users understand usage of the component. It can be added at design time, or programmatically in the processRequest of the controller as follows:

```
OAMessageLovInputBean lovBean = (OAMessageLovInputBean)
webBean.findChildRecursive("ExternalLov");
lovBean.setAttributeValue(PLACEHOLDER_ATTR, placeholder);
```

Text Field LOV

As illustrated in Figures 1 and 3 above, a Text Field LOV is an updateable text field with an associated flashlight icon. Its behavior at runtime depends on your configuration decisions.

Declarative Implementation

Driving View Object Based on Entity Objects

If the view object that you are updating with the LOV selection results is based on entity objects, and your LOV will be returning values mapped to entity-object based attributes on one or more reference entity objects:

1. You must define associations between the driving entity and each reference entity.
2. In the view object definition wizard, you must configure each reference entity object's **Association End** as **Reference** and **Read Only**.

For example, assume you define a page to create a purchase order including an LOV to select a supplier:

- The LOV returns a `SUPPLIER_ID` foreign key value for update on the purchase order (`PurchaseOrderEO`), and a `SUPPLIER_NAME` value to display in the UI.
- The `SUPPLIER_NAME` value is mapped to a reference entity object attribute (`SupplierEO.SUPPLIER_NAME`).
- There is a reference association (`PoToSupplierAO`) that joins the `PurchaseOrderEO.SUPPLIER_ID` and the `SupplierEO.SUPPLIER_ID`.

When you configure the purchase order view object to reference the `SupplierEO` via the `PoToSupplierAO`, you must check both the **Read Only** and **Reference** properties.

If you follow these instructions, OA Framework does not attempt to write any values on reference entities (thereby avoiding a BC4J runtime error regarding updates to a Read Only entity), and BC4J's standard faulting mechanism is used to automatically query the reference values as needed.

LOV View Object and Application Module

Step 1: As described in Implementing the Model, define a view object for your list of values query (note that the OA Framework File Standards recommend creating this application module in the `oracle/apps/<app_short_name>/<module>/lov/server` directory).

- This view object, which must have a primary key associated with it, should include all the columns you want to display in the LOV results table, and any hidden values that you want to return when the user makes a choice. For example, a view object for selecting a buyer might include BUYER_NAME (displayed value), BUYER_ID (hidden primary key), and JOB_TITLE (displayed value).
- You do not need to include a WHERE clause for the base page search criteria items; the OA Framework automatically appends an appropriate WHERE clauses and bind variables based on the LOV criteria. That said, however, your LOV can include a WHERE clause that always restricts the result set (for example, an END_DATE check for a "Currently Employed Buyers" LOV).
- The view object for the *LOV region* can be based on an entity object, but given its limited data selection, it's typically created using plain SQL.

Step 2: If you have not already done so, create a dedicated application module (AM) to contain all view objects that are shared by a package or module (note that the OA Framework File Standards recommend creating this application module in the `oracle/apps/<app_short_name>/<module>/lov/server` directory). Add the view object that you created in Step 1 to this application module.

LOV Field and (Optional) Additional Criteria/Results Fields

Step 3: In the OA Extension Structure pane, select the region where you want to add the LOV field, right-click and select **New > Item**. Set the `Item Style` property to **messageLovInput**.

- Step 3.1: Specify a standards-compliant `ID`, an `Attribute Set` and other properties as usual for text fields.
- Step 3.2: If you don't want the LOV to automatically validate when the user enters a value and tabs out or submits the form without tabbing out, set the `Disable Validation` property to **True**. Note that, per the OA Framework View Coding Standards, this value should always be **False** unless it's essential that you allow partial values in the field (in a search region, for example).
- Step 3.3: To enable or disable the Look Ahead LOV feature, set the `Look Ahead Enabled` property to **True** or **False**, respectively. You can also update the profile FND: Disable Look Ahead LOV at the Site or Application level to **False** to enable the Look Ahead feature for all LOVs. The default value for this profile option is **False** at the Site level. Note that the value you set on the `Look Ahead Enabled` property overrides the profile value for this LOV.
- Step 3.4: Control the minimum number of alpha-numeric character inputs required to activate the Look Ahead LOV feature by setting the profile option FND: Minimum Characters for Look Ahead at the Site or Application levels. The default value for the profile option is **3**. To override this profile option value and set a different minimum number of characters to activate the Look Ahead feature for this LOV, set the `Minimum Characters For Look Ahead` property to a value of 1 or higher.

  Note: For smaller width LOVs, that is, those LOVs whose width is 5 characters or less, if the width of the **messageLovInput** item is smaller (for example, 2) than the width set on the `Minimum Characters For Look Ahead` property or the FND: Minimum Characters for Look Ahead profile (for example, 3), then the minimum number of characters required to activate the Look Ahead LOV defaults to1.
- Step 3.5: Set the `Look Ahead Search Type` property to specify the search type of the Look Ahead LOV. Set the property to **contains**, to have the Look Ahead LOV return records whose LOV field value contains the matching characters anywhere within the value. Set the property to **startsWith** to have the Look Ahead LOV return records that start with the criteria entered. The default value for this property is **startsWith**.

  Note: For selective search criteria LOVs, `Look Ahead Search Type` cannot use the **contains** value.
- Step 3.6: By default, the Look Ahead LOV window fetches a maximum of 50 records. A scroll bar along with **Next** and **Previous** links enable navigation. Set the `Look Ahead Records Displayed` property to specify a different maximum. If more than the maximum number of records match the user's input, then **Next** and **Previous** links appear in the window to let the user navigate among the records.
- Step 3.7: By default, the Look Ahead LOV window displays 10 rows in its visible area. Set the `Look Ahead Records Visible` property to specify a different number of rows to display in the visible area to suit the needs of your specific page.
- Step 3.8: The `Look Ahead Selection Event Enabled` property lets you determine what happens when the user selects a value in the Look Ahead LOV window. If the property is set to:
  - **True** - a server-side LOV validate event fires, and the page refreshes with the resulting fields.
  - **False** - no server-side event fires, and the results are based on the client side.

  Note: According to the OA Framework View Coding Standards, this value should always be **True** if you have special handling in your controller

for LOV events, like using `isLovEvent()`.

Step 4 (optional): Create additional items for LOV result values. For example, if you create an LOV for "Employee Name," you will probably need a hidden field for the "Employee ID" value. Note that you should set the `Style` to **formValue** and the `Rendered` property to **True** for all hidden fields used for LOV results. Items of type **messageStyledText** are also valid for results.

Note: If your result field for the LOV is a read-only field that can assume a null value, you should create a mirror formValue field and associate it with the same view attribute as the read-only field. Also, create a lov mapping for this formValue web bean similar to the read-only field. This ensures that any null result value gets saved/updated in the view object.

Step 5 (optional): Create additional items for LOV search criteria. Note that LOV criteria items must be form elements (for example, they cannot be of type **messageStyledText**). If not, the LOV cannot read their values. For example, if you define a text input field as an LOV criteria item, but you set its `Read Only` property to **True**, it is not rendered as a form element.

Tip: If you want to use the value that you display in a **messageStyledText** field as LOV search criteria, consider creating a mirror **formValue** field to hold the same value. LOVs do allow **formValue** items as criteria items.

Warning: Query criteria for supplemental fields (in addition to the LOV field itself) must be used carefully. For example, assume an "Employee Name" LOV is configured to apply a "Department" value as query criteria. If the user enters a value of "Sales" in the "Department" field and selects the "Employee Name" LOV icon, all employees in the sales department are displayed.

- The value entered in the criteria field is used to perform an *exact match* (the base page LOV field's value is an exception to this rule as it is always used to perform a *partial match*). As a consequence, if the user enters "Sa" in the "Department" field and invokes the "Employee Name" LOV, the query will not find any employees in the "Sales" department.
- Supplemental query criteria items cannot be used as user enterable search criteria within the LOV itself. So, in our "Department" example, the LOV cannot be configured to let the user search on "Department" *within* the LOV window (if she realizes she wants to search in the "Consulting" department instead, she needs to dismiss the LOV and change the "Department" value on the base page before opening invoking the LOV a second time).

Step 6: (optional) As of Release 12.2.6, set the `Placeholder Text` property if you wish to show a short hint describing the expected value for the LOV text input field.

LOV Region

Step 7: Create the LOV region itself. First, you need to decide if you want to create an "external" LOV that can be leveraged in multiple pages by defining unique mappings for each instance, or a single-use LOV for use only in the current page. Instructions for each are provided below.

To create a single-use LOV region:

- Step 7.1 When a you create a **messageLovInput** item, OA Extension automatically creates an inline LOV region (**listOfValues** style) for this item.
- Step 7.2: Select your new **listOfValues** region in the Structure pane, and set its `AM Definition` property to the fully qualified name of the application module you created in Step 2 above (for example, `/oracle/apps/fnd/framework/toolbox/tutorial/lov/server/TutorialLOVAM`).
- Step 7.3: Select the **listOfValues** region again, right-click and select **New > Region Using Wizard** to create your results table and bind it to the view object you created above (see the Tables documentation for additional information about creating this component).
    - Remember to give the region and its items standards-compliant `IDs`, and specify appropriate `Attribute Sets` for the items.
    - Set the item `Style` to **messageStyledText** for any items you want to display in the LOV "Results" table. Note that at least one table item must be of type **messageStyledText** for the LOV to render properly.
    - Set the item `Style` to **formValue** for any hidden items (remember to set their `Rendered` property value to **True** so the OA Framework includes them in the web bean hierarchy).
    - Set the `Search Allowed` property to **True** for any LOV result items you want to present to the user as searchable values. These items are listed in the search poplist the user sees in the LOV window.
        - *At a minimum you must set the* `Search Allowed` *property to True for the* the result table item corresponding to the LOV field on the base page.
        - Do NOT set the `Search Allowed` property to **True** for any result table items that correspond to the supplemental search criteria items that you created in Step 5 above.
    - Set the `Selective Search Criteria` property to **True** to identify items for which the user must provide search criteria (see the Search topic for additional information about selective search criteria).
        - In the simple LOV search, only those items that have *both* the `Search Allowed` and `Selective Search Criteria` properties set to **True** will appear in the **Search By** poplist. At runtime, the user must provide a real search value; if the user enters **%** or tries to perform the search without specifying a **Search By** value, the OA Framework displays the standard selective search criteria error message.

          Note: For backwards compatibility, simple searches will function as they always have if none of the items are marked with the `Selective Search Criteria` property set to **True**.
        - In the advanced LOV search (instructions for enabling an advanced search are provided below), all items whose `Search Allowed` property is set to **True** will render, however, the user must enter a value in at least one of the designated `Selective Search Criteria` fields.

          Note: If you enable an advanced search, you must designate at least one `Selective Search Criteria` item.
- Step 7.4 (optional): Select the **listOfValues** region again, right-click and select **New > searchInstructions** to provide custom help text for the LOV search region. Specify a standards-compliant `ID`, set the `CSS Class` to **OraInstructionText** and specify the Message Dictionary message to display as the help text by setting the `Tip Message Appl Short Name` and the `Tip Message Name` as appropriate.

To create a reusable LOV region:

- Step 7.1: Create the shared region (not a page!) using the instructions provided in Implementing the View: Create a Shared Region. Set its `Style` to **listOfValues**.
- Steps 7.2 - 7.4: Follow as documented above for the single-use LOV region.
- Step 7.5: Associate the reusable LOV with the base page LOV field. In the OA Extension Structure pane, select the LOV item you created in Step 3 and set its `External LOV` property to the fully qualified name of the shared **listOfValues** region you just created (for example, `/oracle/apps/fnd/framework/toolbox/tutorial/lov/webui/EmployeesLovRN`).

  Note: OA Extension confirms that you want to replace the default generated inline LOV region with the external LOV. Select the **OK** button to proceed, and OA Extension will remove the inline LOV region and display the external LOV in a dimmed state since you cannot edit a referenced object.

  Tip: If you change your mind and want to create an inline LOV after setting an external LOV, select the **Set to Default** Property Inspector toolbar button to clear the `External LOV` property. OA Extension automatically recreates the default inline LOV region for you.

LOV Mappings

Step 8: Create the LOV Mappings (regardless of whether you choose to implement a reusable LOV or a single-use LOV, you map its data communication relationship to the base page in the same way).

For the first mapping, select the LOV mapping that was created by default when you created your **messageLovInput**. To create additional mappings, select the **lovMappings** node in the Structure window, right-click and select **New > lovMap** from the context menu.

To configure a mapping:

- Specify the `LOV Region Item` that will participate in the mapping.
- Specify the `Criteria Item` for a base page item whose value is to be used as LOV search criteria.
- Specify the `Return Item` for a base page item whose value is to be populated by the LOV selection.
- Set the `Required` property to **True** for Criteria Items whose values must be populated before the LOV can be invoked (if not, the OA Framework displays an error message in the LOV window).
- Set the `Programmatic Query` property to **True** for any Criteria Items whose values you want to apply to the WHERE clause programmatically.

  Tip: you might use this approach, for example, if you have supplemental Criteria Items whose values should be used for partial matches instead of the default OA Framework behavior of an exact match (remember that the LOV field value itself is always used for a partial match).
- Set the `Use for Validation` property one of the following values:
  - **default** validate-on-submit will be triggered if the base item is of type **formValue,** and if it has no value (this is the OA Framework 5.7 behavior)
  - **yes** validate-on-submit will be triggered if the base item has no value regardless of the item type.
  - **no** validate-on-submit is not triggered by a null value regardless of the item type.

    Note: if you want to turn off validate-on-submit altogether for an LOV input, you need to set the `Use for Validation` property to **no** for all LOV maps whose base item is of type **formValue.**

When configuring your mappings, pay close attention to the following usage notes:

- First and foremost, you need one mapping for each distinct field on the base page that you want to use as criteria and/or to which you want to return a result value. A single LOV mapping for the LOV field can handle both sending criteria values to the LOV, and receiving result values from the LOV. Note that older pages migrated from previous versions of JRAD or AK may show separate mappings for each direction.
- The data types for the `LOV Region Item` and the `Criteria/Results Items` must match. If they do not, and you are running in Developer Test Mode, you will receive an error.
- **You must have a mapping for the base page LOV field.** Specify the name of that field for both the `Criteria Item` and `Return Item` properties. For the `LOV Region Item` property, specify the item in the LOV region that corresponds to the base page LOV field. If you fail to configure the LOV field as a `Criteria Item` and you are running your page in Developer Test Mode, the OA Framework will display an error.
- **In LOV mappings for fields other than the LOV field itself, you can specify either the `Criteria Item` or the `Return Item` property, but not both.** Do not try to create one mapping for a (non-LOV field) field as criteria and another mapping for the same field as a return item, as this will cause runtime errors. For the `LOV Region Item` property, specify the item in the LOV region that corresponds to the appropriate base page field.
- If your LOV does not naturally include a **formValue** result item (so it returns its values only to visible fields), you must add at least one **formValue** result field whose `Use for Validation` property is set to **True**. For example, consider the following Address example:

LOV Usage	Item Name	Item Type
criteria	AddressLov	messageTextInput
result	AddressLov	messageTextInput
result	City	messageTextInput
result	State	messageTextInput

In this case, we have an AddressLov field with associated City and State fields. When the user selects a value from the Address list of values, results are returned to the AddressLov field and its related City and State fields. This configuration can lead to the submission of invalid values when partial page rendering (PPR) is disabled. To resolve this, simply add a **formValue** item for one of the results as shown:

LOV Usage	Item Name	Item Type
criteria	AddressLov	messageTextInput
result	AddressLov	messageTextInput
result	City	messageTextInput
result	State	messageTextInput
result	**AddressFormValue**	**formValue**

If the user tries to submit invalid values for the address fields, the OA Framework detects that the AddressFormValue field is null (meaning that it has not been populated by a valid LOV result value), and so it will validate the values on submission.

General Usage Notes

- The base page's LOV value is applied only to the first, automatic query that the OA Framework executes when the LOV window first opens (or for autovalidation, when the user leaves the LOV field after entering a partial value). Any subsequent searches performed by the user in the LOV window do not use the base page LOV value.
- If there are no criteria in the base page LOV field, the OA Framework does not perform an automatic query when the LOV window opens.
- Query criteria from fields *other* than the LOV field are not displayed in the LOV window, but they will affect all queries executed in the LOV window.
- Any query criteria values from items configured as passive criteria are not automatically added to the WHERE clause; you must manually set them in a controller as described above.
- A table and an LOV used in a table should always used different view objects to avoid stale data errors.
- If an LOV is used in a table or an HGrid, you cannot map to criteria or result fields outside the table or HGrid.
- If you make an LOV text input field read only, the OA Framework hides the LOV icon and renders the data in **messageStyledText** item.
- If an LOV is used in a table with an "Add Another Row" button, and the LOV returns a result to a **messageStyledText** item, add a mirror **formValue** item for the **messageStyledText** item and map it to the same underlying view object attribute. Then, add a special LOV map to return the same value to the **formValue** field that you return to the **messageStyledText** field. This ensures that the data is properly written to the underlying view object.
- If the selected LOV value is longer than the specified maximum length of the LOV field, browsers exhibit different runtime behaviors. For example, Internet Explorer displays a client-side validation error message while Mozilla-based browsers return a truncated value to the base page. When designing your LOV, it is therefore desirable to ensure that the mapped value lengths are consistent.

Enabling Advanced Search in Your LOV

If you want to enable an advanced search in your LOV, in addition to the default simple search, set the `Advanced Search Allowed` property on the **listOfValues** region to **True**.

You can also set this property programmatically by calling `setAdvancedListOfValues(Boolean.true)` on the `OAListOfValuesBean`. Note that this should be called only in the `processRequest` method of a controller associated with the **listOfValues** region.

Advanced Search Usage Notes

- Per the BLAF UI guidelines, the Simple Search always displays by default, even if the Advanced Search is enabled. This state is illustrated in Figure 1 above.
- If the user searches in one mode and toggles to the other, the underlying query criteria object is cleared to avoid inconsistency.
- You should not make any changes to the underlying query criteria object yourself as this might destabilize the LOV.
- The Advanced Search displays any **listOfValues** region items whose `Search Allowed` property is set to **True**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or easily extended .

See the OA Framework Controller Coding Standards for additional information about this and other guidelines that you should consider when writing web bean manipulation code.

Instantiate

The following example code illustrates how to create a Text Field LOV programmatically. See the `oracle.apps.fnd.framework.webui.beans.message.OAMessageLovInputBean` Javadoc for additional information about these methods.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   super.processRequest(pageContext, webBean);
   OAMessageLovInputBean lovInput =
      (OAMessageLovInputBean)createWebBean(pageContext, LOV_TEXT, null, "inputTest");
   webBean.addIndexedChild(lovInput);
   // Specify the path to the base page.
   lovInput.setAttributeValue(REGION_CODE, "/oracle/apps/dem/webui/Basic");
   // Specify the application id of the base page.
   lovInput.setAttributeValue(REGION_APPLICATION_ID, new Integer(20001));

   // Specify the LOV region definition.
```

```
    lovInput.setLovRegion("/oracle/apps/fnd/framework/toolbox/tutorial/webui/EmployeesLovRN", 0);

    // Validation should be enabled for LOVs unless it's essential for the field to allow
    // a partial value (in a "Search" region, for example).
    lovInput.setUnvalidated(false);

    // Configure the LOV mappings.
    // Note that you must call this method after you add the messageLovInput item
    // to the web bean hierarchy.
    lovInput.addLovRelations(pageContext, "inputTest", // base page item
                                          "Empname", // lov item
                                          LOV_RESULT, // direction
                                          LOV_REQUIRED_NO);

    lovInput.addLovRelations(pageContext, "inputTest", // base page item
                                           "Empname", // lov item
                                           LOV_CRITERIA, // direction
                                           LOV_REQUIRED_NO);
    // Enable Look ahead.
    lovInput.setLookAheadEnabled(true);

    // Set the Search Type as contains.
    lovInput.setSearchType(CONTAINS);

    // Change the default number of records displayed to 100.
    lovInput.setLookAheadSize(100);

    // Enable Client side population.
    lovInput.setLookAheadSelectionEventEnabled(false);

    // Set the minimum characters to initiate look ahead as 1.
    lovInput.setMinCharsForLookAhead(1);
}
```

Usage Note: For the above code example, the `REGION_CODE` and `REGION_APPLICATION_ID` attributes must be set to the path of a valid base page and to that valid base page's application ID, respectively. These attribute values must represent an existing personalizable static region reference and cannot be set to arbitrary values. OA Framework validates these values when it renders "**Personalize ...**" region links on the page and a Java exception results when these values do not represent a valid combination.

The following example illustrates how to add an LOV relation programmatically to a declaratively defined LOV. See the `OAMessageLovInputBean` Javadoc for additional information about these methods.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
    super.processRequest(pageContext, webBean);
    OAMessageLovInputBean msglov =
      (OAMessageLovInputBean)webBean.findChildRecursive("Employee");

    msglov.addLovRelations(pageContext,
                          "Employee", // base page item name
                          "EmpName", // lov item name
                          LOV_CRITERIA, // direction
                          LOV_REQUIRED_YES);
}
```

Configure WHERE Clause Based on Passive Criteria

If you have configured one or more criteria items as passive criteria, you must obtain the passive criteria values and manually apply them to the WHERE clause in a controller associated with the LOV region as illustrated in the Dependent Text Field LOV / Passive Criteria Example below.

Switch LOV View Objects Based on Passive Criteria

If you need to change the view object an LOV queries dynamically, create a controller and associate it with the LOV region. Then, create one or more passive criteria items that you can inspect in this controller's `processRequest` method to determine which view object to query.

Remember to add the view object that you dynamically select to your LOV's application module.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
  Dictionary passiveCriteria = (Dictionary)pageContext.getLovCriteriaItems();
  String lov = (String)passiveCriteria.get("SwitchLOV");
  if ((lov != null) && !("".equals(lov)))
  {
    ((OAListOfValuesBean)webBean).setViewUsageName("<ViewInstanceName>");
  }
}
```

Use the LOV as a Context Switcher

The LOV partial page rendering (PPR) behavior differs significantly from other beans that you might configure to enable PPR events. Specifically:

- The value(s) selected by the user was not immediately posted to the base page's underlying view object.
- Only the values of the result items were available to developers trying to catch the LOV PPR event.

The LOV behaves in a consistent manner with other beans:

- The value(s) selected by the user are immediately posted to the base page's underlying view object.
- When the user tabs out of the LOV field, the OA Framework now submits the form. All the base page's form values are written to the underlying view objects as expected.
- When setting a value in a table row with an LOV, you can use the `EVENT_SOURCE_ROW_REFERENCE` to identify the row as described in the Dynamic User Interface documentation.

If you want to make changes in your page based on an LOV selection, add the following code to a `processFormRequest` method associated with the base page. Note that the LOV automatically fires a predefined PPR client action. This is differs from standard PPR client actions, which you must explicitly enable for items such as a poplist.

```
if (pageContext.isLovEvent())
{
   // Form was submitted because the user selected
   // a value from the LOV modal window,
   // or because the user tabbed out of the LOV input.

   // Find out which LOV input triggered the event.
   String lovInputSourceId = pageContext.getLovInputSourceId();

   // At this point, all the data is available in the base VO, just as in
   // regular PPR events. Invoke an AM method to update the application
   // properties VO.
   //
   // if ("myLovInput".equals(lovInputSourceId))
   // {
   //   am.invokeMethod("handleMyLovInputEvent");
   // }
}
```

The `pageContext.isLovEvent` method returns **true** if the event value is **LOV_UPDATE** (meaning the user selected a value from the LOV modal window), or **LOV_VALIDATE** (meaning the user tabbed out of the LOV input field on the base page).

Note that it is no longer necessary to use the method `pageContext.getLovResultsFromSession` to retrieve selected values since the LOV results are now available in the base page view object with all the other form values.

See the Dynamic User Interface documentation for a detailed explanation of how to handle PPR events in general (once you have identified the LOV event, the code procedure is the same for the LOV as it is for all other PPR-enabled beans).

Enabling UIX Client-Side Validation for a LOV Text Input String

Generally, UIX performs client-side validation of user input strings when a user explicitly submits a form by selecting the **Submit** button. It does not, by default, perform this validation if a user implicitly submits the form, such as by invoking a LOV window.

UIX client-side validation ensures that an input string conforms to specified definitions. For example, the NLS_NUMERIC_CHARACTERS parameter may be set to ".," to specify "." (period) as the decimal character and "," (comma) as the group separator for all numbers in an environment. If a user then enters the invalid value "12,34" into a MessageTextInput field and explicitly submits the form, UIX client-side validation would alert the user to the invalid string. If a user enters this invalid string in an LOV text input field and implicitly submits the form by invoking the LOV window, client-side validation would not occur. The string "12,34" becomes "1234" based on more lenient server-side number recognition and the LOV window returns unintended values for "1234".

As of Release 12.1.3, you may programmatically call the UIX `setValidateOnLovPrepare()` API on `OAMessageLOVInputBean` as shown below to enable UIX client-side validation even when a user implicitly submits a form, such as by invoking a LOV window:

```
OAMessageLOVInputBean.setValidateOnLovPrepare("true")
```

For example:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  OAMessageLovInputBean lov = (OAMessageLovInputBean)webBean.findChildRecursive("UserIdLov");
  lov.setValidateOnLovPrepare("true");
}
```

To get the current validation status, call the `getValidateOnLovPrepare()` method, which returns the string `"true"` if validation is enabled or `"false"` if it is not. The default is `"false"`.

Refer to the `oracle.cabo.ui.beans.form.LovInputBean` Javadoc for additional information about these methods.

Defaulting an LOV Field Properly

When you populate a LOV field programmatically, you should also populate any relevant result fields that need to be validated. Include result items that have the following set on the LOV map:

- `LOV_EMPTY_VALUE_ALLOWED` property is set to **LOV_NO**

  and

- `LOV_EMPTY_VALUE_ALLOWED` property is not set and whose item style is **formValue**.

If you do not default the LOV field as suggested, OA Framework will try to auto-populate the result fields based on the query results of the LOV input value. If

the query returns multiple rows or no rows, the auto-population will fail and the user will encounter a LOV validation error message.

Personalization Considerations

- "**Personalize...**" region links are not displayed on LOV modal windows. To personalize an LOV table, use the "**Personalize Page**" link on the top of the page containing the LOV. Tables in internal LOVs (in-line LOVs) can be personalized directly from the Personalization Hierarchy page. To personalize a table in an external LOV, use the "Choose Context" page to select the shared LOV region.

Dependent Text Field LOV

As described above, users cannot select a value in a "dependent LOV" until one or more driving field values have been set. For example, it does not make any sense to select an employee's job title until you have selected an employee.

Furthermore, when the user clears a value in a driving field, the dependent LOV field(s) should be cleared also.

To illustrate how to create a dependent LOV, the following describes how to create the Supplier and Supplier Site LOVs in the ToolBox Tutorial Application's `oracle/apps/fnd/framework/toolbox/tutorial/webui/PoDescPG page`.

Warning: When configuring dependent LOVs, be careful not to create cyclical dependencies that cannot be resolved.

Step 1: Configure the Supplier and Supplier Site **listOfValues** regions as described in the Text Field LOV section above.

Step 2: Create and configure the Supplier **messageLovInput** item and the Supplier ID hidden **formValue** item as described above.

Step 3: Define typical lov mappings for the Supplier field: one for the Supplier criteria/result, and a second for the Supplier ID result.

Step 4: Create and configure the Supplier Site **messageLovInput** item and the Supplier Site ID hidden **formValue** item as described above.

Step 5: Define typical lov mappings for the Supplier Site field: one for the Supplier Site criteria/result, and a second for the Supplier Site ID result.

Step 6: Create an lov map to define the Supplier ID value as criteria for the Supplier Site LOV.

- Set the `LOV Region Item` to the LOV's Supplier ID item.
- Set the `Criteria Item` to the base page's Supplier ID item.
- Leave the `Programmatic Query` property as **False**.
- Leave the `Required` property as **False**.

Step 7: Create a special lov map to define the Supplier Site LOV as being dependent on Supplier.

- Set the `LOV Region Item` to the LOV's Supplier item.
- Set the `Criteria Item` to the base page's Supplier item.
- Set the `Required` property to **True**. This tells the LOV that there must be a value in the Supplier field before the user can use this LOV (so this field is dependent on the Supplier field).
- Set the `Programmatic Query` property to **True**.

Step 8: Create a controller and assign it to the Supplier Site **listOfValues** region. This controller checks the value for the Supplier Name, and if it is null, displays an appropriate error message (the OA Framework displays a generic message otherwise).

Tip: Since the OA Framework displays an error message for this *after* the user selects the LOV icon with a null driving field, it is advisable to include some hint text for the dependent field that advises users of the need to select values in a specific sequence.

```
import java.util.Dictionary;
...

/**
 */
public class SupplierSiteLOVCO extends OAControllerImpl
{
   public static final String RCS_ID="$Header: SupplierSiteLOVCO.java 115.0 2003/02/24 06:49:33 nigoel noship $";
   public static final boolean RCS_ID_RECORDED =
     VersionInfo.recordClassVersion(RCS_ID, "oracle.apps.fnd.framework.toolbox.lov.webui");

   public void processRequest(OAPageContext pageContext, OAWebBean webBean)
   {
     super.processRequest(pageContext, webBean);
     // Get the list of items configured as "passive criteria" for the LOV.

     Dictionary passiveCriteriaItems = pageContext.getLovCriteriaItems();
     String supplierName = (String) passiveCriteriaItems.get("SupplierName");

     // Note: supplierName should not be null since it is defined as a required
     // passive criterion. An exception is raised if it is null. The OA Framework typically
     // handles this itself, and generates a generic error message at the top of the
     // Supplier Site LOV page if there is no
     // supplier name...

     if (supplierName == null || ("".equals(supplierName)))
     {
       throw new OAException ("AK", "FWK_TBX_T_PO_SUP_BEFORE_SITE");
     }
```

```
    // IMPORTANT: DO NOT EXECUTE THE VO QUERY! The LOV code will take care of this
    // based on your lov mappings.
  }
}
```

LOV Choice List

The LOV Choice List is a hybrid between a poplist and a List of Values. It initially renders as a poplist with a maximum of 30 items and a "More..." option (note that the "More..." option always displays even if the result set has fewer than 30 items).

Note: The poplist always renders initially with just a "More..." option.

If the desired item is not found, the user can invoke a full List of Values modal window by selecting the "More..." option in the poplist. When the user makes a choice from the full List of Values, the OA Framework adds the selected value to the poplist's values permanently for the user. In other words, the poplist expands over time based on the user's ongoing List of Values selections.

Declarative Implementation

To add an LOV Choice List to your page:

Step 1: Create a shared LOV region as described in the Text Field LOV section above. Note that the view object you create must have at least two columns that can be mapped to a poplist: a developer key and a display value. For example, in an "Employees" view object, you might include an EMPLOYEE_ID (PK) column and an EMPLOYEE_NAME column to satisfy this requirement. For the LOV window that displays when the desired value isn't found in the poplist, you can include additional columns (DEPARTMENT_NAME, MANAGER_NAME and so on).

Note: The value that you will designate as the developer key cannot exceed 30 characters in length.

Step 2: In the OA Extension Structure pane, select the region to which you want to add the LOV Choice List, right-click and select **New > Item**.

- Step 2.1 Specify a standards-compliant `ID,` and set the `Style` to **messageLovChoice**.
- Step 2.2 Apply the `Attribute Set` associated with the value you are displaying in the poplist. For example in the ToolBox Sample Library, the LOV Choice List created to display employees uses the attribute set `/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxEmployees/FullName`.

Step 3: Set the **messageLOVChoice's** `External LOV` property to the shared LOV region you created in Step 1. Note that this must be set to a fully specified region name (for example, `/oracle/apps/fnd/framework/toolbox/lov/webui/EmployeesLovRN`).

Step 4: Configure the poplist. Set the `Picklist Display Attribute` and the `Picklist Value Attributes` to the view object attribute names for the corresponding items in the LOV you selected in Step 3.

For example, the ToolBox Tutorial Sample Library LOV Choice List mentioned in Step 2 above uses the `/oracle/apps/fnd/framework/toolbox/lov/webui/EmployeeLovRN` region as its LOV. This region's table binds to the `EmployeeNamesVO1` instance as follows:

LOV Region Item ID	Item View Object Attribute
EmpName	EmployeeName
EmpNum	EmployeeNumber

In this case, the `Picklist Display Attribute` value is **EmployeeName**, and the `Picklist Value Attribute` is **EmployeeNumber** (note that the view object attribute associated with the `Picklist Value Attribute` must be less than or equal to 30 characters).

Step 5: Verify that the **messageLovChoice's** `Data Type` property value matches the data type of the view object attribute you specified for the `Picklist Value Attribute`.

Step 6: If you want to allow end-users to personalize the LOV Choice List (add, remove, or reorder values in the list), set the `List Personalization` property to **True**.

Step 7: Select the LOV map that OA Extension created for you and configure it as follows:

- Set the `LOV Region Item` property to the name of the LOV region item name whose value maps to the `Picklist Value Attribute` property you set in Step 4. In the ToolBox example described above, this is **EmpNum**.
- Set the `Return Item` property value to the name of the LOV Choice list item.
- Set the `Required` property to **False**.
- Leave the `Use for Validation` value as **default** (the OA Framework ignores this as it doesn't apply in this case).

Note: To avoid the problem of choice list values that are no longer appropriate for the specified criteria value, do not specify a `Criteria Item` when configuring your LOV Choice map.

Note: You may define multiple LOV maps to return values to additional items, however the values for these additional items are only returned when a user returns from the LOV modal window (as a result of choosing the "More..." option). The values are not populated into the additional items if a user simply chooses a value already in the LOV Choice poplist.

Personalization Considerations

Refer to the Personalizing a LOV Choice List in the Oracle Application Framework Personalization Guide for additional information.

Multiselect LOV (For a Table)

As an alternative to the Add Another Row button, you can create a multiselect LOV to let users quickly populate multiple rows in a table. For example, with a single navigation to a multiselect LOV, a system administrator could select several Oracle Application responsibilities for assignment to an employee. There is no need to select the responsibilities one at a time.

When you implement a multiselect LOV, a special global table button renders to provide access to the multiselect LOV. When the user selects this button, the multiselect LOV window displays (this is the same as a regular LOV window, except that the user can select multiple rows). When the user makes a selection, the rows are populated in the originating table.

Declarative Implementation

Note: The multiselect LOV can be enabled only for tables of type **advancedTable** (`OAAdvancedTableBean`). You cannot use this feature with simple tables (`OATableBean`).

Step 1: Create your advanced table region as described in the "Advanced Tables" document.

Step 2: Create a reusable list of values region as described in the Text Field LOV section above.

Tip: Make sure your LOV's view object query includes all the values that you need to correctly populate the advanced table rows. For example, if your table requires values for items A, B, C, D and E, your LOV should include corresponding items for all of these, even if it only displays A.

Warning: Do NOT base your LOV and advanced table on the same view object.

Step 3: Select the **advancedTable** region in the OA Extension structure pane, right-click and select **New > tableActions**. This automatically creates a **flowLayout** region under the **tableActions** node.

Step 4: Select the **flowLayout** region, and specify a standards-compliant `ID`. Then, right-click and select **New > lovActionButton** to create the special LOV action button. Set the following properties for the **LovAction**:

- Specify a standards-compliant `ID`.
- Set the `External LOV` property to the fully qualified name of the reusable LOV you created in Step 2 above.
- Step 4: Specify an appropriate `Prompt` ("Add <*Objects*>") for the lovAction.

At runtime, the OA Framework creates an `oracle.apps.fnd.framework.webui.beans.nav.OALovActionButtonBean` for this component.

Step 5: Create LOV mappings between the Multiselect LOV, and the base table.

- Change the `ID` of the default mapping that OA Extension created for you to an easily identified value. Also, set the `Lov Region Item` and `Return Item` (in the table) to identify the relationship between the LOV item value and the destination item in the table that will be updated with this value when the user makes a selection.
- For additional mappings, select the **lovActionMappings** node in the OA Extension structure pane, right-click and select **New > lovActionMap**. Repeat the mapping configuration described above for each one.

Step 6: Before the user can add rows to your table with the Multiselect LOV, you must initialize the associated view object as described in View Objects in Detail: Initialization Guidelines.

Runtime Implementation

Behind the scenes, selecting rows from the LOV behaves just as the Add Another Row button does. In other words, the OA Framework will create and insert rows into the view object associated with your advanced table. Note that, although your code may throw validation errors during this process (if, for example, underlying entity object validation logic fails), the OA Framework will ignore these errors so that incomplete rows can be presented to the user. Any subsequent form submit actions on the page that do not explicitly disable client and server-side validation will trigger full validation and error display.

Note: If the attempt to insert a new row with the LOV values fails due to a primary key constraint violation, the OA Framework stops processing. In other words, if the user selects five rows from the LOV and an attempt to insert the first one fails, the OA Framework does not try to process the remaining four rows.

Personalization Considerations

- None

Known Issues

- See a summary of key LOV issues with suggested workarounds if available.

Related Information

- BLAF UI Guidelines
  - LOV (List of Values)
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageLovInputBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageLovChoiceBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAListOfValuesBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OAListOfValuesBean`
- OA Framework Developer's Guide
  - Implementing the View: Creating a Shared Region
  - OA Framework View Coding Standards
  - OA Framework File Standards
  - Classic Tables
  - Advanced Tables
  - Dynamic User Interface (PPR)
- ToolBox Tutorial / Sample Library

    - ToolBox Tutorial Search Lab
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.ListOfValuesPG`
- OA Framework Component Reference (Oracle JDeveloper 10*g* Online Help)
    - OA Item Styles > messageLovInput
    - OA Region Styles > listOfValue

List Search

List Search is a sophisticated search tool that enables users to retrieve information quickly and easily. Introduced in Release 12.2.9 and extended and refined in Release 12.2.10, it allows users to search objects or records using filters. Numeorous options allow customization of the filters and other features to meet a user's specific search needs.

Contents

This section describes how to set up and use List Search, and includes the following topics:

- Implementation Overview
- Enabling List Search
- List Search Elements
  - Search Filters (Filter Panel)
  - Saved Searches Choice List
  - Save Button and Create Saved Search Popup
  - Reset Button
  - Add Filters
  - Remove Filters
  - Programmatic Control of Filters
  - Manage Saved Search Popup
  - Resultant Table Element
  - Match All or Match Any Radio Buttons
- Filter Enhancements
  - Toggle Buttons
  - Multiple Selection
  - Range Selection
  - Multiple Selection
  - Descriptive Flexfields
- Using Saved Search Notifications
- Enabling Quick Search
- Enabling Infotile View of Saved Searches
- Enabling Keyword Search
- Enabling KFF Search
- Dependent Criteria in Advanced Search
- Use Cases for Developer Uptake of Infotiles
- Debugging List Search
  - Table Diagnostics
  - Diagnostic Console
- Event and Controller Flow Comparison Between List Search and Query Bean
- Related Information

Implementation Overview

Oracle Application Framework implements List Search on top of the existing Query Bean component.

When enabling the List Search feature on a specific page or application, Oracle Application Framework migrates or merges the Query Bean panels (Simple, Advanced, and Views) into a two-panel layout: the left panel is the filter panel with the search criteria, while the right panel holds the resultant table with the returned records.

A sort can be saved as an initial sort from the advanced sort popup, when the table is under a list search region.

Enabling List Search

Enabling List Search requires both a profile option to be set and a query bean property to be turned on:

- Set the profile option FND_ENABLE_LIST_SEARCH to 'True'.
  - This profile can be enabled at all levels: Site, Application, Responsibility, and User.
- Turn on the property 'Enable List Search' for a specific query bean.

Two additional List Search options and their setup are discussed in the sections Enabling Quick Search and Enabling Keyword Search.

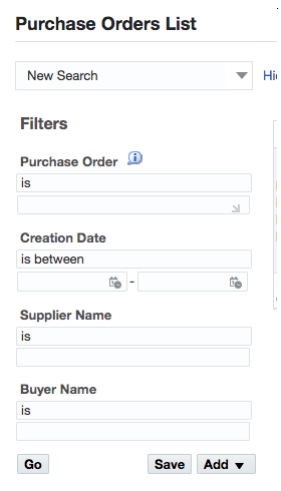
List Search Elements

Search Filters (Filter Panel)

Search filters comprise the filters that are available for end users to filter their records. Search filters appear while performing a new search, as well while selecting a user-saved search or seeded-saved search. A powerful feature of List Search is the capability to view all the filters and filter values for user-saved or seeded- saved searches. Users can add further search filters to refine the results.

The following screenshot shows the Filters area in List Search.

Search Filters



A query bean in an OA Framework page might have one of the following configurations:

Simple Panel	Advanced Panel	Views Panel
Y	Y	Y
Y	Y	N
Y	N	Y
Y	N	N

N	Y	Y
N	Y	N
N	N	Y

As the nature of the filter panel is similar to the advanced search panel in an OA Framework query bean, OA Framework takes advanced search panel contents by default for building the filters area. When a specific page has both advanced and simple search panels, the criteria are not ordered in the advanced search in the same way as they are in the simple search. In such cases, if advanced search metadata is used for building the filter area, important or frequently used criteria are placed under 'Add Menu' in the Filters area of List Search.

To avoid developers re-ordering the meta data (criteria items) of the advanced search panel, Oracle Application Framework autmatically reorders the advanced search panel contents by using the simple search panel as input. This way, the important criteria in the simple search panel appear at the top of the list in the Filters panel of List Search.

However, for automatic reordering to occur, the specific filter must be present in both simple search panel and advanced search panel. The filter should have mappings to the resultant table in both and advanced search mappings, as they map to the same resultant table element. Product developers who wish to use their own order of filters in List Search, and have no interest in the older query bean order, can turn off the query bean property 'Enable Auto Reordering Of Criteria Items'.

Query Bean Metadata Property Name	Default Value	Available Values	Admin Personalization Enabled	Description
Enable Auto Reordering Of Criteria Items	Null	True/False	Yes	• When the value is null, if the query bean has simple, advanced panels and the default panel is 'views/simple', then to render the filters area in the List Search, the criteria will be automatically reordered. • A value of False will be honored only if the query bean has simple, advanced panels, and the default panel is 'views/simple'.

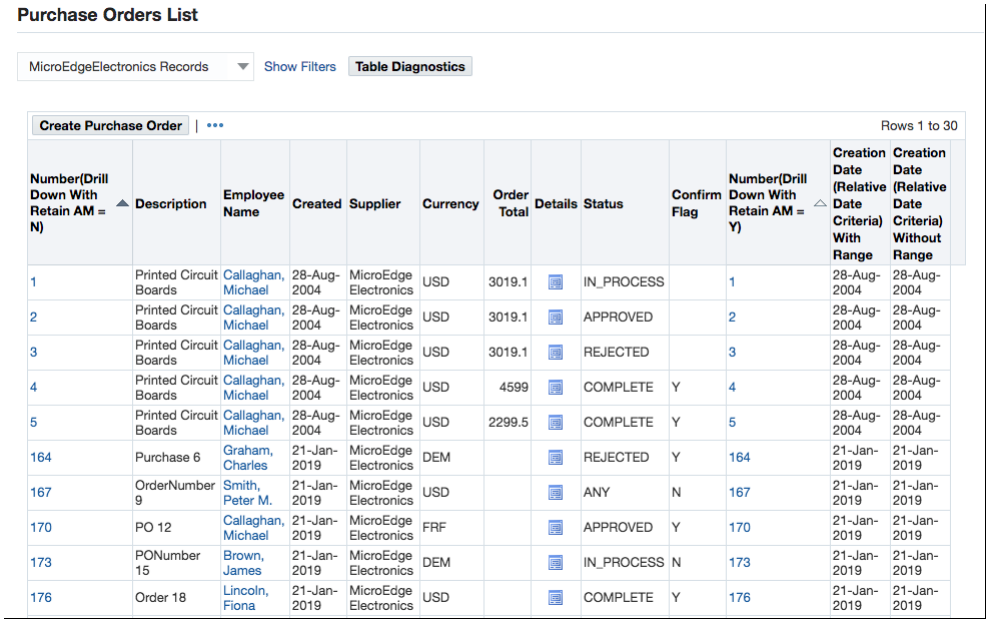
The following table shows which metadata is used for building the filters area, and when auto ordering of criteria items will take place.

Simple Panel Present	Advanced Panel Present	Views Panel Present	Default Panel For Rendering (For Automatic Ordering Feature)	Metadata Going To Filter Area	Automatic Ordering of Criteria Items Enabled
Y	Y	Y	Simple or Views	Advanced Panel	Y (Enabled by default)
Y	Y	N	Simple	Advanced Panel	Y (Enabled by default)
Y	N	Y	N/A	Simple Panel	N
Y	N	N	N/A	Simple Panel	N
N	Y	Y	N/A	Advanced Panel	N
N	Y	N	N/A	Advanced Panel	N
N	N	Y	N/A	Table	N

When 'New Search Option' is selected, the filter panel is opened by default to enable the user to filter records. When a saved search is selected, the filters are hidden initially, and then the state of the filter panel (open or closed) is persisted at the user's session level.

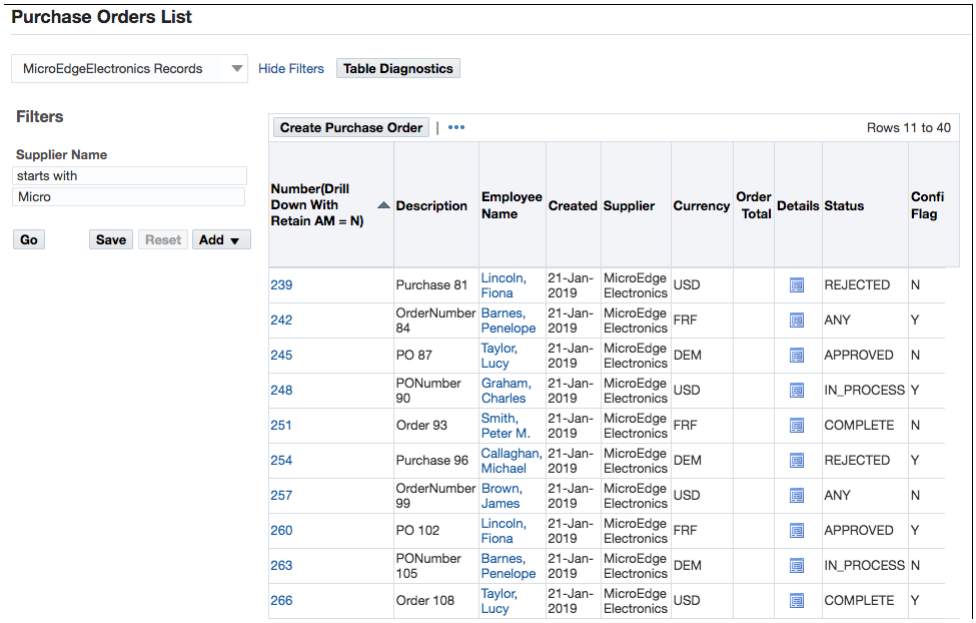
Users can always show or hide the filters panel according to their preferences.

The following screenshot shows the List Search user interface with filter panel hidden.



Number(Drill Down With Retain AM = N)	Description	Employee Name	Created	Supplier	Currency	Order Total	Details	Status	Confirm Flag	Number(Drill Down With Retain AM = Y)	Creation Date (Relative Date Criteria) With Range	Creation Date (Relative Date Criteria) Without Range
1	Printed Circuit Boards	Callaghan, Michael	28-Aug-2004	MicroEdge Electronics	USD	3019.1		IN_PROCESS		1	28-Aug-2004	28-Aug-2004
2	Printed Circuit Boards	Callaghan, Michael	28-Aug-2004	MicroEdge Electronics	USD	3019.1		APPROVED		2	28-Aug-2004	28-Aug-2004
3	Printed Circuit Boards	Callaghan, Michael	28-Aug-2004	MicroEdge Electronics	USD	3019.1		REJECTED		3	28-Aug-2004	28-Aug-2004
4	Printed Circuit Boards	Callaghan, Michael	28-Aug-2004	MicroEdge Electronics	USD	4599		COMPLETE	Y	4	28-Aug-2004	28-Aug-2004
5	Printed Circuit Boards	Callaghan, Michael	28-Aug-2004	MicroEdge Electronics	USD	2299.5		COMPLETE	Y	5	28-Aug-2004	28-Aug-2004
164	Purchase 6	Graham, Charles	21-Jan-2019	MicroEdge Electronics	DEM			REJECTED	Y	164	21-Jan-2019	21-Jan-2019
167	OrderNumber 9	Smith, Peter M.	21-Jan-2019	MicroEdge Electronics	USD			ANY	N	167	21-Jan-2019	21-Jan-2019
170	PO 12	Callaghan, Michael	21-Jan-2019	MicroEdge Electronics	FRF			APPROVED	Y	170	21-Jan-2019	21-Jan-2019
173	PONumber 15	Brown, James	21-Jan-2019	MicroEdge Electronics	DEM			IN_PROCESS	N	173	21-Jan-2019	21-Jan-2019
176	Order 18	Lincoln, Fiona	21-Jan-2019	MicroEdge Electronics	USD			COMPLETE	Y	176	21-Jan-2019	21-Jan-2019

The following screenshot shows the List Search user interface with filter panel displayed.



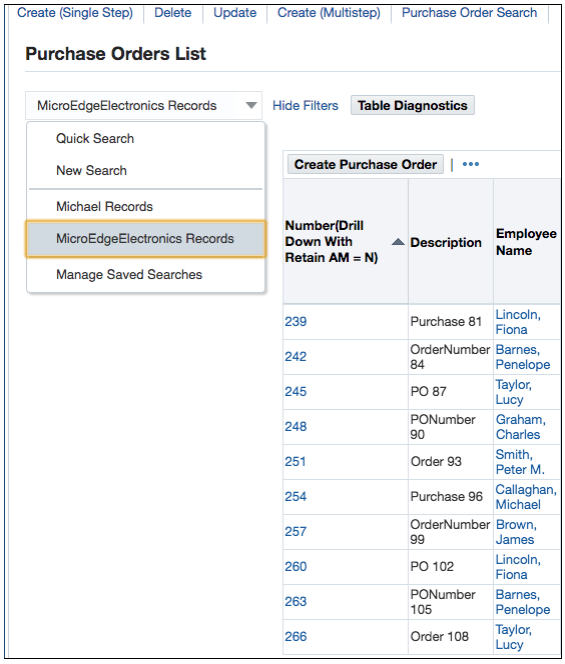
Saved Searches Choice List

The saved searches choice list contains the list of saved searches that are created for the query bean. This will list all seeded views and user views. Existing customer user and seeded views are also listed in this choice menu.

Further there are options to start a new search with filters and to manage the existing the saved searches via a Manage option

The following screenshot shows the saved searches choice list in List Search mode:

Saved Searches Choice List

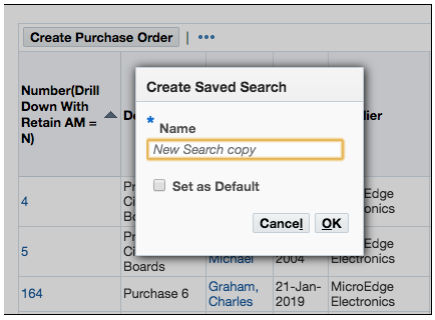


Save Button and Create Saved Search Popup

Users can filter their records by entering search filters and then clicking on the 'Save' button to create a new saved search. The 'create Saved Search' popup launches when the 'Save' button is clicked. The user can then save it with a preferred name, or simply use it as a default saved search to run on returning to the page.

The following screen shot shows the Create Saved Search popup region after the 'Save' button is clicked.

Saved Search Popup Region



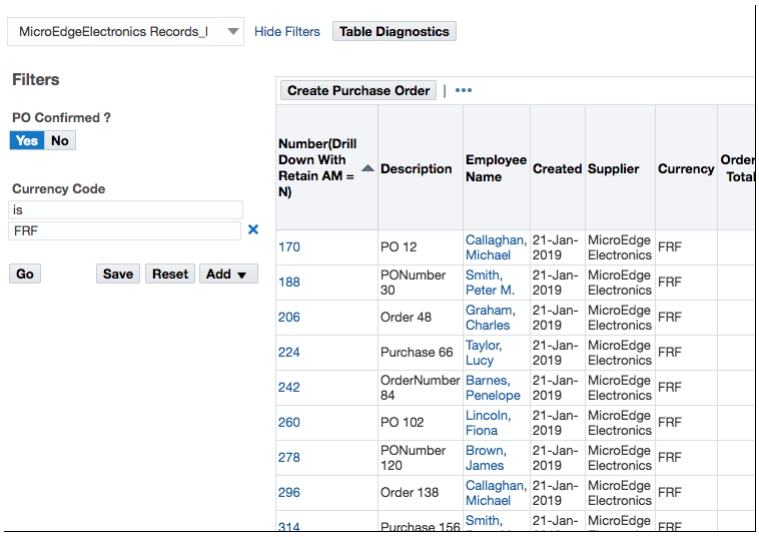
Note: The 'Save' button is unavailable if it was disabled in the query bean panels.

Reset Button

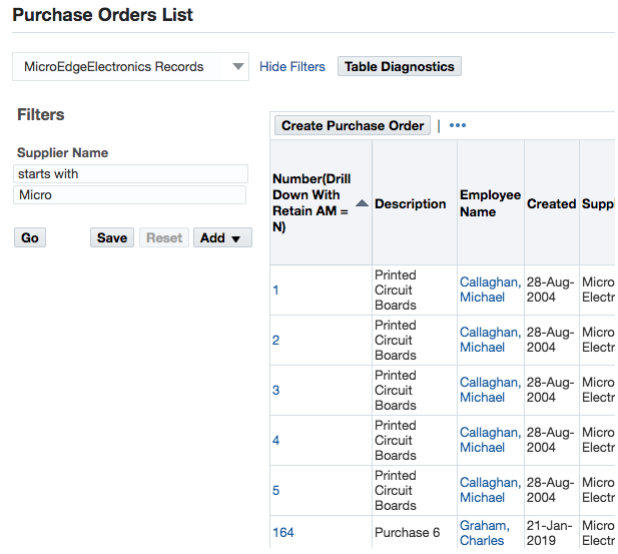
The Reset button is useful when working on user-saved searches. When a user-saved search is selected, the filters area is populated with filters related to that saved search. Users can add additional filters to refine the search. If users want to return to their original saved search filters after multiple levels of refining, they can click the Reset button. The filters will then return to the initial state when the saved search was selected.

Note: The Reset button will not render when either the New Search option or Quick Search option is selected.

The following screenshot shows the Reset button is enabled after the user refined the results in a saved search.



The following screenshot shows the result of pressing the Reset button.



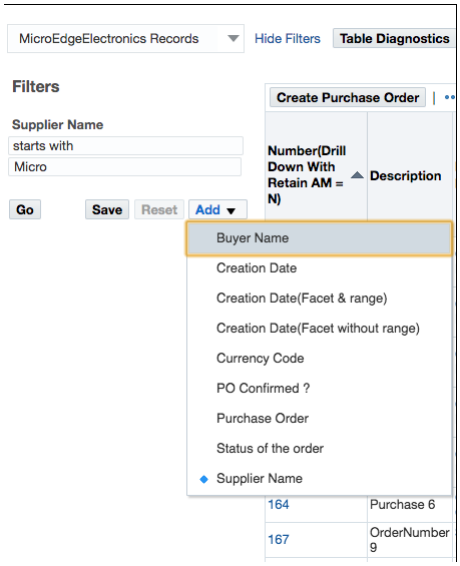
Add Filters

The Add button allows the user to add additional criteria that can be used for filtering records in the resultant table. This is similar to the add option in the query bean advanced search panel.

- A blue diamond indicator before a criterion indicates whether the criterion is already present in the Filters section of Llist Search.
- The add filter event performs a Partial Page Rendering on the page and refreshes the query bean region.

The following screenshot shows the Add button and its menu, with the criterion presence indicator for the criteria that are present in the filters area of List Search.

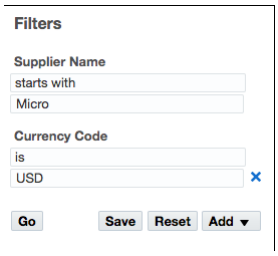
Saved Search Add Button



Remove Filters

A remove filter delete icon appears next to the criteria if additional criteria were added to the filter area via the Add menu. Users can remove these filters and click 'Go' to refine the search.

Saved Search Remove Filters



Programmatic Control of Filters

A new API has been introduced in `OAAdvancedSearchBean.isAddOrRemoveItemEvent`. Its purpose is to facilitate developers checking the add or remove filter event during processFormRequest processing.

The following code sample will illustrate the usage:

```
OAAdvancedSearchBean advSrch = (OAAdvancedSearchBean)
webBean.findChildRecursive(region6);
if(OAAdvancedSearchBean.isAddOrRemoveItemEvent(pageContext, advSrch))
{
int childCount = advSrch.getDisplayedCriteriaCount();
for (int i = 0; i < childCount; i++)
{
OAWebBean child = (OAWebBean) advSrch.findChildRecursive(Value_ + i);
String originalName = advSrch.getOriginalUINodeName(child.getUINodeName());
if (item8.equals(originalName))
{
if(((OAWebBeanFormElement) child).getValue(pageContext) == null)
{
((OAWebBeanFormElement) child).setValue(pageContext, Default Supplier Name);
}
}
}
```

Manage Saved Search Popup

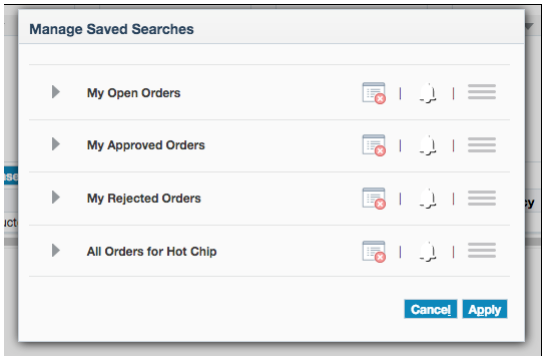
Users can manage existing saved searches via a 'Manage Saved Search' popup. This Manage Saved Search popup is launched from the Saved Searches choice list by clicking the 'Manage' option.

When clicked, the popup shows a list of saved searches and associated edit options for the saved search.

Users can take various actions on saved searches, including:

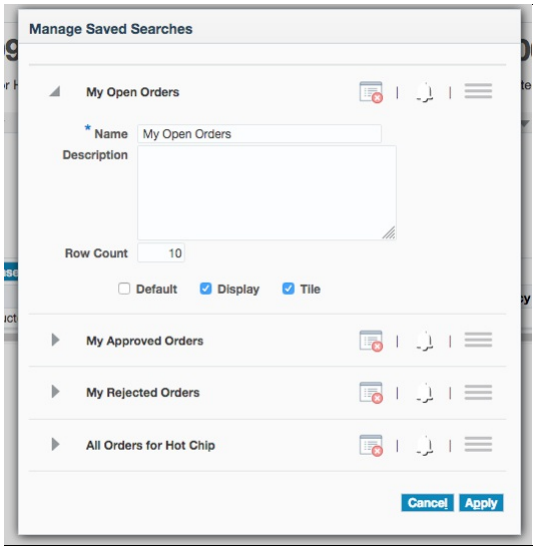
- Rename
- Delete
- Hide
- Set Default Options
- Show
- Update

Manage Saved Searches Popup



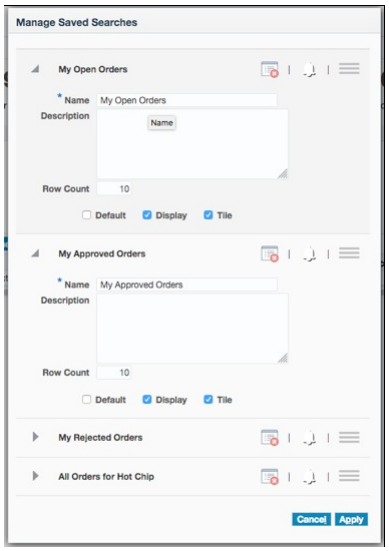
The following screenshot shows the Saved Search Detail Edit - Expand the Saved Search item:

Saved Search Detail Edit



Enable/Disable Infotile View on Seeded Save Search

The following screenshot shows the Expand Multiple Saved Searches For View or Edit in Managed Saved Searches:



Reorder Mode

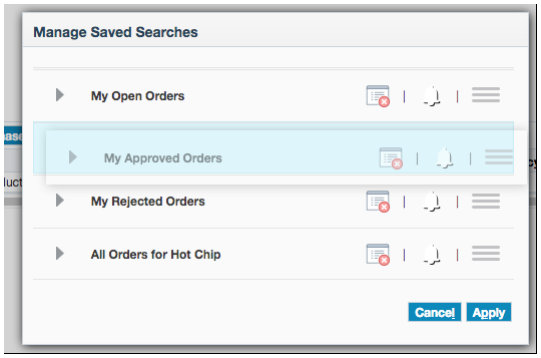
The following screenshot shows *Reorder Mode,* where the user is only able to move a saved search up or down by dragging and dropping.

From Release 12.2.10, the user has control over the display order of saved searches, as follows:

- All saved searches can be reordered, including user and seeded saved searches.
- The saved search order will be reflected in the following three locations:
  - Saved Search Drop Down Menu in List Search UI.
  - Manage Saved Search Popup.
  - Saved search's Infotile when infotile display is enabled.
- A simplified UI provides a drag handle to allow the user to drag and drop a saved search to the desired order.
  - Clicking on the *Apply* button causes the new order to take effect.
- If a new saved search is added after a reorder has been performed, the newly created saved search will appear at the bottom of the list.

When reorder is complete, click on 'Done' or 'Reset' to exit reorder mode:

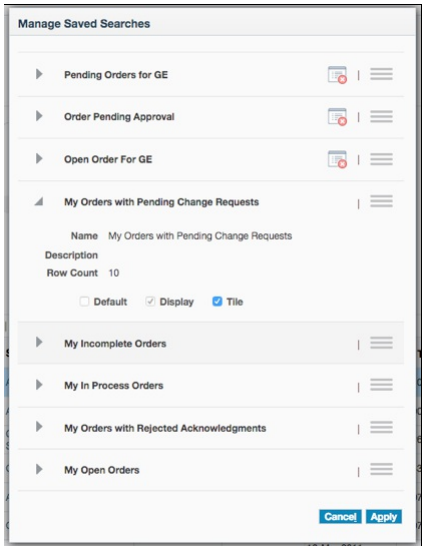
Reorder Mode in Saved Search



Enabling and Disabling Infotile View

The following screenshot shows the Enable/Disable Infotile View on Seeded Saved Search. When Infotile display is enabled on List Search, by default Oracle Application Framework will display the Infotile for all seeded saved searches. Prior to 12.2.10, the user could hide the infotile for user-saved searches, but not seeded saved searches. In Release 12.2.10, checking or unchecking the 'Tile' checkbox allows the user to toggle the display for infotile on seeded views.

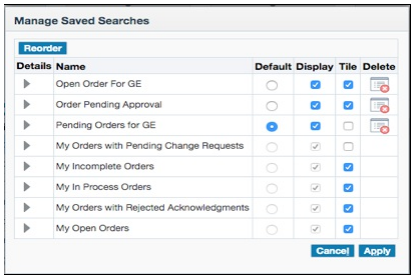
Enable/Disable Infotile View



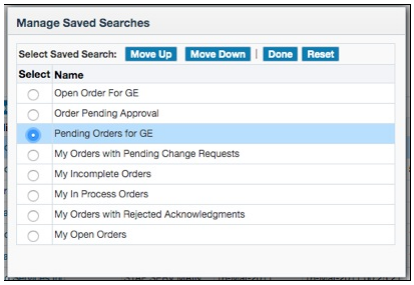
Accessibility Mode

The following screenshot shows *Accessibility Mode*. When this mode is enabled, the manage saved search UI will fall back to table display, with a detailed expansion that contains editable fields for saved searches.

Accessibility Mode



In addition, a 'Reorder' button is rendered at the top. On clicking this button, the manage saved search UI enters reorder mode. In reorder mode, user is only able to move a saved search up or down. When re-ordering is complete, the user can click on 'Done' or 'Reset' to exit reorder mode.



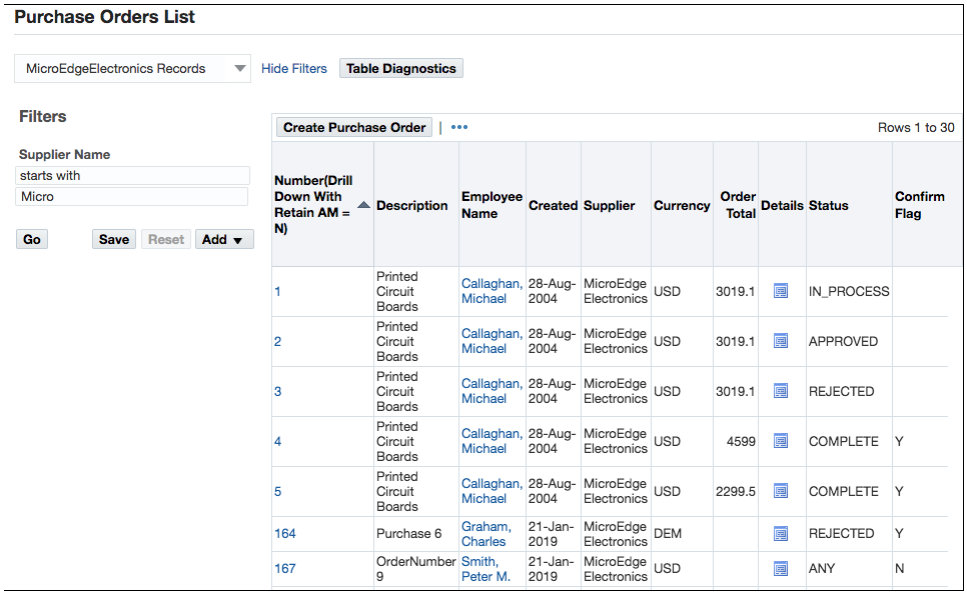
Resultant Table Element

With a two-panel layout user interface in the List Search pattern, the right-side panel shows the resultant table with records that match the criteria in the left-side panel. The resultant Oracle Application Framework classic or advanced table region will be rendered in this panel.

All rich table interactions are supported for the resultant table in List Search mode.

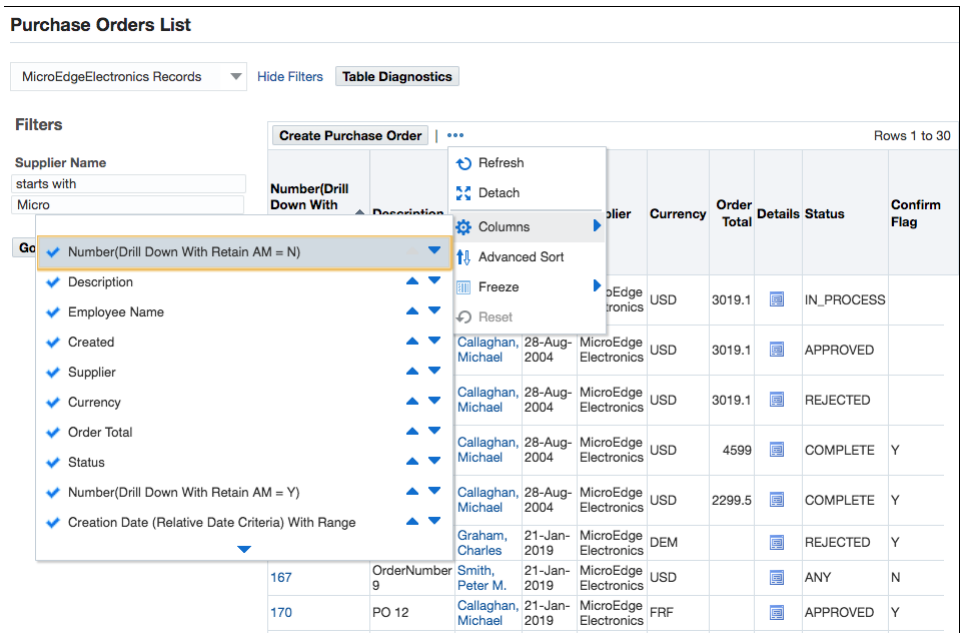
The following snapshot shows the filters area in the left side and the resultant table in the right side:

Filters and Resultant Table



The following snapshot shows rich table interactions enabled in the resultant table:

Rich Table Interactions Enabled in Resultant Table



Match All or Match Any Radio Buttons

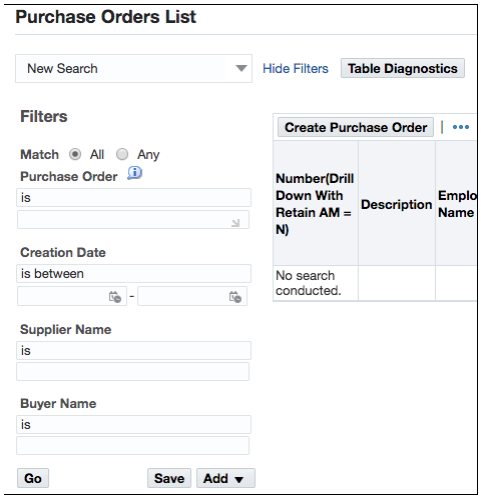
The default match condition in List Search is Match All.

There is no mention about this match All or Any condition in the List Search Cloud GPS Pattern. As OA Framework migrates from existing query bean to List Search, this option will still be enabled for certain cases:

- **Case 1**: When the default panel in the query bean is the advanced search panel, Show All or any radio button will be visible.
- **Case 2**: If the user view that was created earlier (that is, migrated from older query bean mode to new List Search mode) has an 'Or (Match Any)' based filter, 'Match All' or 'Match Any' will be displayed.

The following screen shot shows the rendering of the Match All / Match Any radio button in the filter area of List Search:

Match All / Match Any Radio Button in Filter Area



If a product developer has no requirement for displaying the Match All / Match Any radio button, or for some reason has a need to display the Match All / Match Any radio button even though both cases are not true, this can be done using the query bean property 'Hide Match Condition'.

Query Bean Metadata Property Name	Default Value	Available Values	Admin Personalization Enabled	Description
Hide Match Condition	Null	True/False	Yes	In List Search Mode: • If the attribute is null, if default panel is advanced then the match conditions will be shown. • If the attribute is True, then match conditions are hidden. • If the attribute is False, then match conditions are shown. If the user's saved search has OR conditions, then the value of the above attribute is ignored and the match conditions will be shown.

Filter Enhancements

By default, Oracle Application Framework supports the following UI web beans in the Advanced Search panel:

- OAMessageTextInputBean (Varchar and date datatypes)
- OAMessageChoiceBean
- OAMessageLovInputBean (Validated and non validated)

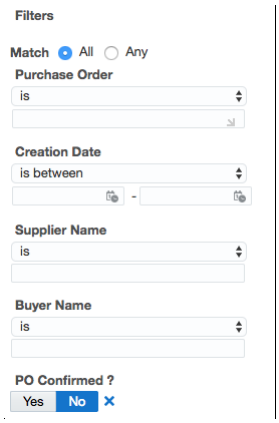
The following enhancements either introduce a new UI web bean or improve the feature of an existing UI web bean.

Toggle Buttons (OAMessageBoxCheckBoxBean)

The OAMessageBoxCheckBoxBean has been enhanced to render as a toggle button. This is particularly useful in the case of Yes / No filters.

The following screenshot shows the rendering of the checkbox toggle button filter in List Search mode:

Checkbox Toggle Button Filter



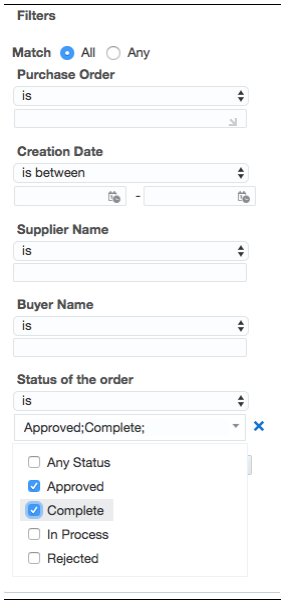
Multiple Selection

This feature allows the user to select multiple values for criteria of type OAMessageChoiceBean.

For a filter of type OAMessageChoiceBean, the supported UI conditions are 'is', 'is not', and 'is not (include blanks'. The supported data types for this case are VARCHAR and NUMBER.

The following screenshot shows the enablement of multiple selection for choice-based criteria.

Multiple Selection for Choice-Based Criteria



Metadata Changes

Choice Bean Property	Default Value	Default Value2	Admin Personalization Enabled	Description
Enable Multiple Selection	Null	True/False	No	• When the value is Null, multiple selection is disabled. • When the value is True, a multi-select choice will appear for the user to pick up multiple values.

For example:

```
<oa:criteriaRow id=POStatusCrit>
  <ui:contents>
    <oa:messageStyledText id=item13 prompt=Status of the order/>
    <oa:messageChoice id=item14 pickListViewName=StatusListVO1 pickListDispAttr=Meaning
     pickListValAttr=LookupCode enableMultipleSelection=true/>
  </ui:contents>
</oa:criteriaRow>
<oa:queryCriteriaMap id=queryCriteriaMap10 criteriaItem=item14 resultsItem=item15/>
<oa:messageStyledText id=item15 viewAttr=StatusCode userCustomizable=true enableMultipleSelection=true/>

<oa:column id=CreationDateCol>
<ui:columnHeader>
<oa:sortableHeader id=CreationDateHdr use=/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxPoHeaders/CreationDate/>
</ui:columnHeader>
<ui:contents>
<oa:messageStyledText use=/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxPoHeaders/CreationDate viewAttr=CreationDate id=CreationDate
promptTranslationExpansion=none dataType=DATE viewName=PoSimpleSummaryVO1 userCustomizable=true enableRangeSelection=true/>

</ui:contents>
</oa:column>
```

Delimiter Support

The default filter that OA Framework uses for separating multi-select choice options is the semicolon (;) character.

Product developers who have employed a lookup code that contains the semicolon character, or who simply want to use a different delimiter for multi-select options than the semicolon character, can set the following property.

Choice Bean Property	Default Value	Available Values	Admin Personalization Enabled	Description
MultiSelect Options Delimiter	NULL	Free flow string	No	• Overrides the default delimiter (semicolon) for a multi- selection choice list

Note: Once a delimiter is chosen, all views in MDS will have the new delimiter. Consequently, changing the delimiter at a later point will invalidate these MDS views. Therefore, consider carefully when choosing a delimiter.

Migration

Value in View	Condition in View	Value Selected in Multi-Select UI	Condition in UI	Query Executed in Multi-Select UI	Query Executed in Single-Select UI (Classic UI)
20	IS NOT (include blanks)	20	IS NOT (include blanks)	((<>20) OR column name IS NULL)	((<>20) OR column name IS NULL
30	IS NOT	30	IS NOT	column_name NOT IN ('30')	<>30
20	IS	20	IS	column_name IN ('20')	= 20

Handling Criteria in the Controller

Developers who handle criteria by calling query.getCriteria() must follow the steps below to ensure the criteria are correctly applied on the view objects. Developers who are not handling criteria do not need to follow the steps below.

Product developers need to look out for the following keys in the criteria dictionary:

- CRITERIA_MULTIPLE_OPTIONS - Boolean. TRUE.
- CRITERIA_MULTIPLE_OPTIONS_DELIMITER - String - If this is null, split by the default seperator. MULTI_CHOICE_SELECTION_DEFAULT_SEPERATOR
- CRITERIA_VALUE - Value in the UI separated by CRITERIA_MULTIPLE_OPTIONS_DELIMITER.

The rest of the dictionary values will be returned as per current behavior.

SQL query to be used :

- `IS Use column_name IN ('val1','val2',.......)`
- `IS NOT Use column_name NOT IN ('val1','val2',.....)`
- `IS NOT (INCLUDE BLANKS) Use ( column_name <> :1 OR column_name IS NULL )`

Range Selection

This feature allows the date and number data type message text input to use a between operator. It applies to number and date data types, OAMessageChoiceBean, OAMessageTextInputBean, and OAMessageLovInputBean, and enables the user to filter on date ranges instead of single date criteria.

Range Selection Recommendations

Oracle Application Framework knows how to handle the criteria when range selection is enabled, when range values are saved in the view, how to retrieve them, and how to build a SQL where clause out of this criteria. We therefore strongly recommend that developers should let OA Framework build the where clause for these types of criteria rather than building their own where clauses for range selection enabled filters. If you still prefer to handle the criteria, be aware of the following dictionary parameters when you build your where clause snippet.

- OAViewObject.CRITERIA_RANGE_OPERATION_ENABLED - Boolean.TRUE
- OAViewObject.CRITERIA_RANGE_END_VALUE - 2nd value for the range operations, if the operator is IS, IS NOT, IS NOT(INCLUDE BLANKS) then the value will be null or empty string.
- OAViewObject.CRITERIA_VALUE - 1st value - value is present for all operations BETWEEN, IS , IS NOT, IS NOT(INCLUDE BLANKS)
- OAViewObject.CRITERIA_CONDITION -
- OACommonUtils.BETWEEN if between was selected in the UI.
- OACommonUtils.AIS if is was selected in the UI.
- OACommonUtils.BNOT if is not was selected in the UI.
- OACommonUtils.JNULL if is not (include blanks) was selected in the UI.
- OACommonUtils.HAFTER From the view if only first value is entered and the second value is not entered.
- OACommonUtils.FGREATE For number data types - during upgrade from single selection to Range selection.
- OACommonUtils.IBEFORE From the view if only second value is entered and the first value is not entered.
- OACommonUtils.GLESS For number data types - during upgrade from single selection to range selection.

OAViewObject.CRITERIA_LOOKUP_NAME	Meaning	CRITERIA_CONDITION	SQL QUery Condition
OACommonUtils.JNULL	if is not(include blanks) was selected by the user in the UI	<>	(COLUMN_NAME <> VALUE_1 OR COLUMN_NAME IS NULL))
OACommonUtils.IBEFORE or OACommonUtils.GLESS	f second value was entered and first was left blank (Date Datatype) if second value was entered and first was left blank (Number Datatype)	<	COLUMN_NAME <= VALUE_2
OACommonUtils.HAFTER or OACommonUtils.FGREATER	if first value was entered and second was left blank (Date Datatype) if first value was entered and second was left blank (Number Datatype)	>	COLUMN_NAME >= VALUE_1
OACommonUtils.BNOT	if is not was selected by the user in the UI	<>	No changes, as you handled earlier.
OACommonUtils.BETWEEN	between was selected by the user in the UI	BETWEEN	COLUMN_NAME BETWEEN VALUE_1 and VALUE_2
OACommonUtils.AIS	if is was selected by the user in the UI	=	No changes, as you handled earlier.

Note: The > and < conditions will appear even when range selection is off. So to distinguish between whether the criteria in the dictionary are range selection enabled or not, use this code: `Boolean isRangeCriteria = dic[i].get(OAViewObject.CRITERIA_RANGE_OPERATION_ENABLED);`

The constant CRITERIA_RANGE_OPERATION_ENABLED was introduced in Release 12.2.9 . If you do not want to pick a compilation dependency, you can use this code:

```
Boolean isRangeCriteria = dic[i].get("criteriaRangeOperationEnabled");
```

Range Selection Conditions

The following conditions are currently supported for searches.

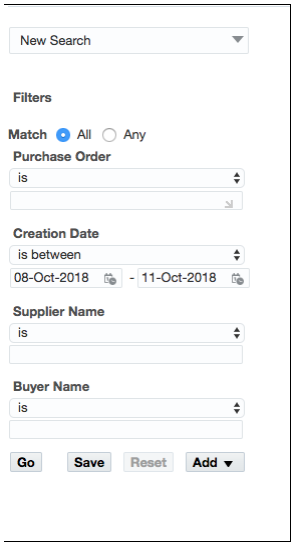
Data Type	Bean Type	Conditions Supported
NUMBER	OAMessageTextInputBean	IS IS NOT GREATER THAN LESS THAN IS NOT(include blanks)
DATE	OAMessageTextInputBean	IS IS NOT GREATER THAN LESS THAN IS NOT(include blanks)
DATETIME	OAMessageTextInputBean	IS IS NOT GREATER THAN LESS THAN IS NOT(include blanks)

Range Selection Enhancements

This enhancement alters these operators with a between operator and a more intuitive UI.

The following screenshot shows the between operator enabled for a date-type filter.

Between Operator Enabled for Date-Type Filter



Supported Beans and Data Types

Bean	Data Type
OAMessageTextInputBean	NUMBER
OAMessageDateFieldBean	DATE
OAMessageDateFieldBean	DATETIME

Metadata Changes

Choice Bean Property	Default Value	Available Values	Admin Personalization Enabled	Description
Enable Range Selection	Null	True, False	No	• When the property value is Null, the range selection is disabled and the Classifc UI displayed. • When the property value is True, a range selector will appear for the user to perform range operations.

This property needs to be set on the criterion web bean under the advanced search bean, as well on the resultant table item element that this criterion is mapped to. In some cases the resultant table item element could be a form value, and it is supported to enable the property for such form value beans.

For example:

```
<oa:criteriaRow id=POCreationDateAdvCrit>
<ui:contents>
                                    <oa:messageStyledText id=item9 prompt=Creation Date/>
                                    <oa:messageTextInput id=item10 dataType=DATE enableRangeSelection=true/>
                                 </ui:contents>
                              </oa:criteriaRow>
<oa:queryCriteriaMap id=queryCriteriaMap8 criteriaItem=item10 resultsItem=CreationDate/>
<oa:column id=CreationDateCol>
                           <ui:columnHeader>
                              <oa:sortableHeader id=CreationDateHdr use=/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxPoHeaders/CreationDate/>
                           </ui:columnHeader>
                           <ui:contents>
                              <oa:messageStyledText use=/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxPoHeaders/CreationDate viewAttr=CreationDat
                           </ui:contents>
                        </oa:column>
```

Supported Operators in Range UI

Data Type	Operators Supported
Number	IS BETWEEN (GREATER THAN and LESS THAN merged in this UI) IS NOT(include blanks) IS NOT
DATE	IS BETWEEN (BEFORE and AFTER MERGED in this UI) IS NOT(include blanks) IS NOT
DATETIME	IS BETWEEN (BEFORE and AFTER MERGED in this UI) IS NOT(include blanks) IS NOT

For ease of implementation, with range operations enabled, the operators saved in the view are: IS, BEFORE, AFTER, BETWEEN.

In NUMBER data types, BEFORE approximates to LESS THAN, and AFTER approximates to GREATER THAN.

Behavior of Between Operator

Data Type	Operator	Range	Comments
Number	BETWEEN	5 - 100	Searches for records that are greater than or equal to the first number, and less than or equal to the second number. Leaving the first value blank returns all records that are less than or equal to the second number; leaving the second value blank returns all records that are greater than or equal to the first number.

DATE	BETWEEN	01-JUN-2017 - 25-JUN-2018	Searches for records that are after or equal to the first date, and before or equal to the second date. Leaving the first value blank returns all records that are before or equal to the second date; leaving the second value blank returns all records that are after or equal to the first date.

Migration

This topic discusses the impact of user views when migrating from a classic conditional operator to a range-based operator.

Data Type	Value in View	Condition in View	Value 1 Selected in Range Filter UI	Value 2 Selected in Range Filter UI	Condition in Range Filter UI	Query Execute (While Range Selection is Enabled)	Que Ran
Number	20	IS	20	Empty	IS	= 20	= 20
Number	30	IS NOT	30	Empty	IS NOT	<> 30	<> 3
Number	30	IS NOT (include blanks)	30	Empty	IS NOT (include blanks)	((<>30) OR column IS NULL)	((<>
Number	20	GREATER THAN	20	Empty	BETWEEN	>= 20	> 20
Number	30	LESS THAN	Empty	30	BETWEEN	<= 30	< 30
Date or datetime	01/01/2017	IS	01/01/2017	Empty	IS	= 01/01/2017	= 01
Date or datetime	01/31/2017	IS NOT	01/31/2017	Empty	IS NOT	<> 01/31/2017	<> 0
Date or Datetime	01/31/2017	IS NOT (include blanks)	01/31/2017	Empty	IS NOT (include blanks)	((<>01/31/2017) OR column IS NULL)	((<> NUL
Date or datetime	01/31/2017	AFTER	01/31/2017	Empty	BETWEEN	>= 01/31/2017	> 01
Date or Datetime	01/31/2017	BEFORE	Empty	01/31/2017	BETWEEN	<= 01/31/2017	< 01

Handling Criteria in the Controller

- Developers who choose to execute the table VO query by calling `queryBean.getCriteria()` should handle the criteria values in the same way OA Framework does for the query execution if `getCriteria()` is not called.
- Developers need to look out for these keys in the criteria dictionary.

SQL Query to Use

- BETWEEN - column_name between value_1 and value_2
- HAFTER - column_name >= value_1 Same is the case with FGREATER for number data types
- IBEFORE - column_name <= value_1 Same is the case with GLESS for number data types
- AIS - no changes as earlier
- BNOT - no changes as earlier
- JNULL - no changes as earlier

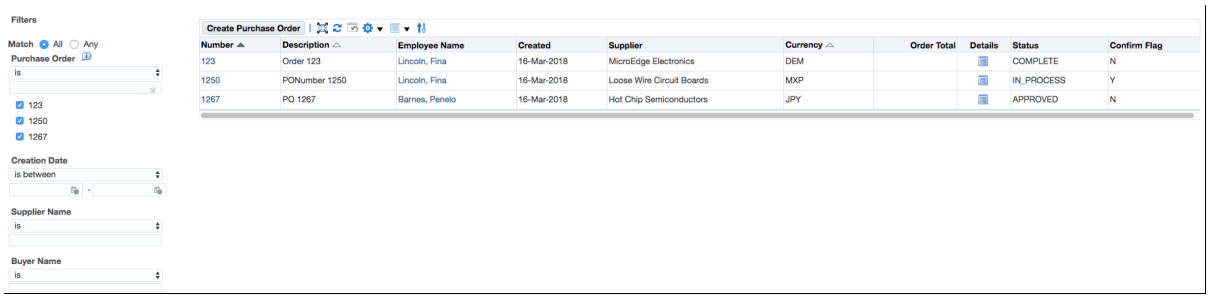
Multiple Selection (OAMessageLovInputBean)

- This feature enables the user to select multiple items from the LOV (list of values) and use it for search. This feature is supported only for the validated list of values bean. It is not supported on unvalidated LOVs.
- This feature works in a similar way to the multiple selection enabled on a message choice bean, as described earlier.
- All records matching the selected values will be filtered in the resultant table.

The following screenshot shows the multiple selection enabled lov filter in List Search mode. In the screenshot, Purchase Order is a multiple selection enabled validated LOV filter. The user can search on the Order Number, Status, Description columns of the LOV table in a single text input.

Once a match is made, or a user selects the record from the lookahead LOV, the record is added as selected value for the filter.

Users can add 'n' number of filters in this way to search multiple records: <n - configuration for multiselect LOV that the developer can set>



Metadata Changes

LOV Bean Property	Default Value	Available Values	Admin Personalization Enabled	Description
Enable Multiple Selection	NULL	True, False	No	• When the property is NULL, multiple selection is disabled. • If True, a multiselect LOV will appear for the user to pick up multiple values.

This property needs to be set on the validated LOV web bean, all resultant form value beans, and the resultant item in the table that the mapping points to.

Example:

```
<oa:criteriaRow id=POAdvCriteria>
<ui:contents>
<oa:messageStyledText id=item3 prompt=Purchase Order/>
<oa:messageLovInput id=item4 externalListOfValues=/oracle/apps/fnd/framework/toolbox/lov/webui/PurchaseOrdersLovRN unvalidated=false
enableMultipleSelection=true>
<lovMappings>
<lovMap id=lovMap1 lovItem=OrderNum resultTo=item4 criteriaFrom=item4/>
<lovMap id=lovMap3 lovItem=Description resultTo=item21/>
</lovMappings>
</oa:messageLovInput>
<oa:formValue id=item21 enableMultipleSelection=true/>
</ui:contents>
</oa:criteriaRow>
```

```
<oa:queryCriteriaMap id=queryCriteriaMap6 criteriaItem=item4 resultsItem=OrderNum/>

<oa:column id=OrderNumCol columnDataFormat=textFormat>
<ui:columnHeader>
<oa:sortableHeader id=OrderNumHdr sortState=ascending initSortSeq=first
use=/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxPoHeaders/HeaderId_Number/>
</ui:columnHeader>
<ui:contents>
<oa:messageStyledText sortState=ascending viewAttr=OrderNumber id=OrderNum initSortSeq=first promptTranslationExpansion=none dataType=NUMBER
use=/oracle/apps/fnd/framework/toolbox/attributesets/FwkTbxPoHeaders/HeaderId_Number viewName=PoSimpleSummaryVO1 destination=OA.jsp?
page=/oracle/apps/fnd/framework/toolbox/tutorial/webui/PoDetailsPG&amp;retainAM=Y&amp;addBreadCrumb=Y&amp;headerId={@OrderNumber} userCustomizable=true
enableMultipleSelection=true/>
</ui:contents>
</oa:column>
```

Migration

Value in View	Condition in View	Value Selected in Multiselect UI	Condition in UI	Query Executed in Multiselect UI	Query Executed in Single Select UI (Classic UI)
30	IS NOT	30	IS NOT	column_name NOT IN ('30')	<> 30
20	IS	20	IS	column_name IN ('20')	= 20
20	IS NOT (include blanks)	20	IS NOT (include blanks)	((<>20) OR column name IS NULL)	((<>20) OR column name IS NULL)

Handling Criteria in the Controller

Developers who handle the criteria by calling `query.getCriteria()` should follow the steps below to ensure the criteria are correctly applied to view objects. Be aware of these keys in the criteria dictionary.

- CRITERIA_MULTIPLE_OPTIONS - Boolean.TRUE
- CRITERIA_MULTIPLE_OPTIONS_DELIMITER - String - If this is null, split by the default seperator MULTI_CHOICE_SELECTION_DEFAULT_SEPERATOR
- CRITERIA_VALUE - value in the UI seperated by CRITERIA_MULTIPLE_OPTIONS_DELIMITER.

The rest of the dictionary values will be returned as of previous releases.

SQL Query to be used:

IS Use column_name IN (val1,val2,.....)
IS NOT Use column_name NOT IN (val1,val2,.....)
IS NOT (INCLUDE BLANKS) Use (column_name <> :1 OR column_name IS NULL)

Disable Operators (All Types of Filter)

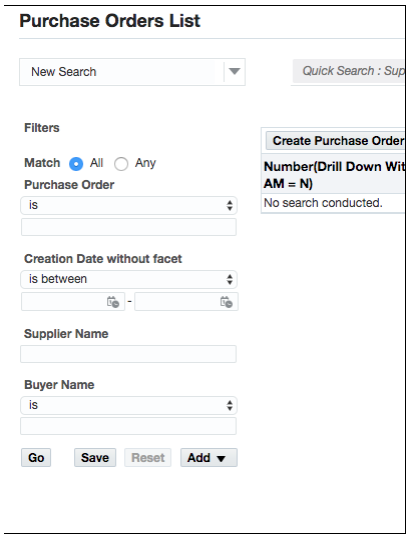
In List Search mode, if there is a requirement to disable filters at individual criteria or filter level, the Disable Operators feature is useful.

Accordingly, for each of the advanced search filters, you can set the property 'Disable Operators'. When this is set, the operators are disabled for the filter, and the operators pick list will not render. The property Disable Operators can be SPEL-enabled. When SPEL is enabled, OA Framework expects the SPEL to return Boolean.TRUE or Boolean.FALSE for the property.Â

Be aware of the following important points:

- This feature is only supported when you land in a page (process request). Do not disable operators via SPEL during a process.
- In the form request stage, OA Framework will not honor this feature.
- On the server, the operator will be defaulted by OA Framework similar to the classic query bean simple search panel. This is to prevent users from choosing unavailable operators.

This sample screen shot showing List Search without operators. The filter Supplier Name is without any operator.



APIs to Clear Criteria Values

In List Search mode, if there is a requirement to clear criteria values for any of the filters in your process form request code, the following enhancement will be helpful.

OA Framework expects the developer to send a vector of advanced search filter IDs for which the values need to be cleared.

The following API needs to be called on the query bean :

```
queryBean.clearDataObjectForFilters(Vector listOfFilterIdentifiersToClearDataObjectFor, OAPageContext pageContext)
```

Sample Code:

```
Vector list = new Vector();
list.add(item8);
list.add(item9);
queryBean.clearDataObjectForFilters(list, pageContext);
```

Controls to Prevent User From Adding Same Criteria

With the introduction of List Search, OA Framework introduced multiple selection in choice and LOV (validated) based filters. For these filters, the user cannot add the filter again through the add menu in the filter area. The same applies to business logic in product pages. Hence an API is provided to prevent users from adding the same criterion once it is available in the filters area.

The following API needs to be called on the query bean in a process request :

```
queryBean.setNonRepetitiveFilterIdentifiersList(Vector listOfFilterIdentifiersToExclude)
```

Sample Code:

```
Vector list = new Vector();
list.add(item8);
list.add(item9);
queryBean.setNonRepetitiveFilterIdentifiersList(list);
```

In this case, users will not be allowed to add item8 and item9 once they are in the filter area.

Facet and Relative Date Range Filters

The Facet feature was designed to:

- Support searching date criteria in relative to SYSDATE terms.
- Allow developers to seed relative date options (for example, the last seven days, the last week, the last month).
- Allow system administrators to personalize the relative date options and add more date options.
- Allow end users to save the relative date options in a view and later run them in a query.

Supported Beans and Data Types

Bean	Data Type
OAMessageDateFieldBean	DATE

Metadata Changes

Metadata for Date Fields

Feature	Bean	Proprty Name	Display Name	Category	Admin Personalizable	User Personalizable	Supported Values	Default Value	Description
Enable Facets for date field filter	OAMessageDateFieldBean	enableFacet	Enable Facet	Functional	Yes	No	String: • Null (Empty) • true • false	Null (Empty)	Renders the search filter in a facet mode with facet options.

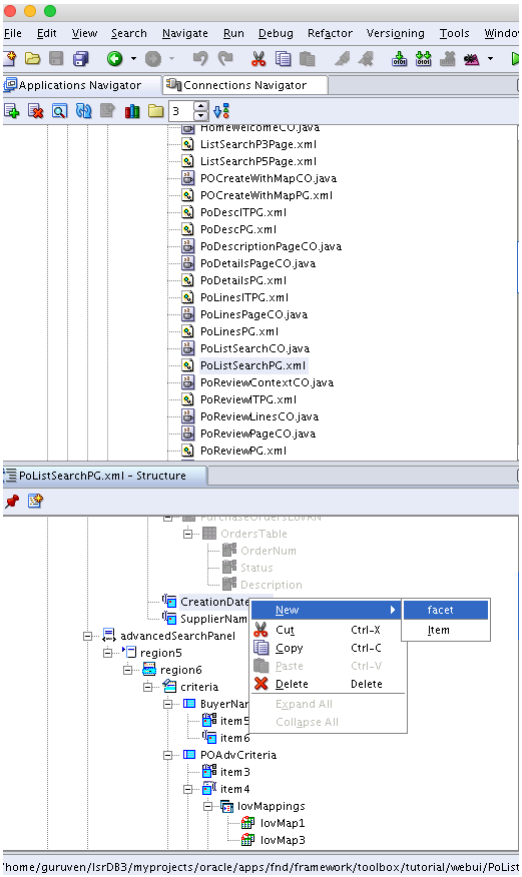
Metadata Structure for Facets

- Right-clicking the message text Input item in the property inspector displays a facet child. Right-clicking the facet child displays a facetOption child.
- Facet is the container, and facet option refers to individual options.
- Each facet option has properties as shown in the following table.

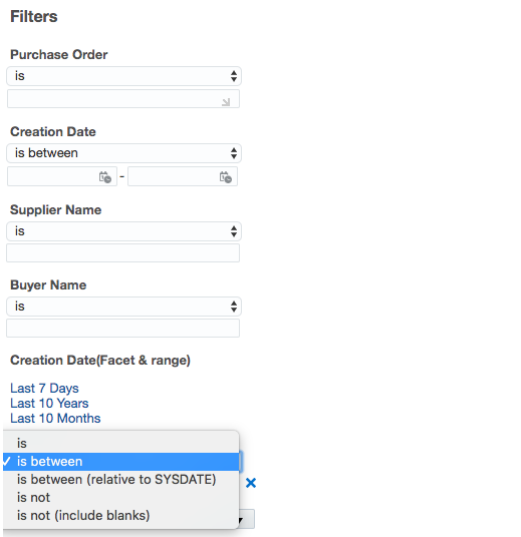
Property Name	Data Type	Allowed Values	Usage
Start Period	Integer	All Java Integer range	Only Start Period: when using only start period, all records entered after that period with no end period will be filtered. For example: • +7, all records after sysdate + 7 days • -8, all records after sysdate - 8 days • In the case of between: ◦ This is the end point.-7 to -3. Records ending point to Sysdate - 7.
End Period	Integer	All Java Integer range	Only End Period: when using only end period, all records entered before that period with no start period will be filtered. For example: • +7 , All records before sysdate + 7 days • -8, All records before sysdate - 8 days • In the case of between: ◦ This is the end point. -7 to -3. Records ending point to Sysdate - 3.
Period Format	String	D, M, Y	D - Day M - Month Y -Year
Identifier	String	Any string	Unique (within a page) metadata identifier for the option.
User Text	String	Any string	Relative options appear as links. This gives the link text that appears in the UI.

Snapshot of the Metadata

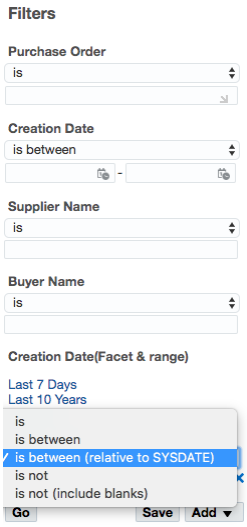
Facet child on right clicking the date field:



Facet Option Child on right clicking the date facet:



Facet Options:



Final XML Snapshot after adding facet and facet options

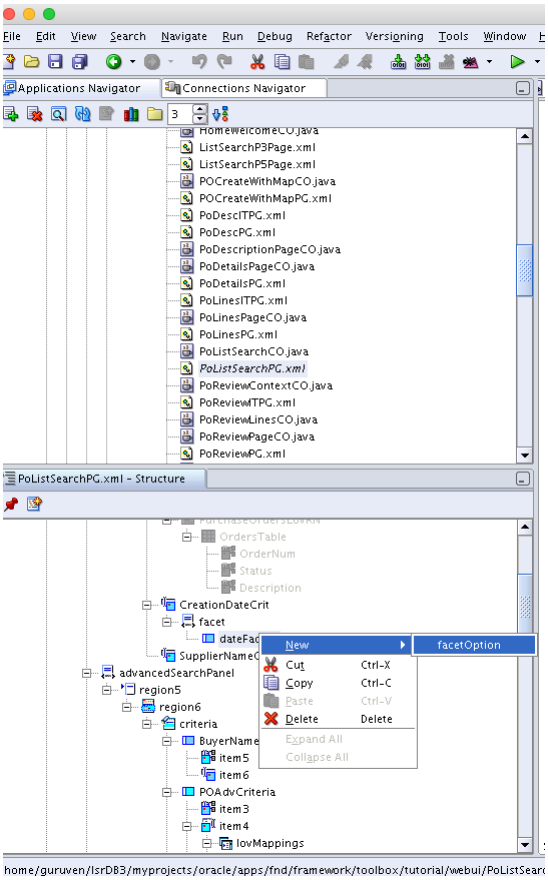
```
<oa:messageTextInput id=item23 dataType=DATE enableRangeSelection=true enableFacet=true>
 <oa:facet>
  <oa:dateFacet id=dateFacet1>
   <oa:facetOption>
    <oa:dateFacetOption id=dateFacetOption1
dateFacetOptionUserText=Last 7 Days dateFacetOptionStartPeriod=-7 dateFacetOptionEndPeriod=1/>
    <oa:dateFacetOption id=dateFacetOption4
dateFacetOptionUserText=Last 10 Years dateFacetOptionStartPeriod=-10 dateFacetOptionEndPeriod=1
dateFacetOptionPeriodFormat=Year dateFacetOptionEnabled=true/>
    <oa:dateFacetOption id=dateFacetOption5
dateFacetOptionPeriodFormat=Months dateFacetOptionUserText=Last 10 Months
dateFacetOptionStartPeriod=-10 dateFacetOptionEndPeriod=1/>
   </oa:facetOption>
  </oa:dateFacet>
 </oa:facet>
</oa:messageTextInput>
```

Range Selection and Facet options

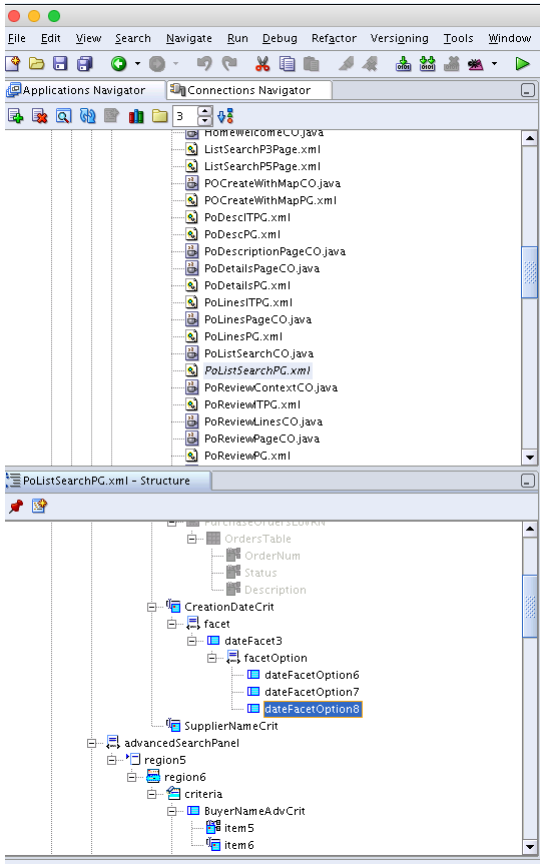
OA Framework supports facet based date web beans with both range selection enabled and range selection disabled.

See below for some example screen shots of the facet and the conditions available when the facet option is enabled Facet options for search field.

Facet options for search field:



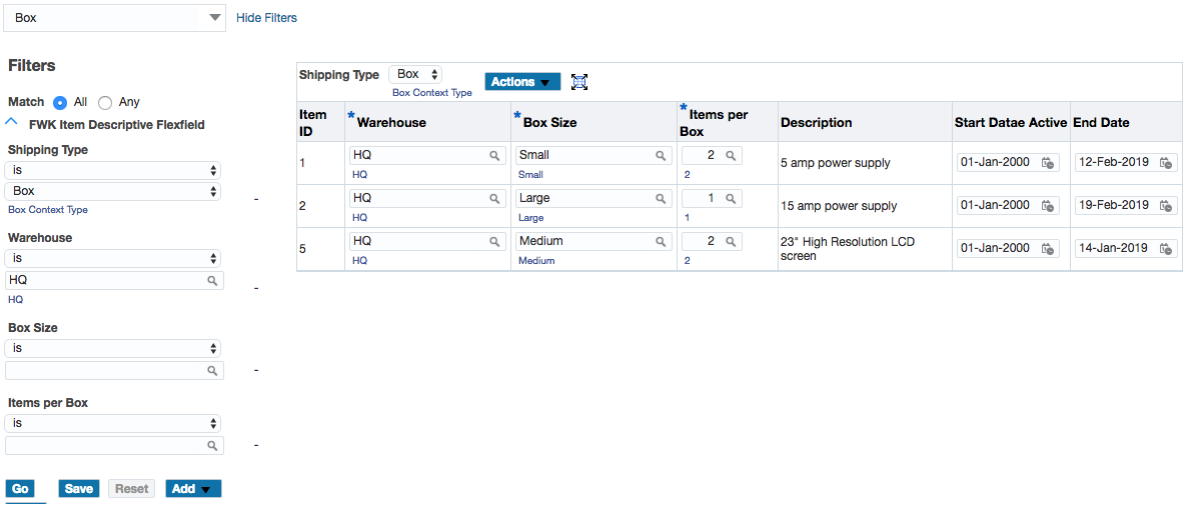
Facet based date field search conditions:



Relative to Sysdate range search option:

Descriptive Flexfields

The following screenshot illustrates the use of descriptive flexfields with List Search.



For more information, refer to the Descriptive Flexfields section in the Search chapter of this book.

Using Saved Search Notifications

1. Requirements
2. Steps to Enable Notification Feature for User
    a. List Search and Infotile Features Enablement
    b. Profile Option
    c. Set Up Mailer
    d. Set Up Email ID
    e. Activate Notification Role
3. Use Notifications
4. Sample Email Report and Bar Chart report
5. Monitoring Saved Search Notifications
6. Administrator Options
    a. Concurrent Manager Tuning
    b. Email Customization

Requirements

- Enable user to get a report of his saved search results in email based on a schedule.
- Enable user to get notified on row insert events on the saved search results.
- Enable user to get notified when record count goes beyond a set threshold for tracking purposes.
- Enable administrators to monitor all such notification requests and take action in case of issues.

Steps to Enable Notification Feature for User

There are number of steps in this process, as follows.

List Search and Infotile Features Enablement

- Search Page with List Search Enabled
- Infotile feature is turned on with List Search

Profile Option

By default the profile is on (set to 'True'). If the profile is turned off (set to 'False'), the notification feature is turned off across the application.

Name	Code	Default Value	Applicable Values
Enable Search Notification	FND_ENABLE_SAVED_SEARCH_NOTIFICATIONS	True	True, False

Set Up Mailer

You can skip this step if your Oracle E-Business Suite environment has an active SMTP server configuration.

1. Mailer Configuration: Go to System Administration > Oracle Applications Manager > Workflow > Notification Mailers
2. Once you are in the mailer config page, click on **Edit**.
3. Enter server name, user name, and password.
4. For Connection Security, ensure SSL/TLS is selected.
5. Click On **Apply**.

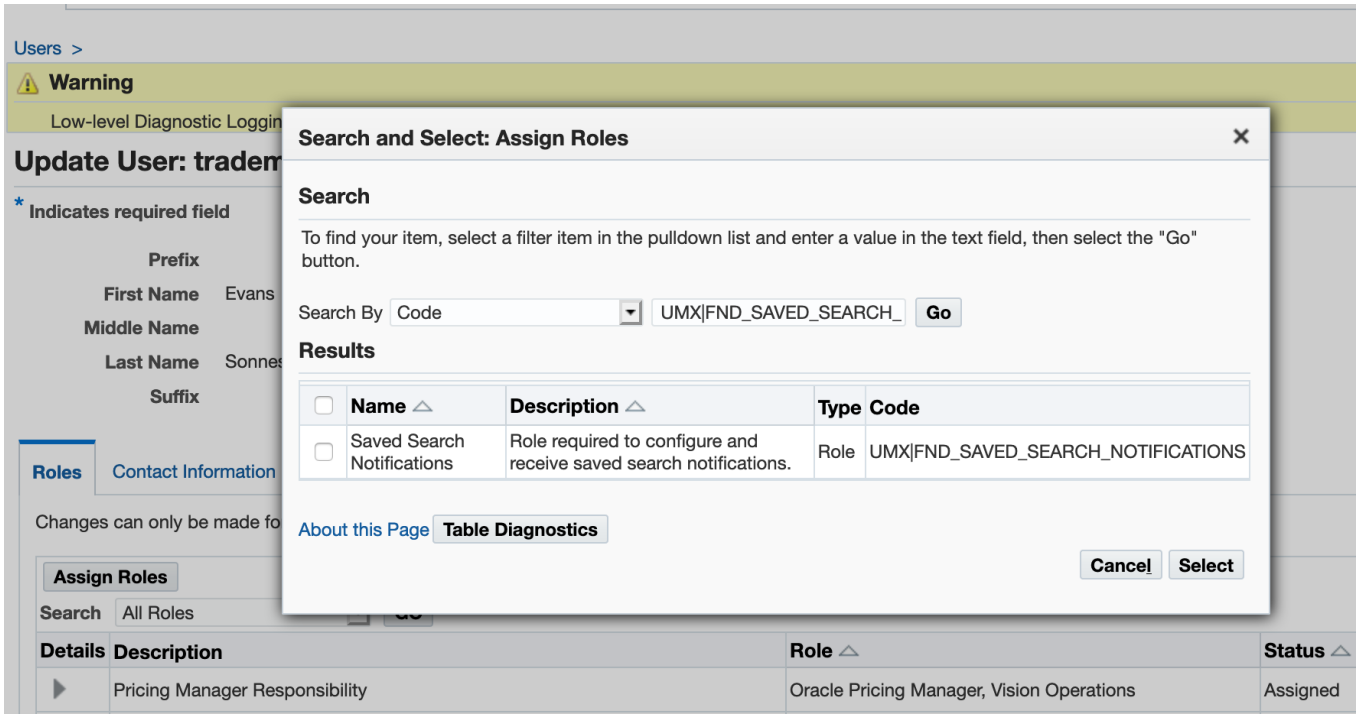
Set Up Email ID

You can skip this step if your Oracle E-Business Suite environment has an active email ID for the user.

1. User Email ID configuration. Seeded users such as sysadmin and operations, do not have an email ID associated with them, so this needs to be set up.
2. Users Originating from PER (HRMS)
   a. Log in to the instance as hrms/<password>. Go to US HRMS Manager, Vision Corporation > People > Enter and Maintain. It opens the form. As an example, give the full name for the user as "Stock, Ms. Pat" for Operations user.
   b. For other users, use this query to get the full name:
   ```
   select description from wf_local_roles where name ="user_name"
   ```
   c. Then edit the record, go to office details section, and in the email field enter the email ID.
3. Users Originating from FND (FND_USER)
   1. Log in to instance as sysadmin/<password>. Go to User Management > User > Search for user. Edit, and add the user's email address.

Activate Notification Role

1. OA Framework ships a new role with role code FND_SAVED_SEARCH_NOTIFICATIONS
2. Log in as sysadmin/<password> - You can also log in other users, but the user you are logging in should have system administration responsibility.
3. Go to User Management responsibility and click Users.
4. Search for the user, and then click on update pencil icon.
5. On the update page, click Assign Roles.
6. Search By: Code
7. Enter the code UMX - FND_SAVED_SEARCH_NOTIFICATIONS
8. Select the row, give the justification for the role in the base page, then click Apply.



For more information, refer to the Descriptive Flexfields section in the Search chapter of this book.

Use Notifications

1. Once the steps to enable notification for an user are done, you can navigate to any search page with list search and infotile enabled.
2. Click Manage Saved Searches from the saved searches drop down. This will open the Manage Saved Search dialog.
3. Click on the Notification icon in the last column. This will take you to the create Notification form.
4. There are three kinds of notification that user can subscribe to. Each is described in detail below.

Option 1 : Report of all records - Choose Type = All Rows.

When subscribed with start date/ end date and repeat unit / interval, this kind of notification sends you a report of all rows matching that criteria at the time of running the report.

You can choose the columns you want to display in the email report. This could be a subset of columns displayed in the table.

Additionally, you can get a bar chart as an attachment in the email. The bar chart can work on a maximum of two attributes. This bar chart is a simple grouping of records.

Option 2 : When new records added - Choose Type = Rows Inserted

When subscribed with start date/end date and repeat unit /interval, this kind of notification sends you a report of all rows that are inserted new from the previous time the job ran.

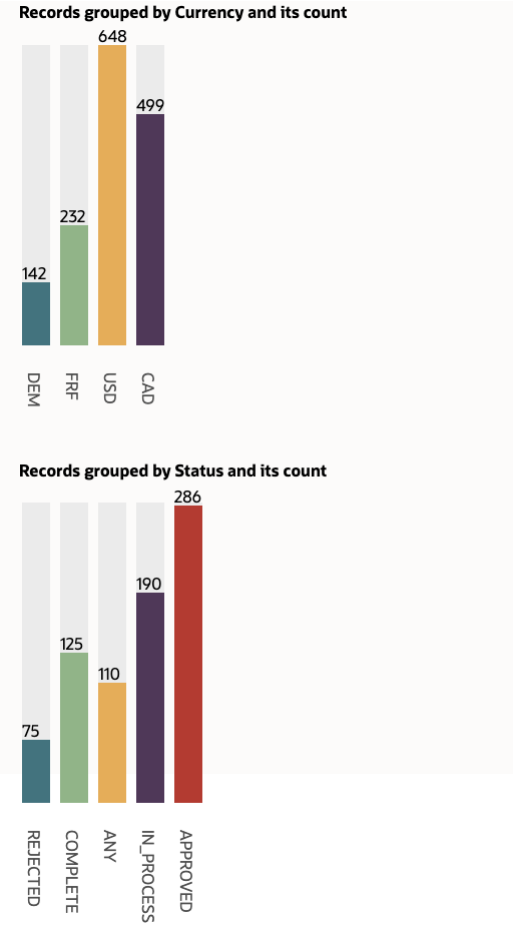
You can choose the columns you want to display in the email report. This could be a subset of columns displayed in the table.

Option 3 : When record count exceeds a set threshold value - Choose Type = Threshold Exceeded

When subscribed with start date/end date and repeat unit/interval, this kind of notification sends you a report of all rows when the count of records exceed the set threshold by the user.

You can choose the columns you want to display in the email report. This could be a subset of columns displayed in the table.

Here is an example of a sample email report as a bar chart report:



Monitoring Saved Search Notifications

System administrators can monitor the saved search notifications requested by each user through an administration page. This page is a search page, and the administrator can use various search criteria to filter the saved search notification he would like to administer.

Navigation: Functional Administrator (Responsibility) > Personalization > Notifications

Administrator Options

Concurrent Manager Tuning

OA Framework uses a Java-based concurrent program to execute each of the repeating saved search notification jobs. The concurrent manager that is shipped, OAF_CP_MGR, has a cache size of 3 and MIN_PROCESSES = 1. That is, the manager can run only one request at a time. If there is a larger number of users for whom this feature is enabled, the administrator could tune these parameters to suit the specific environment needs.

Email Customization

By default OA Framework will use the from address for notification email as 'savedsearch_notification@oracle.com' and the mail subject as 'Saved Search Notification Report - <saved search name>'. If you wish to change these, you can do so by updating the following FND_MESSAGES.

Email Option	FND Message Name
Email Subject	FND_OA_ALERT_EMAIL_SUBJECT
Enable From Address	FND_OA_ALERT_EMAIL_FROM_ADDR

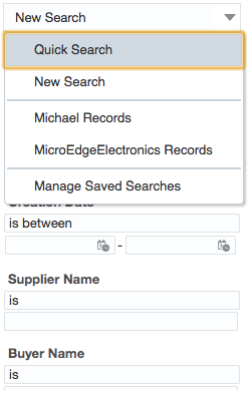
Enabling Quick Search

Objectives

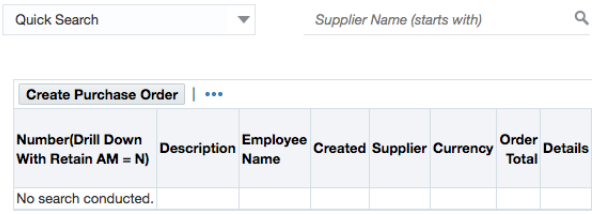
- Quickly enable users to search on a single field in List Search mode without opening the filters area.
- Enable free flow text search for query beans with free text search capabilities.

Examples

The following screenshot shows the Quick Search option being shown when Quick Search is enabled.



The following screenshot shows the Quick Search filter when it is enabled.



Description

- OA Framework will by default pick up the first non-validated LOV or text-based search field among the available filters in List Search mode.
- When users click on New Search or land on a page with the New Search option, they can choose Quick Search or Filtered Search.
- The features works in the same way as in the case of advanced search for developers.
- The criteria dictionary will be populated with the item chosen by the above selection process.
- The search field has default placeholder text: 'FieldName(starts with)'.

Metadata Change and Profile Options

Query Bean Metadata NameÂ	Default ValueÂ	Available ValuesÂ	Admin Personalization EnabledÂ	DescriptionÂ
Enable Quick Search	NULL	True, False	Yes	• If the value is NULL, the quick search option is disabled. • If the value is True, OA Framework will pick up one criterion for the Quick Search field.

Profile Name	Profile Code	Default Value	Available Values	Updatable
FND: Enable Infotile View for Query Beans	FND_ENABLE_INFOTILEVIEW_QUERYBEANS	True	True, False	At all levels
FND: Enable Quick Search for Query Beans	FND_ENABLE_QUICK_SEARCH_QUERYBEANS	True	True, False	At all levels

As of Release 12.2.10, these two profile options have a default value of 'True'.

Both the metadata option and Quick Search profile option must be turned on for this feature to work. There is also a related profile option for enabling Infotile View with query beans.

Developer Control

If developers or system administrators feel that the column chosen for quick search by OA Framework is not appropriate, they can change it via a metadata property. Any column that has a view attribute associated with it can be used for quick search. The attribute 'Quick Search Field Identifier' can be set on the query bean. The ID of the web bean chosen as the column for quick search should be set for the query bean.

Controller Handling

If developers need to get a handle to the event in their controllers, the following API can be called:

```
pageContext.isQuickSearchEvent(queryBeanObject);
```

To get the criteria entered by the user, employ the API `queryBean.getCriteria()` in the same way as for the simple or advanced views panel in the classic query bean.

In the CriteriaMap, you can look for:

- dicCriteria.get(OAViewObject.IS_QUICK_SEARCH); will return Boolean.TRUE
- dicCriteria.get(OAViewObject.CRITERIA_VALUE); will return the search string

The remaining map values that will be returned from the call are:

- dicCriteria.get(OAViewObject.CRITERIA_ITEM_NAME);
- dicCriteria.get(OAViewObject.CRITERIA_LOOKUP_NAME);
- dicCriteria.get(OAViewObject.CRITERIA_CONDITION);
- dicCriteria.get(OAViewObject.CRITERIA_VIEW_ATTRIBUTE_NAME);
- dicCriteria.get(OAViewObject.CRITERIA_DATATYPE);

Enabling Keyword Search

Objective

- Quickly enable users to search on a single field in List Search mode without opening the filters area.
- Enable free flow text search for query beans with free text search capabilities.

Description

- OA Framework by default will pick up the first non-validated LOV or the text based search field among the available filters in List Search mode.
- When users click on the New Search or lands in the page with New Search option, they can choose Quick Search or Filtered Search.
- Works in the same way as in the case of advanced search for developers.
- Criteria dictionary will be populated with the item chosen by the above selection process.
- The search field has 'Search for keywords' as default placeholder text.

Metadata Change and Profile Options

Query Bean Metadata Name	Default Value	Available Values	Admin Personalization Enabled	Description
Enable Quick Search	NULL	True, False	Yes	• If the value is NULL, the Quick Search option is disabled. • If the value is True, OA Framework will pick up one criterion for the Quick Search field.

Profile Name	Profile Code	Default Value	Available Values	Updatable
FND: Enable Infotile View for Query Beans	FND_ENABLE_INFOTILE_VIEW_QUERYBEANS	True	True, False	At all levels
FND: Enable Quick Search for Query Beans	FND_ENABLE_QUICK_SEARCH_QUERYBEANS	True	True, False	At all levels

As of Release 12.2.10, these two profile options have a default value of 'True'.

Both the metadata option and Quick Search profile option must be turned on for this feature to work. There is also a related profile option for enabling Infotile View with query beans.

Controller Handling

If developers need to get a handle to events in their controllers, they can call the following API:

```
OAQueryBean - public boolean isQuickSearchEvent();
```

To get the criteria entered by the user, employ the API `queryBean.getCriteria()` in the same way as for the simple or advanced views panel in the classic query bean.

In the CriteriaMap, you can look for:

- dicCriteria.get(OAViewObject.IS_QUICK_SEARCH); will return Boolean.TRUE
- dicCriteria.get(OAViewObject.CRITERIA_VALUE); will return the search string

The remaining map values that will be returned from the call are:

- dicCriteria.get(OAViewObject.CRITERIA_ITEM_NAME);
- dicCriteria.get(OAViewObject.CRITERIA_LOOKUP_NAME);
- dicCriteria.get(OAViewObject.CRITERIA_CONDITION);
- dicCriteria.get(OAViewObject.CRITERIA_VIEW_ATTRIBUTE_NAME);
- dicCriteria.get(OAViewObject.CRITERIA_DATATYPE);

Developer Control

If developers or system administrators feel that the column chosen for quick search by OA Framework is not appropriate, they can change it via a metadata property. Any column that has a view attribute associated with it can be used for quick search. The attribute 'Quick Search Field Identifier' can be set on the query bean. The ID of the web bean chosen as the column for quick search should be set for the query bean.

If developers need to handle the keyword search for VO manipulation, they can do so by calling this API on pageContext:

```
pageContext.isKeywordSearchEvent(queryBeanObject);
```

To get the actual keyword entered by the user, they can do so by calling this API on pageContext:

```
pageContext.getKeywordSearchParameter(queryBeanObject);
```

Alternatively, product developers can call `queryBean.getCriteria()`. The dictionary will have the following information:

- dicCriteria.get(OAViewObject.IS_KEYWORD_SEARCH); will return a BOOLEAN.TRUE
- dicCriteria.get(OAViewObject.KEYWORD_SEARCH_TEXT); will return the actual search string entered by the user

OA Framework recommends the use of dictionary criteria to get the keyword text rather than the pageContext API, as the pageContext API may not satisfy further enhancements around keyword search. In contrast, the dictionary is always scalable as more information can be added to it.

The query behind the keyword search needs to be executed by the product team code, as the product team will have a better understanding of the keywords for the table. OA Framework does not execute any query if the product team does not handle the keyword search.

Enabling Infotile View of Saved Searches

Oracle Application Framework introduced the infotile feature as a part of Release 12.2.5. List Search utilizes the UI component to render infotiles with a single metric (number of records matching the search criteria

Objective

- This feature can be used to show the total count of records for each saved search (Seeded or User).
- The Infotile View provides a quick overview of a user-saved search.
- Saved searches can be accessed easily, without the user having to select the saved search choice options.
- User-level controls for the tile can be displayed or not according to user preferences.

Description

- OA Framework query bean table VO execution can either take place through OA Framework internal code, or through product team manual handling of the criteria using `queryBean.getCriteria()` / `queryBean.getNonViewAttrCriteria()`.
- This feature works out of the box for cases when OA Framework automatically executes the query for the table VO.
- Minimal uptake is required for cases when a developer tries to get a handle of criteria and modify the table VO query.
- OA Framework prepares the query for each of the saved search in the page rendering cycle, but does not execute the query.
- Once the page loads on the client side, the query prepared is executed using a JDBC callable statement to obtain the count of the number of records.

Metadata Change and Profile Option

Profile Level Control

Profile NameÂ	Profile Code	Default Value	Available ValuesÂ	Description
Enable Quick Search	NULL	False	True, False	• When False, the individual query bean property is not honored. • When True, the individual query bean property determines whether to enable or disable Infotile View.

Query Bean Level Control

Query Bean Property NameÂ	Default Value	Available ValuesÂ	Description
Enable Quick Search	NULL	True, False	• When False, the individual query bean property is not honored. • When True, the individual query bean property determines whether to enable or disable Infotile View.

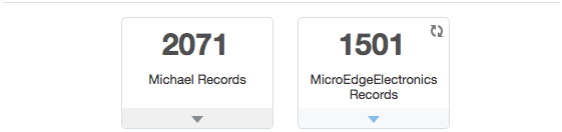
User-Level Control

By default, when the feature is enabled all the seeded views render in the tiles. However, a user view does not show as a tile by default. This is to prevent the page from getting overloaded with information not required by the user.

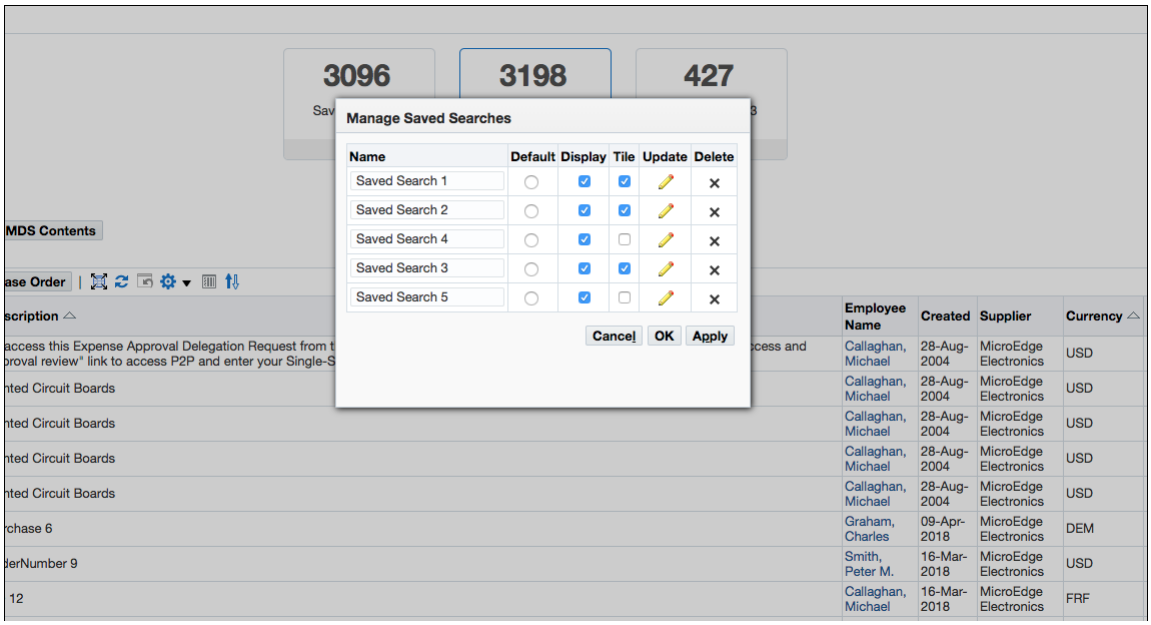
Hence a user-level control is established. While creating the view and then managing it in the Manage Saved Search popup, users can choose to set the display of the view in the tile to On or Off.

Users can also refresh the count of each of the individual saved searches using a small refresh icon that appears in the top-right corner of the tile. This will work if the user keeps the web page idle and then clicks it later. However, the Oracle E-Business Suite session must be active for a successful refresh to happen.

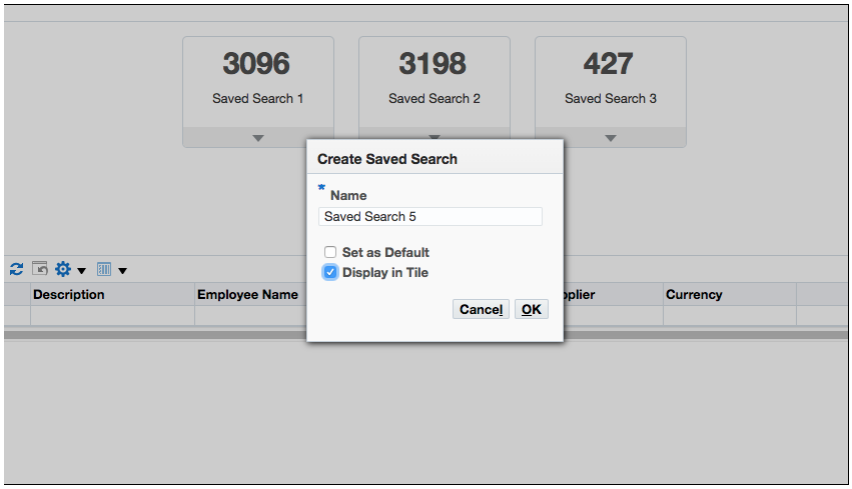
The following screenshot shows the infotile that is rendered when the Infotile View for a saved search is enabled.



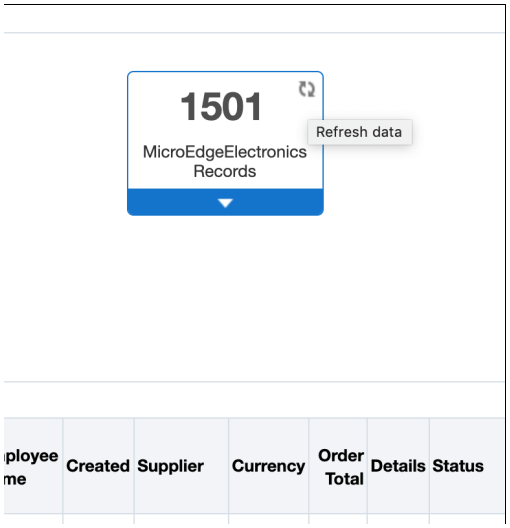
The following screenshot shows how Infotile View can be turned on or off via the Manage Saved Search popup.



The following screenshot shows the Infotile View when the saved search is created through the Create Saved Search popup.



The following screenshot shows the refresh icon at the top right corner of the tile.



OA Framework Approach to Get Row Count

OA Framework in the page processing cycle (Process Request) will prepare the SQL query for each query bean view and save the query to be executed in a map. Later, the client raises a REST request to the server to execute a JDBC callable statement based on the saved query, and thereby get the row count.

While the page is being prepared, OA Framework parses all views whose tile view is enabled, then gets the associated view content from MDS. The view content is subsequently used by OA Framework to build the WHERE clause for the table VO. OA Framework will use its internal `OAViewObjectImpl` methods to derive SQL auery for the view contents.

Note the following important points:

1. If developers modify the SQL query for the VO based on criteria in the VO, OA Framework cannot build the SQL query accurately.
2. At present, OA Framework developers use the `initQuery()`, `executeQuery()` or `executeQueryForCollection()` APIs to modify the VO's query based on the current view.
3. If OA Framework calls any of these methods for preparing the SQL query for an infotile, this will result in execution of the BC4J query which will not give the best performance.

Hence an uptake is required to refactor this code.

Enabling KFF Search

Release 12.2.10 introduces support for KFF Search. This enhancement of the query bean search criteria provides significant new capabilities, such as adding KFF items as search criteria items in an OAF search region.

The following items are addressed as part of this feature:

List Search:

- New search adding KFF as search criteria (Note: KFF criteria can only be added once to the search panel.)
- Save a new search that contains KFF search criteria
- Update existing saved search to include KFF or modify the existing KFF search criteria

Classic Search:

- Simple Search with KFF criteria
- Advanced Search with KFF criteria (Note: KFF criteria can only be added once to the search panel)
- Save Search with KFF criteria

Setting up Key Flexfield in QueryBean

There are three main tasks that the developer needs to perform when setting up a key flexfield (KFF) as a criteria item in QueryBean.

Each of these tasks should be validated, inspected, and tested before moving to the next.

The three tasks and their steps are as follows.

1. **Include KFF in search result table region**

   a. Define the KFF column in the table.

   b. Runtime control - Add code to initialize the KFF in page level controller or any ccontroller for the table region. This will include setting key properties such as structureCode and CCIDAttributeName. In addition, the product code can specify various other properties, such as extra WhereClause on KFF, validation Date, and FlexMaps.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
  {
    super.processRequest(pageContext, webBean);
      OAKeyFlexBean kffId =
        (OAKeyFlexBean) webBean.findIndexedChildRecursive("AssetNumberFlex");
      setPropKFF(kffId);
}
  private void setPropKFF(OAKeyFlexBean kffId)
  {
    // Set the code combination LOV
    kffId.useCodeCombinationLOV(true);
    //set the structure code for the item key flex
    kffId.setStructureCode("Asset Number Flex Field");
    //Set the attribute name to the item
    kffId.setCCIDAttributeName("CCID");
    kffId.setDynamicInsertion(true);
  }
```

   Note that the 'CCID' attribute above needs to be already defined in the backing ViewObject for the result table, FWK will use this 'CCID' attribute to store the actual KFF ccid value, which is then used to look up for the combination string in the KFF base table.

2. **Add KFF search criteria to advanced search panel**

   a. Add KFF search criteria row - Add a new criteria row under the advanced search criteria panel, then insert 'flex' item to the criteria row. The properties in the 'Flexfields' section should match what is defined for the search table KFF item.

   b. Add KFF search criteria map - Add a new criteria map under the 'advancedSearchMappings' for the newly added KFF search criteria above.

   c. Runtime control - Add code to initialize the KFF criteria search item in advanced search region. **Note**: The KFF initialization code should only be added to the controller associated with the 'Advanced Search' region, as 'region1' under the 'advancedSearchPanel'. This is needed because the same advanced search region controller can be accessed from the global personalization page to create or

update seeded views associated with the search page. Developers do not have direct access to the KFF web bean under the advanced search region, so need to use the `OAAdvancedSearchBean.setFlexCriterionInfoMap() API` to pass in the KFF properties that are to be set. The following example illustrates this:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
 {
   super.processRequest(pageContext, webBean);
   // Initialize Advanced Search KFF.
   java.util.HashMap kffInfoMap = new java.util.HashMap();
   OAFlexCriterionInfo assetKFFInfo = new OAFlexCriterionInfo();

   // Set ID of KFF item in result table that
   // this KFF search item is mapped to.
   assetKFFInfo.setTableFlexItemId("AssetNumberFlex");

   // Set attributes required to initialize the KFF item:
   // 1. StructureCode.
   // 2. CCID attribute name.
   assetKFFInfo.setCCIDAttributeName("CCID");
   assetKFFInfo.setStructureCode("Asset Number Flex Field");
   // Set any extra attributes on the KFF.
   // SssetKFFInfo.setWhereClause(...)
   // AssetKFFInfo.setFlexMaps(...)
   // <k, v> - k: ID of the search KFF item name in theadvanced search region.
   kffInfoMap.put("AssetFlexCItem", assetKFFInfo);
   // Repeat above for multiple KFF search criteria items.
   // Invoke the below API to add the HashMap to the AdvancedSearchBean.
   ((OAAdvancedSearchBean)webBean).setFlexCriterionInfoMap(kffInfoMap);
 }
```

3. **(Optional) Add KFF search criteria to simple search panel** - This task is needed only for cases where KFF search criteria are to be included when implementing Classic Search with the Simple Search panel.

   a. Define KFF simple search criteria item.

   b. Define KFF simple search criteria map.

   c. Runtime control - Add code to initialize the KFF criteria search item in the simple search region.

      For the simple search region, developers can directly get the handle to the KFF web bean and set the criteria item on the web bean itself. The initialization code needs to be inside the page level controller, or whichever controller is associated with the simple search region.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
super.processRequest(pageContext, webBean);
//initialize the KFF item in results table.
OAKeyFlexBean kffId =
(OAKeyFlexBean) webBean.findIndexedChildRecursive("AssetNumberFlex");
setPropKFF(kffId);
// Initialize Simple Search KFF criteria item.
 OAKeyFlexBean kffSrchId =
(OAKeyFlexBean) webBean.findIndexedChildRecursive("SearchAssetNumberFlex");
kffSrchId.setDynamicInsertion(false);
// Always turn off dynamic insertion for the search criteria item
}
```

Dependent Criteria in Advanced Search

This feature enables developer to build criteria items dependent on other criteria items. Advanced search is utilized by OAF in these three flows: Advanced Search, Create Saved Search, and Update Saved Search. This feature is supported only in List Search.

Any specific coding done for this feature must be supported on all three flows. In the case of Create Saved Search or Update Saved Search, the main page comes from OAF and the advanced search web bean metadata of the product is embedded as a region in that page.

It is advisable to code controller specific logic for this feature in advanced search web bean controller, and not at page controller or at table level controller. We will assume a use case that there is a Asset Flex Criteria item(Flex Bean that is dependent on Asset Type criteria (Message Text Input Bean

```
<oa:criteriaRow id="AssetFlexCriteriaRow">
<ui:contents>
<oa:messageStyledText id="item131" prompt="Asset Flex"/>
<oa:flex id="AssetFlexCItem" flexFieldAppShortName="FND" flexName="ASN#" flexStyle="key" rendered="${oa.AdvancedSearchFilterPVO1.AssetFlexRendered}" paren
</ui:contents>
</oa:criteriaRow>
<oa:criteriaRow id="AssetTypeCriteriaRow">
<ui:contents>
<oa:messageStyledText id="item11" prompt="Asset Type"/>
<oa:messageTextInput id="AssetTypeCItem">
<ui:primaryClientAction>
<ui:firePartialAction event="assetTypeChange"/>
</ui:primaryClientAction>
</oa:messageTextInput>
</ui:contents>
</oa:criteriaRow>
```

The following are the six strategic steps required by developers to enable dependent criteria:

1. Since Asset Type criteria is a trigger element, **enable fire partial change on that item**. The event name associated in the above example is "assetTypeChange".
2. Asset flex item is the child filter. **This child will be rendered on/off based on the Asset Type entered by the user.** Therefore, bind the rendered property of this element to a PVO attribute in the same way you enable PPR on items. The PVO in the above example is AdvancedSearchFilterPVO1.AssetFlexRendered.
3. For each of the child items that are dependent on the parent, you can set the **parentFilterIdentifier attribute**. In the above example for the flex, the parent is set to Asset Type criteria item.
4. The next step is to initialize this PVO based on the business logic. Remember that there is saved search flow, create saved search flow, advanced search flow, update saved search flow. To handle the initialization properly, code the following in the **advanced search controller process request**.

```
OAQueryBean queryBean =
        (OAQueryBean)pageContext.getRootWebBean().findChildRecursive("QueryRN");
    if (queryBean != null && "CUSTOMIZE".equals(queryBean.getCurrentSearchPanel()) { // Saved Search Flow
        Dictionary criteria[] =
            queryBean.readSavedSearchCriteria(pageContext);
        boolean isAssetTypeCriteriaFound = false;
        if (criteria != null && criteria.length > 0) { // Saved Search Flow
            for (int i = 0; i < criteria.length; i++) {
                Dictionary _crit = criteria[i];
                String criteriaName =
                    (String)_crit.get(OAViewObject.CRITERIA_ITEM_NAME);
                if (criteriaName != null &&
                    "AssetType".equals(criteriaName)) {
                    String criteriaValue =
                        (String)_crit.get(OAViewObject.CRITERIA_VALUE);
                    if (criteriaValue != null) {
                        isAssetTypeCriteriaFound = true;
                        am.handleAssetTypeChangeEvent(criteriaValue);
                    }
                }
            }
            if (!isAssetTypeCriteriaFound) { // In saved search the filter is not present. So default PVO.
                am.initializeAdvancedSearchFilterPVO();
```

```
                }
            } else { // New Advanced Search Flow
                am.initializeAdvancedSearchFilterPVO();
            }
        }
        else { // Create Saved Search or update Saved Search flow
            Criteria savedCriteria =
                OAQueryUtils.getSavedSearchCriteriaForUpdatePG(pageContext);
            boolean isAssetTypeCriteriaFound = false;
            if (savedCriteria != null && savedCriteria.size() > 0) {
                for (int k = 0; k < savedCriteria.size(); k++) {
                    Criterion individualCrit = savedCriteria.get(k);
                    String nameOfCriterion = individualCrit.getComponentRef();
                    if ("AssetType".equals(nameOfCriterion)) {
                        String criteriaValue = individualCrit.getOperand();
                        if (criteriaValue != null) {
                            isAssetTypeCriteriaFound = true;
                            am.handleAssetTypeChangeEvent(criteriaValue); // update saved search
                        }
                    }
                }
                if (!isAssetTypeCriteriaFound) {
                    am.initializeAdvancedSearchFilterPVO();
                }
            } else {
                am.initializeAdvancedSearchFilterPVO(); // create saved search
            }
        }
```

5. Continuing with the steps needed, in the **process form request of advanced search controller**, you now need to handle the asset type change event and set the PVO based on the business logic.

```
String event = pageContext.getParameter("event");
        if ("assetTypeChange".equals(event)) {
            String source =
                getAdvancedSearchItemNameForOriginalUINodeName((OAAdvancedSearchBean)webBean,
                      "AssetTypeCItem",
                      pageContext);
            FwkTbxNewEmployeeAMImpl am = getApplicationModule(pageContext);
// As mentioned at the top of the document,
// AM is associated with advanced search depending on the page.
// If the page is create/update saved search then AM of the advanced search
// is nested under OAF AM.
am.handleAssetTypeChangeEvent(pageContext.getParameter(source));
        }
public FwkTbxNewEmployeeAMImpl getApplicationModule(OAPageContext pageContext) {
        OAApplicationModuleImpl am =
            (OAApplicationModuleImpl)pageContext.getRootApplicationModule();
        if (am instanceof FwkTbxNewEmployeeAMImpl) {
            return (FwkTbxNewEmployeeAMImpl)am;
        }
        String amNames[] =
            pageContext.getRootApplicationModule().getApplicationModuleNames();
        if (amNames != null) {
            for (int i = 0; i < amNames.length; i++) {
                String amName = amNames[i];
                if (amName.indexOf("FwkTbxNewEmployeeAM") >= 0) {
                    return (FwkTbxNewEmployeeAMImpl)pageContext.getRootApplicationModule().findApplicationModule(amName);
                }
            }
        }
        return null;
    }
        public String getAdvancedSearchItemNameForOriginalUINodeName(OAAdvancedSearchBean webBean,
                                                                      String webBeanNameToSrchFor,
                                                                      OAPageContext pageContext) {
        try {
            OAAdvancedSearchBean adv = (OAAdvancedSearchBean)webBean;
            int displayedCriteriaCount = adv.getDisplayedCriteriaCount();
            for (int i = 0; i < displayedCriteriaCount; i++) {
                String newItemName =
                    OAWebBeanConstants.VALUE + Integer.toString(i);
                if ((pageContext.getParameter(newItemName) != null)) {
                    String originalChildName =
                        adv.getOriginalUINodeName(newItemName);
                    if (originalChildName.equals(webBeanNameToSrchFor))
                        return newItemName;
                }
            }
        } catch (Exception e) {
            System.out.println("Something went wrong.");
        }
        return null;
    }
```

6. Finally, **implement the AM methods**, initialize the PVO, and handle the criteria change.

```
public AdvancedSearchFilterPVOImpl getAdvancedSearchFilterPVOInstance() {
        ViewObject allVOs[] = getViewObjects();
        for(int i=0;i>allVOs.length;i++){
            ViewObject currentVO = allVOs[i];
            String fullName = currentVO.getFullName();
            if(fullName.indexOf("CustomJRADTableViewsAM") >=0 && fullName.indexOf("AdvancedSearchFilterPVO") >=0){
                return (AdvancedSearchFilterPVOImpl)currentVO;
            }
        }
        return getAdvancedSearchFilterPVO1();
    }
    public void initializeAdvancedSearchFilterPVO() {
        OAViewObjectImpl advSrchPropertiesVO = getAdvancedSearchFilterPVOInstance();
        if (advSrchPropertiesVO != null) {
            if (advSrchPropertiesVO.getFetchedRowCount() == 0) {
                advSrchPropertiesVO.setMaxFetchSize(0);
                advSrchPropertiesVO.executeQuery();
                Row row = advSrchPropertiesVO.createRow();
                advSrchPropertiesVO.insertRow(row);
                // Set the primary key value for this single-row VO.
                row = (OARow)advSrchPropertiesVO.first();
                row.setAttribute("RowKey", new Number(1));
            }
        }
        Row currentRow = advSrchPropertiesVO.getCurrentRow();
        if (currentRow != null) {
            currentRow.setAttribute("AssetFlexRendered", Boolean.FALSE);
        }
    }
    public void handleAssetTypeChangeEvent(String assetTypeValue) {
        OAViewObjectImpl advSrchPropertiesVO = getAdvancedSearchFilterPVOInstance();
        if (advSrchPropertiesVO != null) {
            if (advSrchPropertiesVO.getFetchedRowCount() == 0) {
```

```
                advSrchPropertiesVO.setMaxFetchSize(0);
                advSrchPropertiesVO.executeQuery();
                Row row = advSrchPropertiesVO.createRow();
                advSrchPropertiesVO.insertRow(row);
                // Set the primary key value for this single-row VO.
                row = (OARow)advSrchPropertiesVO.first();
                row.setAttribute("RowKey", new Number(1));
            }
        }
        Row currentRow = advSrchPropertiesVO.getCurrentRow();
        if (currentRow != null) {
            if ("ipad".equalsIgnoreCase(assetTypeValue)) {
// Logic here is: If user enters asset type as iPad, asset flex is rendered, else asset flex is not rendered.
                currentRow.setAttribute("AssetFlexRendered", Boolean.TRUE);
            } else {
                currentRow.setAttribute("AssetFlexRendered", Boolean.FALSE);
            }
        }
    }
```

Use Cases for Developer Uptake of Infotiles

Some important examples for infotile uptake by developers are described in detail below.

Three distinct cases will be considered in turn.

Case 1: Developer calls queryBean.getCriteria(to execute the query

In this case, the developer builds the full query for the table view object. OAF does not handle any of the filters.

Existing code handles the criteria, sets the VO where clause and VO where clause parameters, and then executes the query.

Step 1

Move this logic of setting where clause and whereclause params to VOImpl under a method.

Let's say we call this method `handleCriteria()`. The sample skeleton of the method looks like this:

```
public void handleCriteria(Dictionary[] criteriaDictionary)
{
// Iterate the dictionary.
// Set where clause on the ViewObject.
// Set where clause params on the view object.
// Do not execute the query here.
}
```

Step 2

In your VOImpl.java, override the superclass method `initWhereClauseFromCriteriaDictionary`.

The method should look exactly like this in your code:

```
public boolean initWhereClauseFromCriteriaDictionary(Dictionary[]
customDic, String viewName)
{
handleCriteria(customDic); // Call to the above method.
return false; // This will instruct OAF that it should not interpret any criteria in the dictionary, as it handled them already.
}
```

Step 3

Wherever you have your code to handle the criteria (your page controller, query bean controller or advanced search bean controller, make this change:

```
......... // Existing lines of code. Code to handle the criteria moved to Step 1.
Dictionary criteria[] = queryBean.getCriteria();
if(criteria != null && criteria.length > 0)
{
OAViewObjectImpl myVO = ... // Get a handle to the table view object.
myVo.handleCriteria(criteria);
myVo.executeQuery(); // Remember to execute the query.
}
```

Case 2: Developer calls queryBean.getNonViewAttrCriteria() to build part of the query

In this case, the developer builds the portion of the query for non-view based criteria, but does not execute the table VO query: OA Framework executes the table VO query.

Existing code handles the non-view attribute criteria, sets the VO where clause and where clause parameters for those non-view attributes, and then waits for OA Framework to bind other view-based attributes.

Step 1

Move this logic of setting where clause and where clause params to VOImpl under a method. Let's say we call this method `handleCriteria()`. A sample skeleton of the method looks like this:

```
public void handleNonViewAttributeCriteria(Dictionary[] criteriaDictionary)
{
// Iterate the dictionary. Filter all non view attribute criteria's. You
can check if the criteria has view Attr or not by checking the presence of
OAViewObject.CRITERIA_VIEW_ATTRIBUTE_NAME.
// criteriaDictionary[i].get(OAViewObject.CRITERIA_VIEW_ATTRIBUTE_NAME);
If this returns some value then it is a view based criteria dont handle. If
it is not then handle.
// set where clause on the view object.
// set where clause parameters on the view object.
}
```

Step 2

In your VOImpl.java, override the superclass method initWhereClauseFromCriteriaDictionary.

The method should look exactly like this in your code.

```
public boolean initWhereClauseFromCriteriaDictionary(Dictionary[]
customDic, String viewName)
{
handleNonViewAttributeCriteria(customDic); // call to the above method.
return true; // This will instruct OAF that it shouldÂ Â interpret any
criteria in the dictionary that are not yet handled.
}
```

Step 3

Wherever you have your code to handle the non-view attribute criteria (your page controller, query bean controller or advanced search bean controller), make this change:

```
..... // Existing lines of code. Code to handle the criteria moved to
Step1.
Dictionary criteria[] = queryBean.getNonViewAttrCriteria();
if(criteria != null && criteria.length > 0)
{
OAViewObjectImpl myVO = ... // Get a handle to the table view object.
myVo.handleNonViewAttributeCriteria(criteria);
// Do not execute the query.
}
```

Case 3: Developer overrides the executeQuery(), executeQueryForCollection(), or initQuery() before executing the query

In this case, the developer overrides the executeQuery(), executeQueryForCollection(), initQuery(), or any other method, before executing the query and trying to change the where clause and where clause parameters.

We will take an example of a scenario where seeded views alone are manipulated based on the name, and then the corresponding where clause and where clause parameters are set.

Step 1

All the logic you have in your VOImpl method executeQuery(), executeQueryForCollection(), or initQuery() that specifically deals with view name plus building criteria is moved to a centralized method.

```
public void handleCriteria(Dictionary[] criteriaDictionary, String viewName)
{
// Here the use case is based on seeded view name. So use view name and
build the where clause , where clause params.
// Do not execute query here.
}
```

Step 2

In your VOImpl.java, override the superclass method initWhereClauseFromCriteriaDictionary.

The method should look exactly like this in your code:

```
public boolean initWhereClauseFromCriteriaDictionary(Dictionary[] customDic,
String viewName)
{
handleCriteria(customDic, viewName); // call to the above method.
return true; // This will instruct OAF that it shouldÃ interpret any
criteria in the dictionary that are not yet handled.
}
```

Step 3

In your controller, do the following to handle the seeded view load and execute the query.

```
processRequest()
{
.........//other code.
if(CUSTOMIZE.equals(panel))
{
String currentCust = queryBean.getCurrentCustomization();
if(currentCust is seeded)
{
OAViewObjectImpl myVO = ... // Get a handle to the table view object.
myVo.handleCriteria(criteria, seededSavedSearchName);
myVo.executeQuery();
}
}
}
```

The overall aim of this strategy is to:

1. Separate the criteria handling part from the query execution part. The criteria handling part should be a separate method.
2. Override the OAF initWhereClauseFromCriteriaDictionary and call the method created in (1).
3. In the place where you usually execute the query, if needed call (1) and then call executeQuery().

Actions For Query Snippets

With regard to the return value of the method `initWhereClauseFromCriteriaDictionary`, when developers build query snippets for all the criteria in their code, there is no need for OA Framework to interpret the query criteria. In such cases, developers must return a value of False from this overridden VOImpl method. Specifically, if developers use `queryBean.getCriteria()` code in their flows, they must return a value of False from this method.

If developers build portions of query snippets based on criteria in their code, however, OA Framework needs to interpret the query criteria further. In such cases, developers must return a value of True from this overridden VOImpl method. Specifically, if they use `queryBean.getNonViewAttributeCriteria()` code in their flow, they must return a value of True from this method.

Debugging List Search

This section describes several different strategies you can use to investigate issues with List Search.

Table Diagnostics

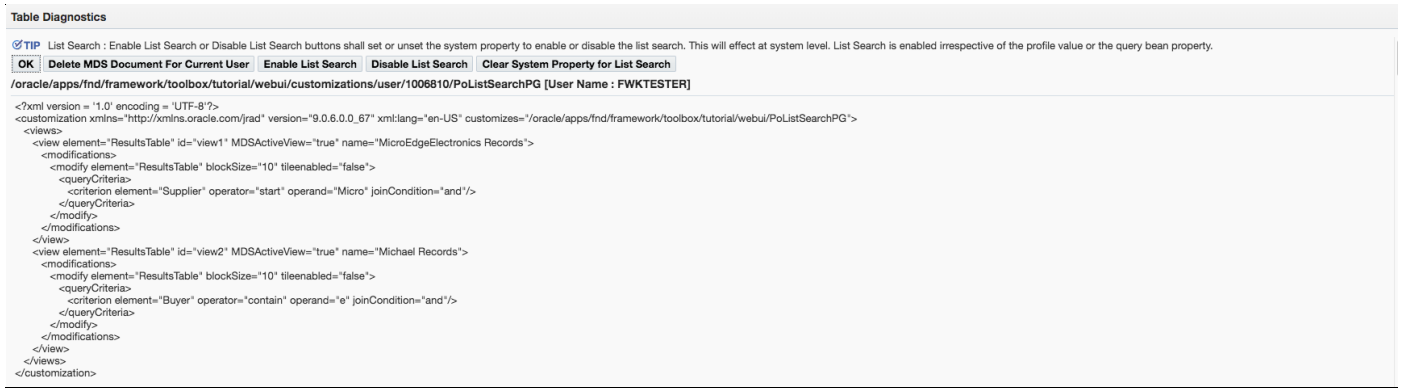
When the FND_DIAGNOSTICS profile is enabled, a button called Table Diagnostics is rendered. When it is clicked, a popup is displayed with information about various components. Each of these is discussed below.



MDS Views

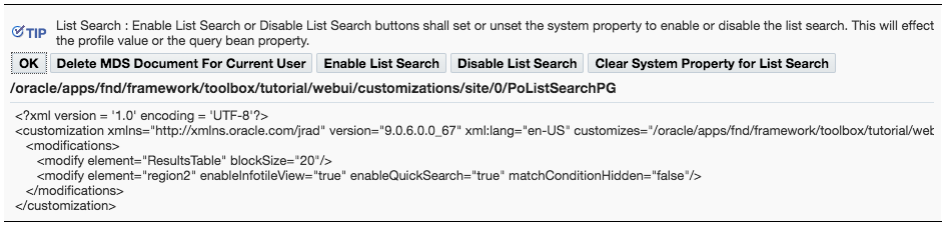
All the user (current user) created views will be displayed here. The contents of the view will be displayed. This will help developers to quickly check database contents without the need to write an SQL query to get

the view contents.



Site Level Personalization

Any personalization done at the site level will be shown.



Buttons

There are buttons to enable List Search for current user, disable List Search for current user, delete MDS view content for the current user.

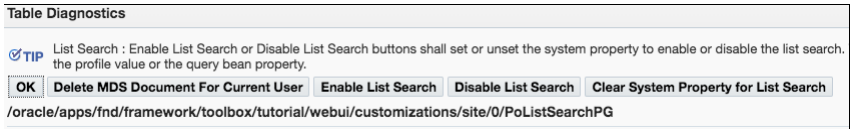


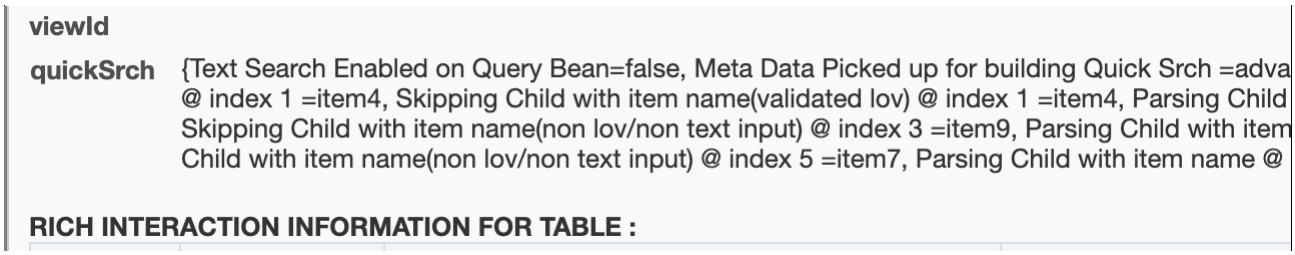
Table Information

This returns information about the currently selected view (saved search).



Quick Search

This shows Quick Search related information (if enabled)



Rich Table Information For Resultant Table

RICH INTERACTION INFORMATION FOR TABLE :

Column No	Column ID	Column Prompt	Column Rendered(Without Context)	Column Rendered(With Context)	
0	OrderNumCol	Number(Drill Down With Retain AM = N)	true	true	
1	OrderDescCol	Description	true	true	
2	BuyerCol	Employee Name	true	true	
3	CreationDateCol	Created	true	true	
4	SupplierCol	Supplier	true	true	
5	CurrencyCol	Currency	true	true	
6	OrderTotalCol	Order Total	true	true	
7	DetailsCol	Details	true	true	
8	StatusCol	Status	true	true	
9	column1	Confirm Flag	true	true	
10	OrderNumCol1	Number(Drill Down With Retain AM = Y)	true	true	
11	column2	Creation Date (Relative Date Criteria) With Range	true	true	
12	column21	Creation Date (Relative Date Criteria) Without Range	true	true	

Query Execution Related Information

Query Related Info..

Query stmt =>

SELECT * FROM (SELECT PurchaseOrderHeaderEO.HEADER_ID AS ORDER_NUMBER, PurchaseOrderHeaderEO.DESCRIPTION, PurchaseOrderHeaderEO.
PurchaseOrderHeaderEO.CREATION_DATE, PurchaseOrderHeaderEO.BUYER_ID, EmployeeEO.FULL_NAME AS BUYER_NAME, SupplierEO.NAME AS SUPP
BUYER_EMAIL, (select sum(nvl(line.quantity, 0) * nvl(line.unit_price, 0)) from fwk_tbx_po_lines line where line.header_id = PurchaseOrderHeaderEO.HEADER_I
EmployeeEO.EMPLOYEE_ID, PurchaseOrderHeaderEO.STATUS_CODE, PurchaseOrderHeaderEO.CONFIRM_FLAG FROM FWK_TBX_PO_HEADERS Purchas
EmployeeEO, FWK_TBX_SUPPLIERS SupplierEO WHERE PurchaseOrderHeaderEO.BUYER_ID = EmployeeEO.EMPLOYEE_ID and PurchaseOrderHeaderEO.
WHERE ((UPPER(SUPPLIER_NAME) like UPPER(:1) AND (SUPPLIER_NAME like :2 OR SUPPLIER_NAME like :3 OR SUPPLIER_NAME like :4 OR SUPPLIER_

Where Clause =>

(UPPER(SUPPLIER_NAME) like UPPER(:1) AND (SUPPLIER_NAME like :2 OR SUPPLIER_NAME like :3 OR SUPPLIER_NAME like :4 OR SUPPLIER_NAME lik

Where Clause Params =>

At Index 1 =>Micro%

At Index 2 =>MI%

At Index 3 =>mi%

At Index 4 =>Mi%

At Index 5 =>mI%

Is Prepared For Exec ? =>

false

Stack Trace of Query Execution

Stack Trace of the query Exec =>

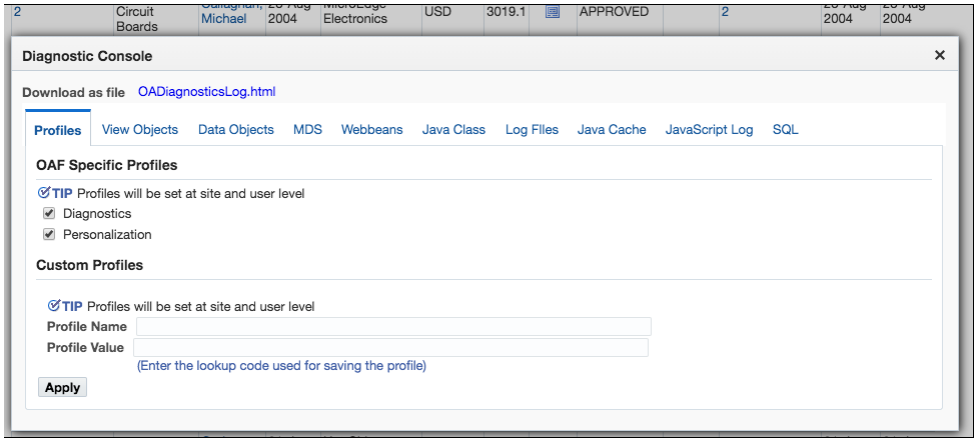
Class Name	Method Name	Line Num	File Version
java.lang.Thread	getStackTrace	1589	Not available
oracle.apps.fnd.framework.server.OAViewObjectImpl	executeQuery	480	$Header: OAViewObjectImpl.java 120.31.1202(
oracle.apps.fnd.framework.webui.OAWebBeanBaseTableHelper	queryDataInternal	2080	$Header: OAWebBeanBaseTableHelper.java 12
oracle.apps.fnd.framework.webui.OAAdvancedTableHelper	processRequestAfterController	910	$Header: OAAdvancedTableHelper.java 120.44
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequest	716	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanTableHelper	processRequest	2541	$Header: OAWebBeanTableHelper.java 120.19.
oracle.apps.fnd.framework.webui.OAAdvancedTableHelper	processRequest	809	$Header: OAAdvancedTableHelper.java 120.44
oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean	processRequest	787	$Header: OAAdvancedTableBean.java 120.25.
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequestChildren	1101	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequestChildren	1068	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequest	717	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanContainerHelper	processRequest	283	$Header: OAWebBeanContainerHelper.java 12
oracle.apps.fnd.framework.webui.beans.layout.OARowLayoutBean	processRequest	-1	$Header: OARowLayoutBean.java 120.5 2005/
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequestChildren	1101	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequestChildren	1068	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequest	717	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanContainerHelper	processRequest	283	$Header: OAWebBeanContainerHelper.java 12
oracle.apps.fnd.framework.webui.beans.layout.OATableLayoutBean	processRequest	353	$Header: OATableLayoutBean.java 120.6 2008
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequestChildren	1122	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequestChildren	1068	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequest	717	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanContainerHelper	processRequest	283	$Header: OAWebBeanContainerHelper.java 12
oracle.apps.fnd.framework.webui.OAPanelSplitterHelper	processRequest	343	$Header: OAPanelSplitterHelper.java 120.0.120
oracle.apps.fnd.framework.webui.beans.layout.OAPanelSplitterBean	processRequest	331	$Header: OAPanelSplitterBean.java 120.0.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequestChildren	1101	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequestChildren	1068	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanHelper	processRequest	717	$Header: OAWebBeanHelper.java 120.29.1202
oracle.apps.fnd.framework.webui.OAWebBeanContainerHelper	processRequest	283	$Header: OAWebBeanContainerHelper.java 12

Information About Current View

Diagnostic Console

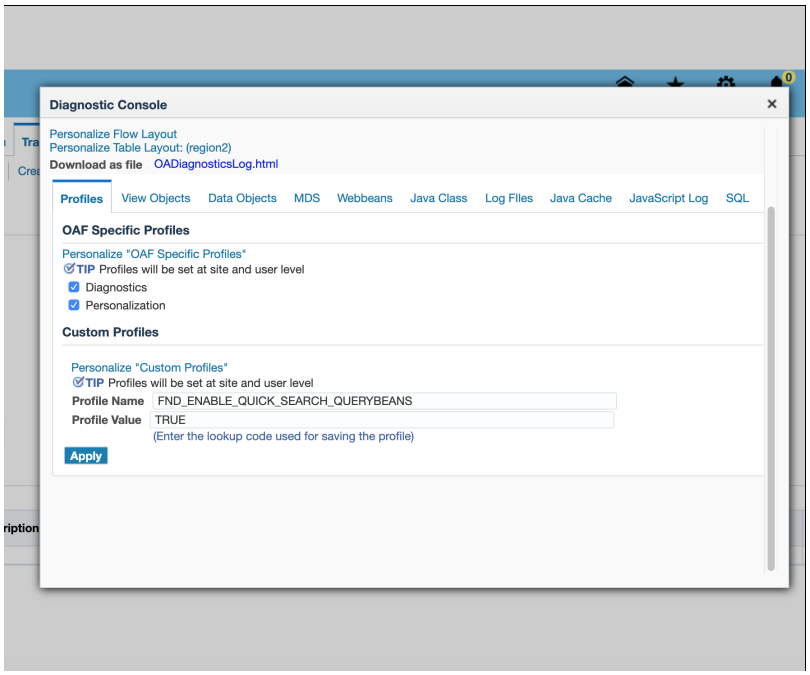
This feature is a more of page level diagnostics for any OA Framework page. When FND_DIAGNOSTICS is enabled, then the page level button Diagnostic Console appears at the bottom of the page.

On clicking the button, a popup with subtab content opens. Each of the subtabs has specific functionality for an OA Framework page. They are as shown below.



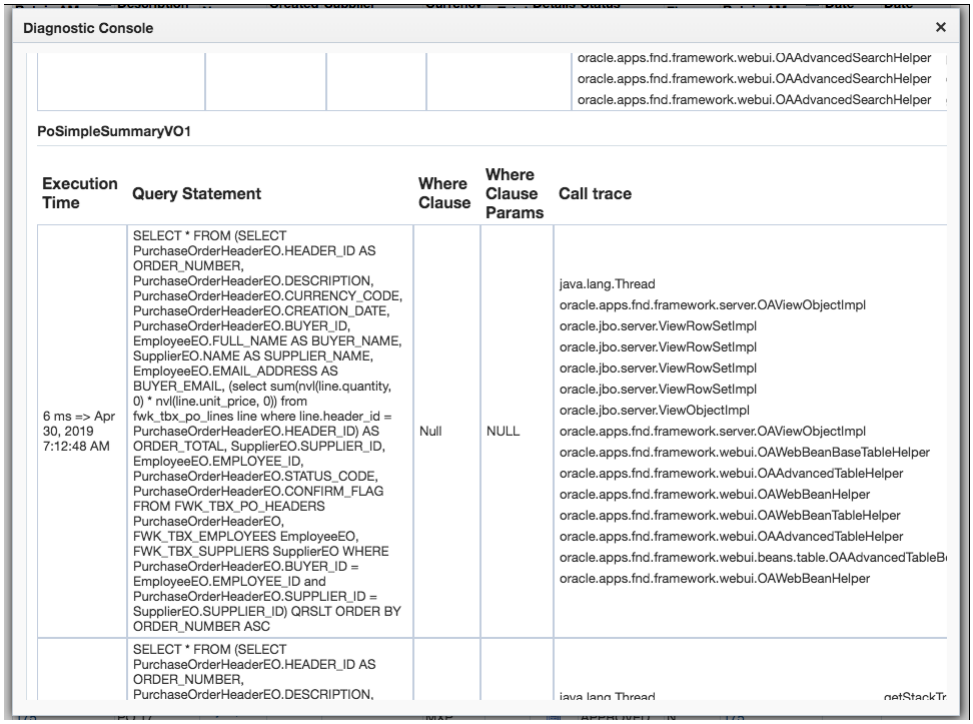
Profiles Tab

Allows you to quickly turn on/off OA Framework specific profiles. Also, other Oracle E-Business Suite profiles can be turned on and off at site and user level.



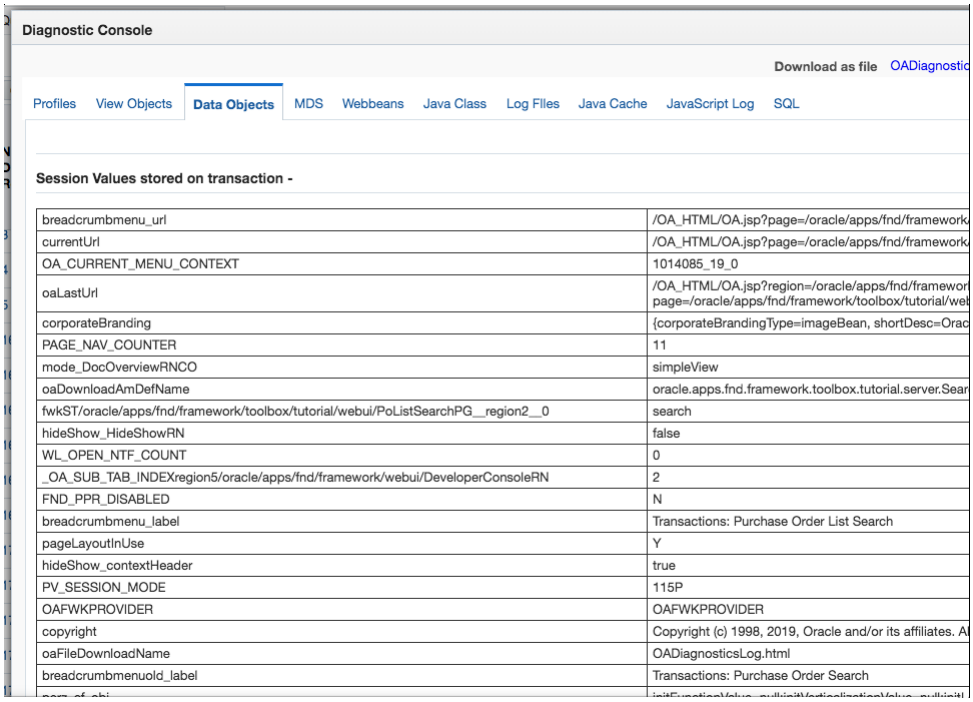
View Objects Tab

This tab gives more information about the all the view objects of the page. Information such as execution time, Query statement, where clause, where clause parameters, and call stack trace can be seen.

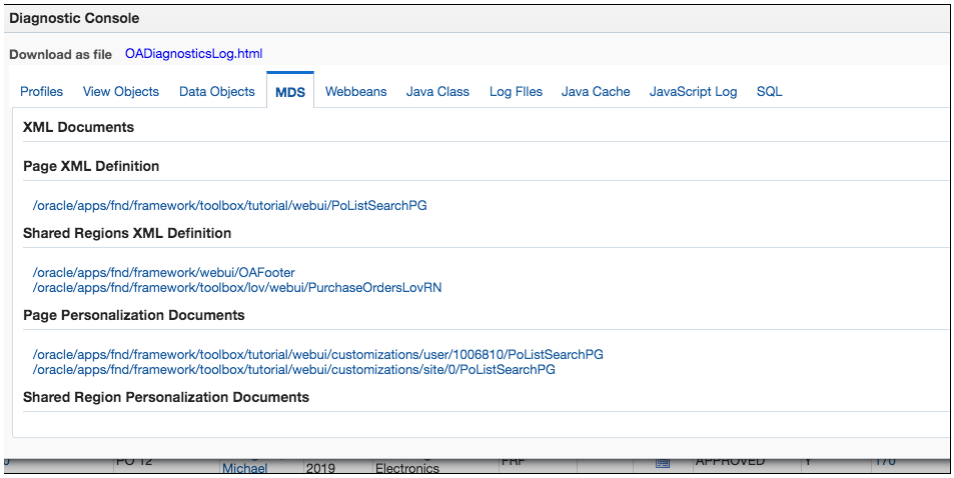


Data Objects Tab

This gives information about transaction , session, page context values at the current state of the application tier.

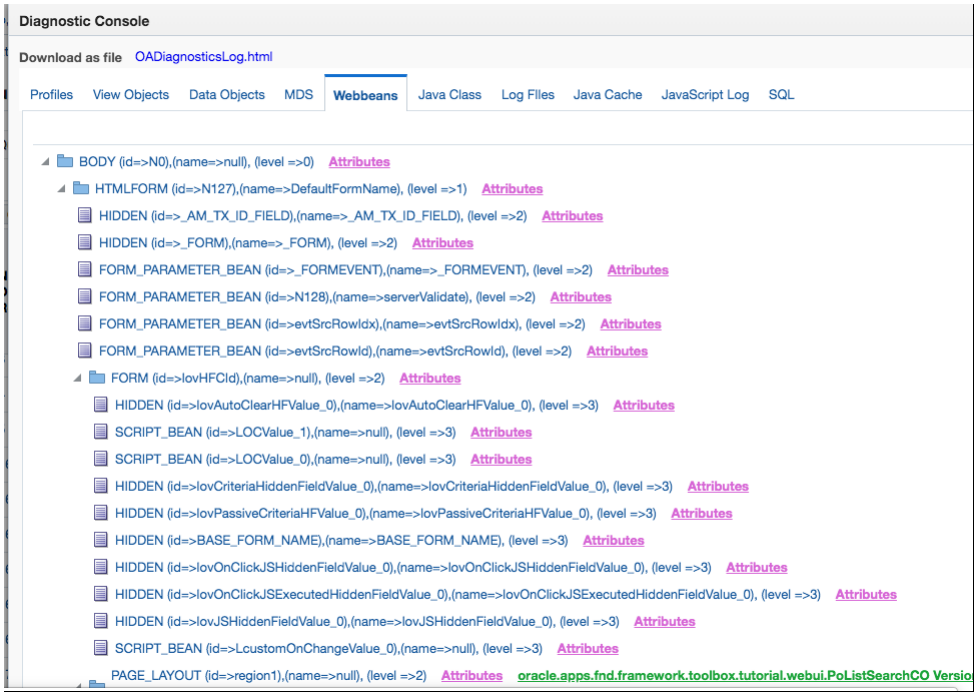


MDS Tab



This gives information about the XML involved in the page. All base page documents, personalized page documents, and extended region documents are shown, with links to see the entire content of those documents.

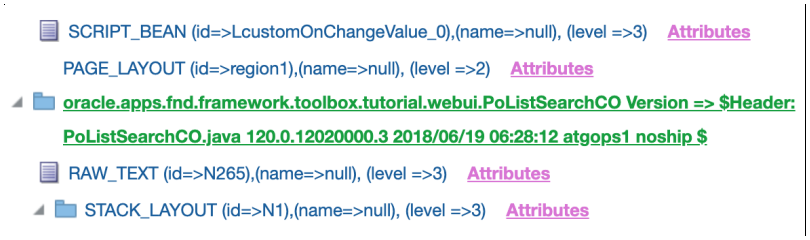
Webbeans Tab

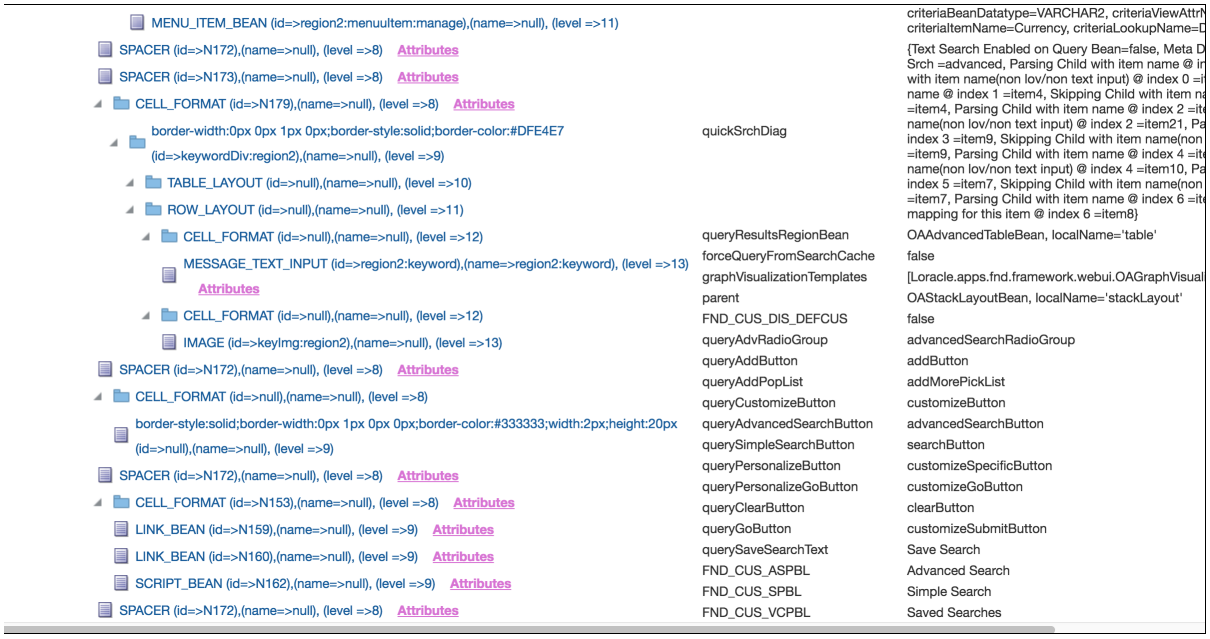


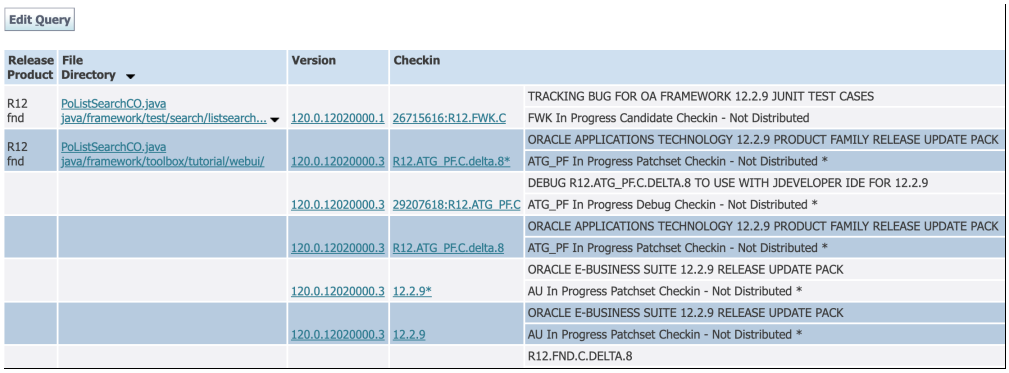
This is a most powerful tab that renders the complete web bean hierarchy of the page. This includes web beans that product developer added and also theother web beans added as a part of its code.

It also displays the properties of the web beans.

The controllers at the region levels are listed with a link.

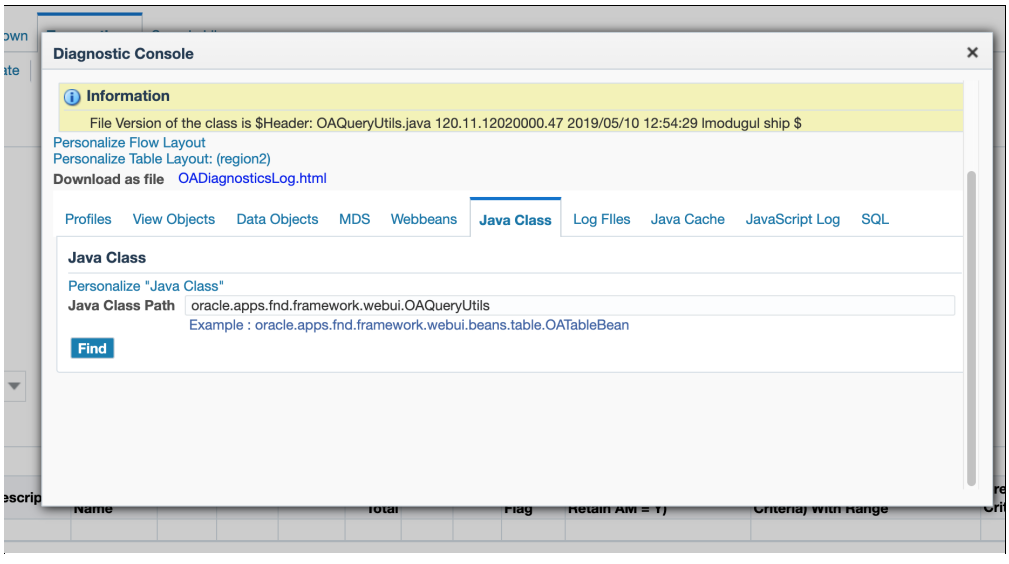






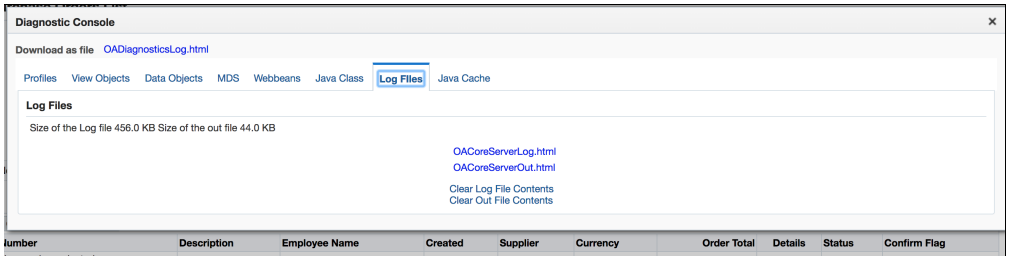
Java Class Tab

Allows you to find the Java class path and version information quickly.

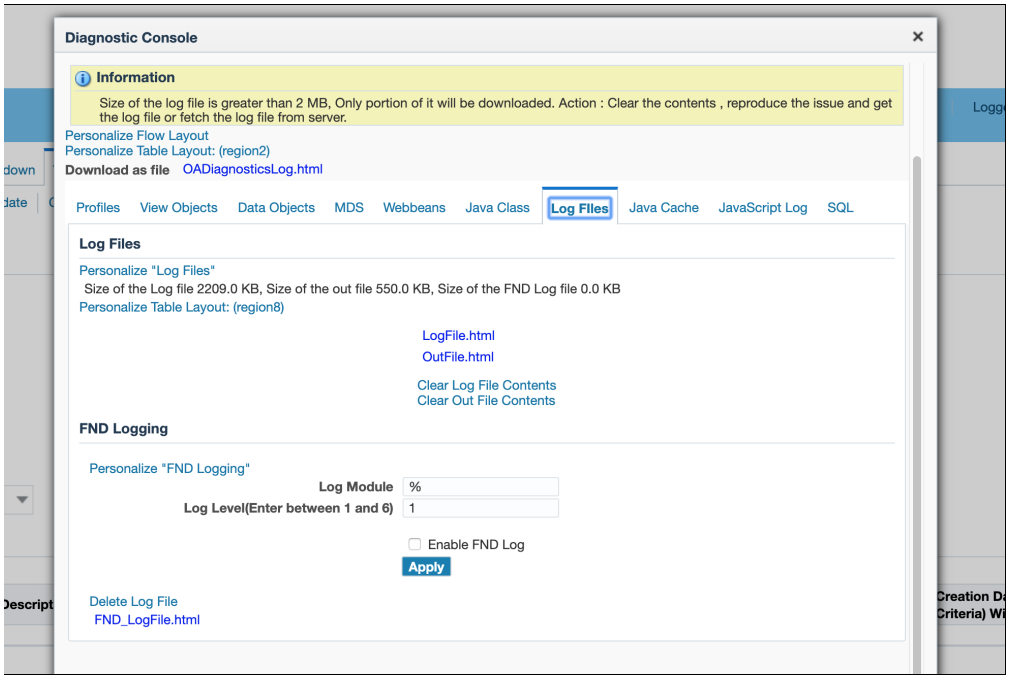


Log Files

This screen shows oacore log and out files. Once log and out files are greater than a certain size, the file content is truncated.

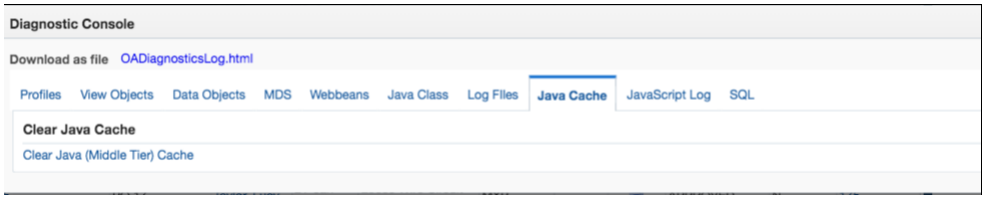


There is an option to clear the log and out files.



Java Cache

The Java cache can be cleared with the click of a button.



Diagnostics Log File - Download as HTML

All this information can be downloaded as HTML with the click of a link.

lastSortState_/oracle/apps/fnd/framework/toolbox/tutorial/webui/PoListSearchPG_ResultsTable	ascending
_AM_TX_ID_COUNTER	10

Page Context values

Page Path

/oracle/apps/fnd/framework/toolbox/tutorial/webui/PoListSearchPG

Shared Region Paths

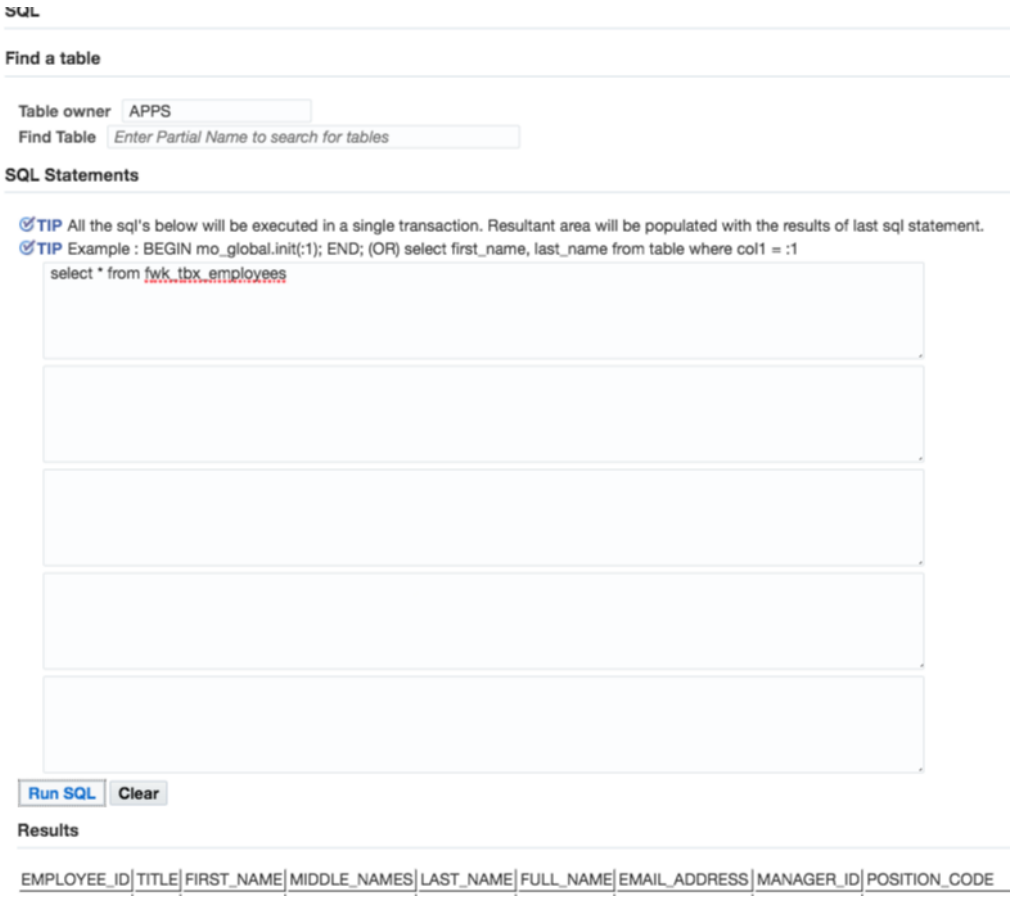
/oracle/apps/fnd/framework/webui/OAFooter
/oracle/apps/fnd/framework/toolbox/lov/webui/PurchaseOrdersLovRN

Page Personalization Paths

/oracle/apps/fnd/framework/toolbox/tutorial/webui/customizations/user/1006810/PoListSearchPG
/oracle/apps/fnd/framework/toolbox/tutorial/webui/customizations/site/0/PoListSearchPG

Shared Region Personalization Paths

Diagnostic Console - SQL



Diagnostic Console - JS

4/30/2019, 9:08:14 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:08:37 AM	encoded viewname = MicroEdgeElectronics Records
4/30/2019, 9:08:47 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:08:57 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:09:02 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:09:10 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:09:25 AM	encoded viewname = Michael Records
4/30/2019, 9:34:39 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:34:50 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:35:11 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:35:34 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:36:05 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 9:37:37 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 10:10:47 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:11:17 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:12:53 AM	Multi Select LOV created for LOV -- Value_0
4/30/2019, 10:13:06 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:14:49 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:15:48 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:15:48 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:17:12 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:17:12 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:17:42 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:17:42 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:18:15 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:18:15 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:19:05 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:19:05 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:19:17 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:19:17 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:19:47 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]
4/30/2019, 10:19:47 AM	----[object HTMLDocument],,,,[object HTMLBodyElement]

Event and Controller Flow Comparison Between List Search and Query Bean

Note: In List Search mode, `getCurrentPanel ()` always returns CUSTOMIZED (Views panel).

Feature	Event Raised (List Search)	Previous Mode Flow	List Search Mode Flow	New Public APIs
Change the view from saved searches dropdown.	queryBean.getPersonalizeGoButtonName()	PFR of query bean page.	1. PFR of query bean page. 2. Redirect to same page. 3. PR of query bean page.	N/A
Click Go button to filter the results.	queryBean.getGoButtonName()	PFR of query bean page.	PFR of query bean page.	N/A
Manage Saved Search Launch.	Internal OA Framework Popup Event.	PFR of query bean page. Forward request to personalize saved searches page.	1. PFR of query bean page. 2. If no errors, launch popup from client side. 3. Add a filter and remove a filter. 4. Partial Page Rendering event on query bean page.	N/A
Manage Saved Searches - OK/Cancel button click.	Internal OA Framework Popup Event.	PR of Query Bean page coming from personalize saved search page OK/Cancel.	PR of Query bean page while coming from Manage Popup OK/Cancel.	N/A
Save a new Search.	queryBean.getSaveSearchButtonName() PFR	PFR of Query bean page Get request issued for create view page.	PFR of query bean page Popup (create Saved Search) launched once the response comes back.	N/A
Reset a filtered saved search.	queryBean.getPersonalizeGoButtonName()	N/A	1. PFR of query bean page. 2. Redirect to same page. 3. PR of query bean page.	N/A
Add a filter.	queryBean.getAddButtonPoplistName()	PFR of Query bean page Redirect back to query bean page, PR of query bean page.	1. PFR of Query bean page. 2. Redirect back to query bean page. 3. PR of query bean page.	N/A
Remove a filter.	Internal OA Framework event name and source name.	N/A	1. PFR of Query bean page. 2. Redirect back to query bean page. 3. PR of query bean page.	Two new APIs in OAQueryBean: • public boolean isSavedSearchRemoveFilterEvent(); • public boolean getSavedSearchRemoveFilterSource();

Related Information

- BLAF UI Guidelines
  - Buttons (Links)

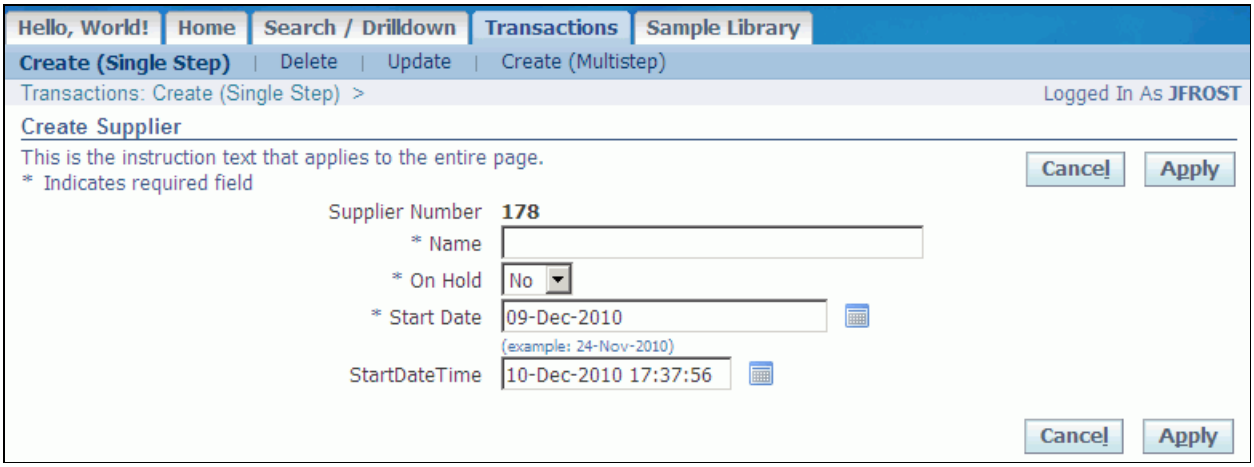
Locator Element: Breadcrumbs

Overview

As described in the Oracle Browser Look and Feel (BLAF) UI Guideline: Locator Element (Breadcrumbs), breadcrumbs are a series of links that display the user's current location within the application page hierarchy.

Figure 1, from OA Framework's ToolBox Tutorial, shows a typical breadcrumbs example. In this case, the user selected the Transactions tab and the Create (Single Step) horizontal navigation menu entry to display a Suppliers search page. In the Suppliers page, the user selected the Create Supplier button to navigate to the current displayed page.

Figure 1: OA Framework ToolBox Tutorial breadcrumbs example.



Note: Previously, the final item in the breadcrumbs list was an unlinked entry for the current page as shown in Figure 2. Now, the breadcrumbs list no longer displays the unlinked entry for the current page (see Figure 1 above). However, OA Framework still appends this item to its internal in-memory list so any preexisting code that references it will not break.

Figure 2: Example of previous breadcrumbs list with unlinked item for current page.



When the user navigates from a top-level page down through the hierarchy, the breadcrumbs reflect the page hierarchy in relation to the top-level page. This lets the user identify their location within the application, and affords a convenient navigation tool to intermediate pages in the hierarchy between the current page and the top-level page.

While breadcrumbs *do* provide a certain amount of history of where the user has been, they are somewhat poorly named since they are really intended to help the user locate themselves within the overall information architecture hierarchy (where they are vis-à-vis the top-level menus). That means they effectively "start over" when the user navigates from one top-level task to another. They do not drag on endlessly behind the user showing every place she has been.

When breadcrumbs render, they *always* begin with the top-level identifier such as a tab or global button as appropriate.

- If the top-level identifier is a global button, the first breadcrumb reflects that global button's label as follows:

  Global Button (for example, Shopping Cart)

- If the top-level identifier is a tab/horizontal navigation selection, the first breadcrumb appears as follows:

  Tab : Horizontal Navigation (for example, Order Status : Receiving)

- If the page is on a side navigation menu that is associated with a selected tab/horizontal navigation, the first breadcrumb appears as follows:

  Tab : Horizontal Navigation: Side Navigation

- Breadcrumbs *never* appear on a top-level page (a page that displays as a result of selecting a global button, a tab or a horizontal navigation or side navigation with no parent tab). They are only displayed when you navigate to another page from that top-level page.

- When the user changes top-level identifier (for example, switches from Tab A to Tab B or selects a global button), the breadcrumbs reflect this change.

Implementation

You do not need to explicitly add a breadcrumbs region or item to your page. Assuming you follow the steps described below, OA Framework automatically manages the breadcrumb history in memory and displays them on your page. (OA Framework creates an `oracle.apps.fnd.framework.webui.beans.nav.OABreadCrumbsBean` and adds it to your **pageLayout**).

Step 1: Set the `Title` property in your **pageLayout** region. This value is used for the breadcrumb text (label). See the Primary Header (Page Title) topic in the Headers and Subheaders document for additional information about setting the page title.

When modifying the page title programmatically or binding the page title to a view object value, use the following code to have the breadcrumb text of the current page use the dynamic page title value:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  // 1. Your controller
  //    i) directly sets the page title attribute value to a string value or
  //    ii) indirectly sets the page title attribute value by resolving the
  //    view object value that determines the page title
  //    -- for the latter case (binding case), you would be initializing the
  //    view object row and attribute values in this controller code portion.

  // Approach i:
  OAPageLayoutBean pageLayoutBean = (OAPageLayoutBean)webBean;
  // If this code is in a non-pageLayout region's controller:
  // OAPageLayoutBean pageLayoutBean = pageContext.getPageLayoutBean();
  pageLayoutBean.setTitle("<Page Title>");

  // Approach ii:
  // You should invoke a method on the AM that performs VO initialization
  // operations as below:
  // ...
  // pageTitleVORow.setAttribute(...);
  // pageTitleVO.insertRow(pageTitleVORow);
  // ...

  // 2. Call prepareForRendering() ONLY IF you perform #1 operations in
  //    the controller of a non-pageLayout region.  An ideal coding practice is
  //    to modify/resolve the properties and data of a region in the
  //    corresponding controller instead of in the child region's controller.
  //
  //    prepareForRendering() allows dependent property values to be resolved
  //    when the driving property value is changed/resolved.  Breadcrumb's text
  //    is derived from the page title.

  // pageLayoutBean.prepareForRendering(pageContext);
}
```

Note: If you want the breadcrumb link text to differ from the page title text, such as displaying a shortened version of the page title text as the breadcrumb text, programmatically override the breadcrumb text as described in the Runtime Control section below.

Tip: If you reuse a page multiple times in a menu (each instance has a different page title) and you want to show multiple breadcrumb instances for that dynamic page in a single breadcrumb list (Page 1 > Page 2 > Page 3, where pages 1 - 3 actually correspond to the same **pageLayout** region), do *not* set the title declaratively. In this case set the page title in your **pageLayout** controller.

Step 2: Set the `addBreadCrumb` URL parameter as you navigate between pages to tell OA Framework to add the target page to the in-memory breadcrumb list that it maintains (you can set this in any of the forward/redirect methods, or explicitly set it as a URL parameter.

For example, if you define an `OAButtonBean` to perform simple navigation to another page that should display breadcrumbs, you would set the `Destination URI` property to `OA.jsp?page=/oracle/apps/dem/employee/EmpDetailsPG&addBreadCrumb=Y`.

Tip: If you set the `Destination URI` property declaratively without specifying the `addBreadCrumb` parameter, OA Framework interprets this as **N**. Alternatively, you can set this value when performing a JSP forward as shown below.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
```

```
  super.processFormRequest(pageContext, webBean);
  ...

  pageContext.setForwardURL(< some page >,
                            KEEP_MENU_CONTEXT, // Keep current menu context
                            null,
                            pageParams,// additional URL parameters
                            true, // Retain AM
                            ADD_BREAD_CRUMB_NO, // Do not display breadcrumbs
                            IGNORE_MESSAGES);

}
```

Tip: If you call the `setForwardURL()` or `forwardImmediately()` methods without specifying a value for the `addBreadCrumb` parameter, OA Framework always interprets this as **N**.

Valid values for this parameter include:

URL Parameter Value	Corresponding OAWebBeanConstant	Behavior
addBreadCrumb=Y	ADD_BREAD_CRUMB_YES	A breadcrumb item is created for the target page URL (page URL containing `addBreadCrumb=Y`) and added to the in-memory breadcrumb list (internal storage for breadcrumbs information). If the in memory breadcrumb list already contains a breadcrumb item with the same page title, the breadcrumb list is rewound to the matching breadcrumb item. In other words, all the items after the matching item will be removed from the list and the breadcrumb (page title) will not be added twice. The associated URL value of the current page's breadcrumb item will *not* be changed. If two pages have the same breadcrumb label (defaulted to page title except for the first breadcrumb that is based on the menu selection), but they are associated with different applications, they will be treated as different pages with different breadcrumbs.
addBreadCrumb=N	ADD_BREAD_CRUMB_NO	The in-memory breadcrumb list is cleared.
addBreadCrumb=S	ADD_BREAD_CRUMB_SAVE	The in-memory breadcrumb list is saved/preserved as it is with no changes.
addBreadCrumb=RS	ADD_BREAD_CRUMB_RESTART	The in-memory breadcrumb list is cleared and then the target page URL is immediately added to the newly empty breadcrumb list.

addBreadCrumb=RP	ADD_BREAD_CRUMB_REPLACE	Searches for the first breadcrumb item in the in-memory breadcrumb list that has the same breadcrumb label and application ID as the one you are trying to add to the list. If a matching item is found, the URL value of the breadcrumb item found is *replaced* with the target page URL value, and the breadcrumb list is rewound to the matching breadcrumb item. In other words,all the items *after* the matching item are removed from the list Otherwise (if a matching item is not found), a new breadcrumb item is added to the existing breadcrumb list. This is the same behavior as `addBreadCrumb=Y`.

Before reviewing the usage examples below, read the General Breadcrumbs Behavior section to understand how menu breadcrumbs are added and how the in-memory breadcrumb list gets displayed in the UI. By default, all the breadcrumbs in the in-memory list are displayed in the UI, except for the last breadcrumb item in the in-memory list.

Note: See Control Visual Properties for more details on how to further control whether or not the in-memory breadcrumb list gets displayed in the UI.

Usage Example #1

Note: In each of the following use case examples, assume that the `addBreadCrumb` value is set when navigating to the target page.

Navigate From Page	Navigate To Page	addBreadCrumb Value
User selects **Tab 1** to navigate to Page 1.		
Page 1	Page 2	Y
Page 2	Page 3	Y
Page 3	Page 4	N

When **Page 1** renders, no breadcrumbs are displayed (breadcrumbs do not render on top-level menu pages).

When **Page 2** renders, the breadcrumbs display as:

Tab 1 >

When **Page 3** renders, the breadcrumbs display as:

Tab 1 > Page 2 >

When **Page 4** renders, no breadcrumbs display (the in-memory list is cleared).

Usage Example #2

Navigate From Page	Navigate To Page	addBreadCrumb Value
User selects **Tab 1** to navigate to Page 1.		
Page 1	Page 2	Y
Page 2	Page 3	Y
Page 3	Page 3	S
Page 3	Page 4	Y

After completing each navigation in this example, the breadcrumbs display as:

Tab 1 > Page 2 > Page 3 >

Note that Page 3 is not repeated in the breadcrumb list.

Usage Example #3

Note: In this example, assume that the developer performs a JSP forward from Page 3 -> Page 3 to programmatically display **someRegion2** instead of **someRegion1**.

Navigate From Page	Navigate To Page	addBreadCrumb Value
User selects **Tab 1** to navigate to Page 1.		
Page 1	Page 2	Y
Page 2	Page 3	Y
Page 3	Page 3 (dynamically display "someRegion1")	Y
Page 3	Page 3 (dynamically display "someRegion2")	RP
Page 3	Page 4	Y

Tab 1 > Page 2 > Page 3 >

Note: Page 3's URL should have a URL parameter to indicate that **someRegion2** should display.

Usage Example #4

Navigate From Page	Navigate To Page	addBreadCrumb Value
User selects Tab 1 to navigate to Page 1.		
Page 1	Page 2	Y
Page 2	Page 3	Y
Page 3	Page 4	RS
Page 4	Page 5	Y
Page 5	Page 6	Y

When Page 6 renders:

- Assuming "Tab 1" is selected when Page 4 renders, the breadcrumbs display as:

  Tab 1 > Page 4 > Page 5 >

  The in-memory list of breadcrumbs restarts with the navigation to Page 4. Note that OA Framework automatically adds the breadcrumb for the menu entry "Tab 1" to reflect the current page hierarchy.

- If no menu is selected when Page 4 renders, such as if Page 4 has no associated menu, the breadcrumbs display as:

  Page 4 > Page 5 >

Miscellaneous Rules

- **Rules for Menu Breadcrumbs:** Before reviewing the rules below, read the General Breadcrumbs Behavior section to understand menu breadcrumb behavior.
  - If you drill down from Page 1 (the first page, which is normally rendered with a menu selection) to Page 2, and the first page does *not* have a menu item highlighted, but you want to include Page 1 in the breadcrumb list, then specify `addBreadCrumb=Y` for Page 1. OA Framework shows the breadcrumb for Page 1 with its page title as the breadcrumb label.

    Note: Every page should sit in a menu hierarchy, so normally you should not encounter this exception case.

  - To navigate from a drilldown page to the top-level page that corresponds to the menu breadcrumb, use `addBreadCrumb=N` or do not specify any `addBreadCrumb` URL parameter. This clears the breadcrumbs when the top-level page is rendered.
  - Navigating back to the top-level page with `addBreadCrumb=Y` should be avoided because, in this case OA Framework tries to add a breadcrumb based on the top-level page's page title. However, if the page title differs from the menu breadcrumb's label, which is based on the selected menu items' labels, OA Framework adds a new breadcrumb based on the page title instead of rewinding and clearing the breadcrumb list.

Runtime Control

Programmatically change breadcrumb properties in the target page's controller.

Access the OABreadCrumbsBean

To access the `OABreadCrumbsBean` directly, call `getBreadCrumbsLocator()` on your `OAPageLayoutBean`.

Usage Notes

- As a rule, control the breadcrumbs using the URL parameter described above. Use direct manipulation of the `OABreadCrumbsBean` only as a last resort if, for example, your page is dynamic and it's necessary to ensure correct behavior.
- Use the `OABreadCrumbsBean` methods to access/modify the breadcrumbs web bean structure (including setting the bread crumbs link text). Do *not* use methods in the UIX superclasses such as `BreadCrumbsBean` or `LinkContainerBean`. The `OABreadCrumbsBean` class synchronizes breadcrumb web bean changes in memory. Using methods in the parent classes breaks the association between the in-memory list and breadcrumbs web bean.

  Exceptions: You may use the `getOrientation()` and `setOrientation()` methods in the super class (`BreadCrumbsBean`).
- Do *not* use the `setLocation()` method in `OAPageLayoutBean`. OA Framework calls this method internally to render breadcrumbs in the page layout UI. Setting the breadcrumbs locator to point to a new `OABreadCrumbsBean` object breaks the association between the in-memory list and breadcrumbs web bean.
- Setting the page title to an empty string in your controller has no effect on the breadcrumbs. The breadcrumb list will *not* be modified since breadcrumbs cannot contain empty labels.

Control Visual Properties

To hide breadcrumbs while preserving the in-memory list so that they can be displayed on another page, call `setBreadCrumbEnabled(false)` on your `OAPageLayoutBean`.

Tip: If you try to modify the breadcrumb properties from a controller associated with a region beneath the `OAPageLayoutBean` in the page hierarchy, you must remember to call `pageLayoutBean.prepareForRendering(pageContext)` *after* you modify the breadcrumbs as shown in the setTitle() example above. Note that the practice of modifying ancestor web beans from descendent controllers is a violation of the OA Framework Controller Coding Standards, but this tip is included since so many people make this particular mistake.

To control whether the breadcrumb list is displayed horizontally or vertically, call use the `setOrientation()` method on the `BreadCrumbsBean` super class.

General Breadcrumbs Behavior

- **Menu Breadcrumb:**
  To represent the user's current location within the application page hierarchy in the breadcrumb list displayed, the first breadcrumb item breadcrumb list needs to reflect the currently active menu hierarchy. OA Framework achieves this by automatically adding a menu breadcrumb, which concatenates the selected menu item labels.
  - **Menu breadcrumb text (label):**
    The menu breadcrumb text typically has the format of ***<Selected Tab> : <Selected Subtab> : <Selected Side Navigation>*** or ***<Selected Global Button>***. Basically, it is a concatenation of the selected (highlighted) menu item labels. When a drilldown occurs from Page 1 to Page 2 with `addBreadCrumb=Y` (or `RS`, `RP`) specified on Page 2's URL, and the in-memory breadcrumb list is initially empty, OA Framework automatically adds a menu breadcrumb for Page 1 in the in-memory breadcrumb list before it adds the breadcrumb for Page 2, based on its page title, to the in-memory list.

    The text of the menu breadcrumb text for Page 1 corresponds to the menu item labels that were shown selected in Page 1 (the last page visited) before the drilldown occurred. If the last page visited before the drilldown did not have any menu items selected (highlighted), OA Framework does not add a menu breadcrumb for that last page visited.

    Note on Menu Selection (Highlighting): If a menu item such as a tab, subtab, side navigation, or global button is highlighted or selected in a page and its URL matches the page URL, the selections are carried over to the next pages until you click on another menu item.
  - **Menu breadcrumb URL:**
    The menu breadcrumb URL corresponds to the URL of the last page visited before the drilldown event. Using the above example with Page 1 and Page 2, it would correspond to Page 1's URL with the addBreadCrumb parameter removed.
  - **Menu breadcrumb enforcement:**
    When OA Framework adds the second breadcrumb item into the in-memory breadcrumb list, and the label of the existing first breadcrumb is different from the menu selection label (that is, the first breadcrumb is not based on the menu selection), OA Framework replaces the first breadcrumb with the menu selection breadcrumb. This ensures that the first breadcrumb in the list always starts with a menu selection hierarchy if a menu selection is present.

- **Breadcrumb clearing behavior:**
  The breadcrumb list is cleared whenever a menu selection breadcrumb is selected.

  Exceptions:

  - The breadcrumb list is *not* cleared when menu links in the side navigation are selected.
  - If the menu link performs a Javascript submitForm action or PPR partial client action, the breadcrumb list is *not* cleared when you select such menu links that submit the form. This is because the web bean hierarchy only changes for non-form-submit requests. If you must clear the breadcrumb list, you may have the form submit action forward back to the current page with the `ADD_BREAD_CRUMB_NO` parameter.

- **Programmatically changing properties of the menu breadcrumb:**
  You should not change the menu breadcrumb that was derived by OA Framework. However, if you do need to change the properties you must change the menu breadcrumb properties in the drilldown page because that is when OA Framework adds the menu breadcrumb.

  For instance, suppose you have the page flow, Page 1 -> ... Page M -> Page (M + 1) ... Page N. If you start or restart the breadcrumb list on Page (M + 1) with `addBreadcrumb=Y`, that is, in the middle of the page flow as opposed to at the beginning of the page flow launched from the global Home page, the URL of the menu breadcrumb on Page (M + 1) corresponds to Page M's URL. To change this URL to correspond to (Page 1's URL, you should change the menu breadcrumb properties programmatically in the drilldown page. (Normally, OA Framework does not expect this type of page flow.) See the Runtime Control section for more information on how to make programmatic changes on the breadcrumbs.

- See Miscellaneous Rules - Rules for Menu Breadcrumbs for additional rules of which you should be aware.

  Note: The current OA Framework menu breadcrumb behavior and BLAF UI standards have minor discrepancies for less common page flow scenarios, therefore, if the behavior of the menu breadcrumb rendered by OA Framework is noticeably un-intuitive, use the programmatic workarounds suggested above. Due to backward-compatibility issues, OA Framework will not add or revise breadcrumbs behavior unless the change has a localized impact.

- If the in-memory breadcrumb list contains only one item, it will not be shown in the UI. So, for example, if you select a tab and a page renders, breadcrumbs do not display for that page.

  Note: The *in-memory* breadcrumb list always contains one more item than the *displayed* breadcrumb list. So, for breadcrumbs to be displayed on your page, the in-memory list must contain at least two items.

- When you select a breadcrumb link, the target page is built from scratch. OA Framework does not display the page content cached by the browser.

- Each breadcrumb item in the in-memory breadcrumb list is uniquely identified by its displayed breadcrumb label/text (defaulted to page title except for the first breadcrumb, which is based on the menu selection) within an application. If two pages have the same breadcrumb label, but are under different applications, they will be treated as different pages with different breadcrumbs. Within the same application, breadcrumbs do not repeat in the displayed breadcrumb list.

- Pages without page titles or empty page title strings are not added to the breadcrumb list.

- Trains and breadcrumbs cannot be used together (see the BLAF UI Guidelines for instructions on when to use a train, and when to use breadcrumbs). If you try to enable breadcrumbs on a page with a train by setting the `addBreadCrumb` URL parameter to Y, OA Framework ignores the instruction and does not modify the in-memory breadcrumb list. The train takes precedence.

- A breadcrumb list is associated with a given transaction id. Whenever you return to the E-Business Home page and revisit a page or relogin and revisit a page, a new transaction id is assigned. In this case, a new breadcrumb list is started. Therefore, there is no need to specify `addBreadCrumb=RS` on the portal menu option function JSP URL to restart the list.

- According to the Oracle UI team, the "Return to" link does not have a direct correlation with breadcrumbs. It should direct the user to some logical point in the page flow rather than the last URL, and should be defined at design time.

Personalization Considerations

- See a summary of Locator Element: Breadcrumbs personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key Locator Element: Breadcrumb issues with suggested workarounds if available.

Related Information

- **BLAF UI Guidelines**
  - Locator Element (Breadcrumbs)
- **Javadoc**
  - oracle.apps.fnd.framework.webui.beans.nav.OABreadCrumbsBean
  - oracle.cabo.ui.beans.nav.BreadCrumbsBean
  - oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean

.

Locator Element: Page/Record Navigation

Overview

As described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Locator Element: Page/Record Navigation, there is a single navigation bar control that can be configured and used in two seemingly different contexts. This document describes the use of the `oracle.apps.fnd.framework.webui.beans.nav.OANavigationBarBean` for page navigation, and record navigation.

Support for Touch Devices

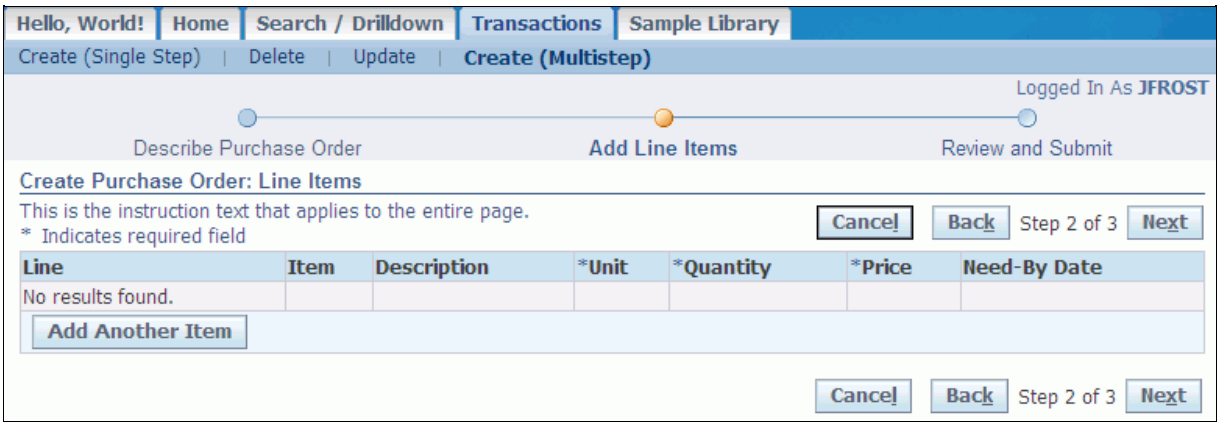
As of Release 12.2.4, OA Framework provides gestures support for page and record navigation. For touch devices, you may **Swipe-Left** or **Swipe-Right** to navigate to the next or previous page and **Swipe-Up** or **Swipe-Down** to navigate to the next or previous set of records in a table. Refer to the Gestures Support topic for additional Usage Restrictions specific to navigating between the pages of a train.

Page Navigation

When used for page navigation, the `OANavigationBarBean` renders a Back and Next button with locator information ("Step X of Y") between them. You can also configure this control to display a poplist instead of static locator information so users can quickly navigate to a specific page in the sequence.

Note: If there are only two visible nodes/links in the navigation bar, the button text will appear as 'Continue' instead of 'Next'.

Figure 1: Example of page navigation buttons in multistep transaction.



Declarative Implementation: "Basic" Navigation Bar

This section describes how to add a "basic" navigation bar to your page as shown in Figure 1 above. See the Declarative Implementation: Interactive Navigation Bar section if you want a poplist to render between the Back and Next buttons so users can quickly navigate to a specific page.

Note that this example includes the addition of Cancel and Submit buttons as these are typically required with a page navigation control (the page navigation control itself renders Back and Next buttons with locator information between them).

Step 1: Build a shared region to include the page navigation control and the Cancel and Submit buttons. Specify a standards-compliant `ID`, and set the `Style` to **pageButtonBar**.

Step 2: Select the region you created in Step 1, right-click and select **New > Item** to add a Cancel button (see

Step 3: Select the **pageButtonBar** region again, right-click and select **New > Region** to add the page navigation. Specify a standards-compliant `ID`, set the `Style` to **navigationBar** and set the `First Step` and `Last Step` values based on the number of pages in the linear page flow.

Step 4: Select the **pageButtonBar** one final time, right-click and select **New > Item** to add a Submit **submitButton**.

Step 5: Add the **pageButtonBar** to each page included in the page flow.

- Step 5.1 Select the **pageLayout** region in the Structure pane, right-click and select **New > Region**.
- Step 5.2 Assign a standards-compliant `ID` to the region and the `Extends` property to the fully qualified name of the shared **pageButtonBar** region (for example, **/oracle/apps/dem/employee/webui/EmpTrainFooterRN**).

Repeat steps 5.1 and 5.2 for each page in the page flow.

Runtime Control: "Basic" Navigation Bar

When working with the "basic" navigation bar, you need to address two things. First, you need to property initialize the component on the page to reflect the current page, and second you need to handle the button navigation.

Initialize

When you navigate to the first page in a page flow, you must add a URL parameter that you can check to render the navigation bar bean correctly. For example:

```
import com.sun.java.util.collections.HashMap;
...

public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{

   if ("UPDATE".equals(pageContext.getParameter("empEvent")))
   {

     HashMap pageParams = new HashMap(1);

     pageParams.put("empStep", "1");

     pageContext.setForwardURL("OA.jsp?page=/oracle/apps/dem/employee/webui/EmpDescPG",
                               null,
                               OAWebBeanConstants.KEEP_MENU_CONTEXT,
                               null,
                               pageParams,
                               true, // Retain AM
                               OAWebBeanConstants.ADD_BREAD_CRUMB_NO, // Do not display breadcrums
                               OAWebBeanConstants.IGNORE_MESSAGES);

  }

}
```

Then, in a controller associated with your shared region containing the `OANavigationBarBean`, add code like the following to your `processRequest()` method. This checks the URL parameter specifying the current page step, and sets this accordingly. Note that we also use this parameter to ascertain whether the submit button should render (it should appear only on the last page of the flow).

```
import oracle.apps.fnd.framework.webui.beans.form.OASubmitButtonBean;
import oracle.apps.fnd.framework.webui.beans.nav.OANavigationBarBean;

...


public void processRequest(OAPageContext pageContext, OAWebBean webBean)
  {

    super.processRequest(pageContext, webBean);


    OANavigationBarBean navBean =
      (OANavigationBarBean)webBean.findChildRecursive("NavBar");

    // Determine which page we're on so we can set the selected
    // value. Each time we navigate to and within the flow, the
    // URL includes a parameter telling us what page we're on.

    int step = Integer.parseInt(pageContext.getParameter("empStep"));

    navBean.setValue(step);

    // Figure out whether the "Submit" button should be rendered
    // or not; this should appear only on the final page (Step 3).

    OASubmitButtonBean submitButton =
      (OASubmitButtonBean)webBean.findIndexedChildRecursive("Submit");

    if (submitButton == null)
    {
      MessageToken[] tokens = { new MessageToken("OBJECT_NAME", "Submit") };
      throw new OAException("ICX", "FWK_TBX_OBJECT_NOT_FOUND", tokens);
    }
```

```
    if (step != 3)
    {
      submitButton.setRendered(false);
    }

} // end processRequest()
```

Handle Navigation Events

Note: The following simple example shows how to navigate between the pages with static code. See Declarative Pageflow Using Workflow for instructions on implementing page navigation using Oracle Workflow.

Add the following `processFormRequest()` method to your **EmployeeUpdateFooterCO**. This ascertains which `OANavigationBarBean` button was selected so the correct destination page can be rendered. It also displays a Confirmation message when the user selects the Submit button.

```
import com.sun.java.util.collections.HashMap;


import oracle.apps.fnd.common.MessageToken;
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;

import oracle.apps.fnd.framework.OAException;
import oracle.apps.fnd.framework.webui.OADialogPage;
...

public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{

  super.processFormRequest(pageContext, webBean);


  if (pageContext.getParameter("Submit") != null)
  {

    // Assuming the "commit" succeeds, we'll display a Confirmation
    // dialog that takes the user back to the main Employee search.

    MessageToken[] tokens = { new MessageToken("EMP_NAME",
      employeeName) };

    OAException confirmMessage = new OAException("ICX",
      "FWK_TBX_T_EMP_UPDATE_CONFIRM", tokens);

    OADialogPage dialogPage = new OADialogPage(OAException.
                                          CONFIRMATION,
                                          confirmMessage,
                                          null,
                                          APPLICATION_JSP + "?OAFunc=FWK_TBX_LABS_EMPLOYEES2",
                                          null);

    // Note that we release the root "UI" application module
    // so we can correctly handle any subsequent "Back" button
    // navigation and attempts to resubmit the PO transaction.

    pageContext.releaseRootApplicationModule();
    pageContext.redirectToDialogPage(dialogPage);

  }
  else if (GOTO_PARAM.equals(pageContext.getParameter(EVENT_PARAM))
  "NavBar".equals(pageContext.getParameter(SOURCE_PARAM))
  {
    // We use the parameter "value" to tell use the number of
    // the page the user wants to visit.

    // Also note the check of "source" above to ensure we're
    // dealing with the page-level navigation here and not
    // table-level navigation which is implemented with the
    // same Bean configured differently.

    int target = Integer.parseInt(pageContext.getParameter(VALUE_PARAM));

    String targetPage;

    switch(target)
```

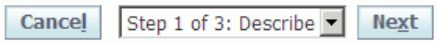
```
    {
      case 1: targetPage = "/oracle/apps/dem/employee/webui/EmpDescPG"; break;
      case 2: targetPage = "/oracle/apps/dem/employee/webui/EmpAssignPG"; break;
      case 3: targetPage = "/oracle/apps/dem/employee/webui/EmpReviewPG"; break;
      default: throw new OAException("ICX", "FWK_TBX_T_EMP_FLOW_ERROR");
    }

    HashMap pageParams = new HashMap(2);
    pageParams.put("empStep", new Integer(target));

    pageContext.setForwardURL("OA.jsp?page=" + targetPage,
                              null,
                              OAWebBeanConstants.KEEP_MENU_CONTEXT,
                              null,
                              pageParams,
                              true, // Retain AM
                              OAWebBeanConstants.ADD_BREAD_CRUMB_NO, // Do not display breadcrumbs
                              OAWebBeanConstants.IGNORE_MESSAGES);

  }
} // end processFormRequest()
```

Declarative Implementation: Interactive Navigation Bar

The interactive navigation bar lets users navigate directly to a selected page in the flow as shown in Figure 2. Note that you must use an interactive navigation bar if you are coupling it with an interactive train.

Note: If a Train is associated with a Navigation Bar and the RENDERED property of a step in the Train is set to false, it is the responsibility of the developer to explicitly ensure the RENDERED property for that step in the associated Navigation Bar is also set to false. A mismatch between the Train and the Navigation Bar is created if RENDERED properties are not synchronized. For example, if the second step of a three-step Train is not rendered, the Train will display only two steps (Step 1 and Step 3), and the Navigation Bar will display all three steps.

Figure 2: Example of an interactive page navigation bar.



To create an interactive navigation bar, follow the declarative instructions above for the "basic" navigation bar. Then, select the **navigationBar** region in the JDeveloper Structure pane, right-click and select **New > Item**. Assign the item a standards-compliant `ID`, set its `Style` to **link**, set the `Text` to the value you want to display in the poplist and set the `Destination URI` to the fully-qualified name of the target page in the flow (for example, **/oracle/apps/dem/employee/webui/ EmpAssignPG**). Repeat as needed for each page.

Warning: Make absolutely certain that the navigation bar links match the train step links if you are using these components in tandem. Also make sure that the displayed text matches.

Runtime Control: Interactive Navigation Bar

When you add an interactive navigation bar to your page, the OA Framework handles all the navigation for you automatically; you don't have to do anything.

If you couple your page navigation component with supplemental buttons that must render or not based on the current page step (for example, a Submit button in a multipage transaction), add the following code to your controller associated with the shared navigation region:

```
import oracle.apps.fnd.framework.webui.beans.form.OASubmitButtonBean;
import oracle.apps.fnd.framework.webui.beans.nav.OATrainBean;

...


public void processRequest(OAPageContext pageContext, OAWebBean webBean)
  {

    super.processRequest(pageContext, webBean);


    // Figure out whether the "Submit" button should be rendered or not;
    // this should appear only on the final page (Step 3).

    // The OATrainBean is a named component of the page layout, so we have
    // a special way of getting a handle to it (we can't "find" it like
    // we do for normal, indexed children that would be below the current
```

```
    // region in the hierarchy).

    OATrainBean trainBean =
      (OATrainBean)pageContext.getPageLayoutBean().getLocation();

    // You must call the following before getting the target page index.

   trainBean.prepareForRendering(pageContext);
   int step = trainBean.getSelectedTrainStepRenderedIndex();

   if (step + 1 != trainBean.getNumberOfRenderedTrainSteps())
   {
     OASubmitButtonBean submitButton =
       (OASubmitButtonBean)webBean.findIndexedChildRecursive("Submit");

     submitButton.setRendered(false);
   }

} // end processRequest()
```

Record Navigation

Under the covers, the same `OANavigationBarBean` that you use for page navigation is also used for record navigation in tables. Assuming you implement a **table** or **advancedTable** region, the OA Framework configures this for you automatically. There is no need for you to create or interact with this component.

Personalization Considerations

-

Known Issues

- None

Related Information

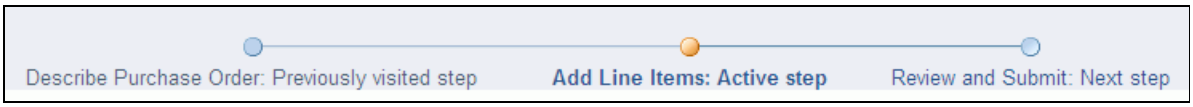
- BLAF UI Guidelines
  - Locator Element: Page/Record Navigation
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.nav.OANavigationBarBean`
- OA Framework ToolBox Tutorial / Sample Library
  - Update Lab

Locator Element: Train

Overview

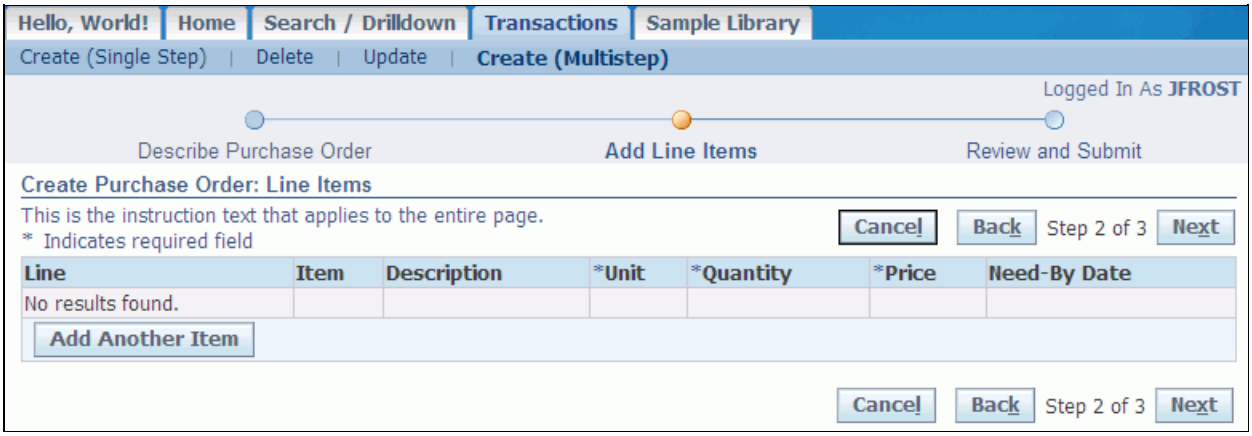
As described in the Oracle Browser Look-and Feel (BLAF) UI Guidelines: Locator Element (Train), the train is a graphical component used to show a user his current location in a linear process as shown in Figure 1.

Figure 1: Illustration of a basic train showing the previous, current, and next step visible state.



Per the UI Guidelines, all trains are implemented in conjunction with a page navigation component as illustrated in Figure 2 (there is one exception: if you leave the main page flow to perform a subordinate task you should display a disabled train coupled with simple Apply and Cancel buttons on the drilldown page). When present, the navigation bar component is synchronized with the train to ensure that both regions reflect the user's current location within the flow.

Figure 2: Example of a train and a page navigation component together on a page.



Contents

- Basic Train
- Interactive Train
- Train Within Train
- Support for Touch Devices
- Personalization Considerations
- Known Issues
- Related Information

Basic Train

This section describes the steps for implementing a basic train. See the Interactive Train section below for supplemental instructions on how to make the basic train interactive (so the user can navigate by selecting train nodes). See the Train Within Train instructions for representing a "subtask" within a linear train flow.

Declarative Implementation

Note: When you implement a train, you must also implement a page navigation component -- unless you are dealing with the drilldown exception described above. This document deals exclusively with the train; see Locator Element: Page/Record Navigation for corresponding implementation instructions.

Step 1: Create a shared train region for inclusion in all the pages comprising your multistep flow. Assign a standards-compliant `ID`, and set the `Style` to **train**.

Step 2: Add nodes to the train region for each page that you want to display.

- Step 2.1 Select the **train** region in the Structure pane, right-click and select **New > Link**.
- Step 2.2 Assign a standards-compliant `ID` to the link, enter the `Text` that you want to display below the train node (for example, "Assignments"), and set the `Destination URI` to the fully qualified page name that renders when this node is

active (for example, **/oracle/apps/dem/employee/webui/ EmpAssignPG**).

Repeat steps 2.1 and 2.2 for each node in the train.

Step 3: Add the shared train region to each page in the flow.

- Step 3.1 Select the **pageLayout** region in the Structure pane, right-click and select **New > Location**.
- Step 3.2 Assign a standards-compliant `ID` to the train node and set the `Extends` property to the fully qualified name of the shared train region (for example, **/oracle/apps/dem/employee/webui/EmpTrainRN**).

Repeat steps 3.1 and 3.2 for each page in the flow.

Runtime Control

If you want to get access to your train for any reason, use the following code in `processRequest()`:

```
OATrainBean train = (OATrainBean)pageContext.getPageLayoutBean().getLocation();
```

Interactive Train

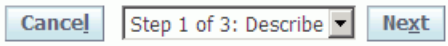
Interactive trains let users let users quickly navigate to any previous step, and one step forward, by selecting a train node (the nodes are implemented as links) as shown in Figure 3.

Figure 3 : Illustration of an interactive train.



Interactive trains should always be implemented in conjunction with the interactive page navigation element as shown in Figure 4.

Figure 4: Example of an interactive page navigation control.



Making the Train Interactive

To make a train interactive, select the train region in the JDeveloper Structure pane and set its `Allow Interaction` property to **true** (you can also set this property by calling `allowInteraction(true)` on the `oracle.apps.fnd.framework.webui.beans.nav.OATrainBean`). Then, add an updateable Next/Back locator as described in the Locator Element: Navigation document.

Warning: If you do not add this navigation control to your page, the OA Framework throws a Developer Mode exception if you work with this test mode enabled.

Handling a Step Selection

When a user selects a train node, the OA Framework automatically performs a JSP forward to the target page using the `Destination URI` specified for the link step. If you want to override this automatic behavior and handle the JSP forward yourself, call `setAutoRedirect(false)` on the `OATrainBean`. You can then handle the train step selection as described in the Basic Train section above.

When a user selects a step before the current step, the OA Framework does not alter the train state to make current step appear that it was not visited. For example, if an application displays a six-step train, and the user advances to Step 5 before returning to Step 2, the train renders with Steps 2, 3 and 4 appearing as visited. If the user changes data in Step 2 that invalidates the work done in Steps 3, 4, and 5, you should reset the train state these steps no longer appear visited. To do this, call `resetVisitedSteps()` on the `OATrainBean`. When you call this method, all steps after the current step appear "unvisited."

Synchronizing the Train and the Page Navigation

When the user navigates using the page navigation instead of the train, the OA Framework automatically synchronizes the train with the new target page. To achieve this, the train steps' `Destination URI` property (as specified in the Basic Train implementation Step 2.2 above) must match the `OANavigationBarBean` link steps.

Train Within Train

Some linear tasks require a "branch" to a subprocess. For example, while ordering office supplies in a requisition (a linear process

with a train), the user adds a business card item to her shopping cart. The shopping cart page renders with a button for the user to select so she can access another multistep flow with its own train to configure her business card / stationary order.

To implement this, you need to create two separate, unrelated trains: one for the main process, and another for the subprocess that visually indicates that it is part of a larger flow as shown in Figure 5.

Figure 5: Example of train within train rendering.



To configure a train as a subtrain, simply set its `Sub Train` property to **true**. Otherwise, you define and implement this train and the associated page navigation just as you would for any other standalone train flow. See the `oracle.apps.fnd.framework.webui.beans.nav.OATrainBean` Javadoc for information about identifying a subtrain programmatically.

Support for Touch Devices

As of Release 12.2.4, OA Framework provides gestures support for trains. For touch devices, you may **Swipe-Left** or **Swipe-Right** to navigate to the next or previous page. Refer to the Gestures Support topic for additional Usage Restrictions specific to navigating between the pages of a train.

Personalization Considerations

Known Issues

- See a summary of key Locator Element: Train issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
  - Locator Element (Train)
  - Locator Element: Page/Record Navigation
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.nav.OATrainBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OANavigationBarBean`
- ToolBox Tutorial Application
  - See the Multistep (Create) module for an example implementation of the "basic" train with Oracle Workflow.
  - See the Update lab.

Masonry Layout

Overview

Traditionally, all the contents of an OA Framework page (or contents inside a region of a OA Framework page) have been laid out the same way in all devices, regardless of the available screen size. This mean that horizontal or vertical scroll bars may be needed to display the complete content on different devices that have different screen shapes and sizes. Also, the horizontal layout of the header and footer takes up more space on some devices.

Responsive Web Design is an approach that suggests design and development should respond to the user behavior and environment, based on screen size, platform and orientation. As the user switches from a laptop to a tablet, the website should automatically switch to accommodate for resolution, image size and scripting abilities. In other words, the website should have the technology to automatically respond to the uses preferences and choice of device.

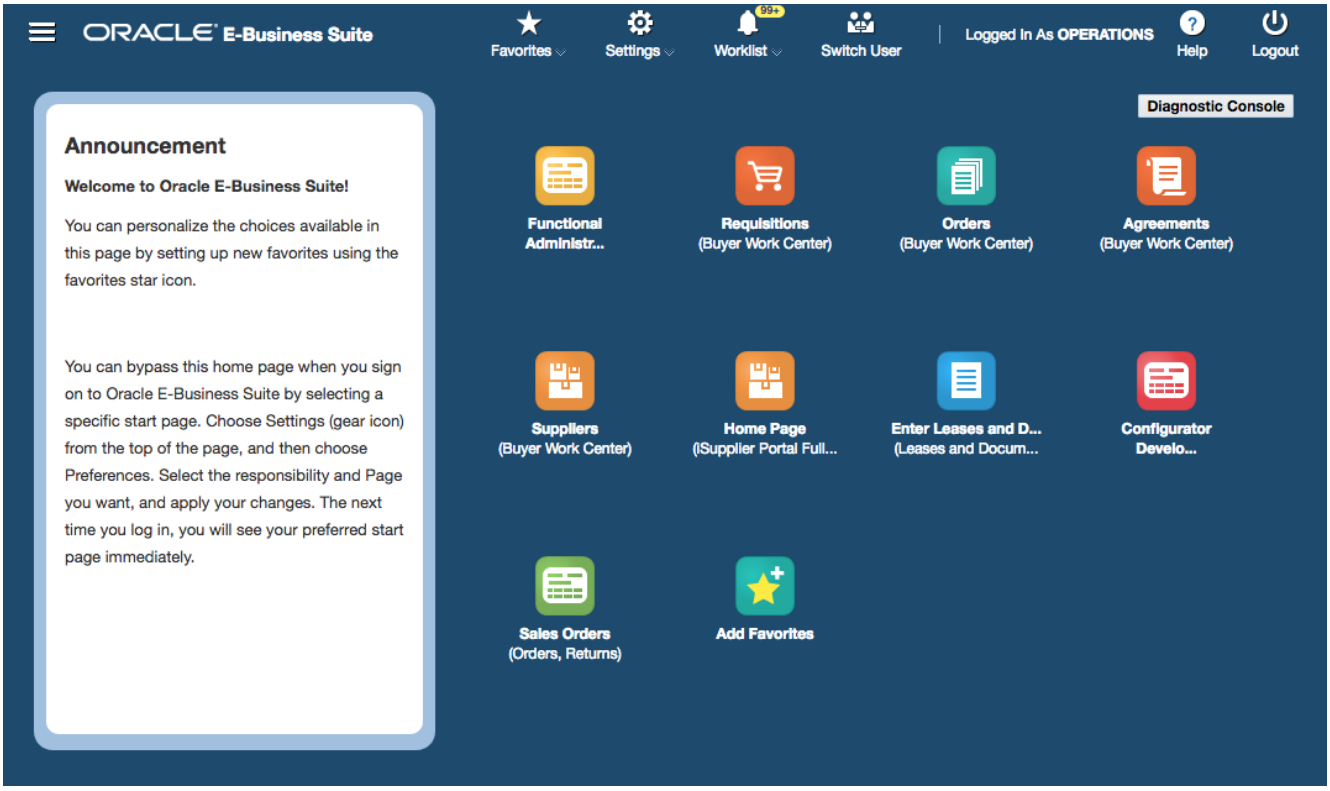
The OA Framework solution to this issue is delivered in Release 12.2.9, through the new Masonry layout mechanism. A Masonry layout will lay out its children in optimal positions, based on the available horizontal and vertical space, to produce the best page layout across different devices. A Masonry layout has masonry tiles inside it to hold the contents.

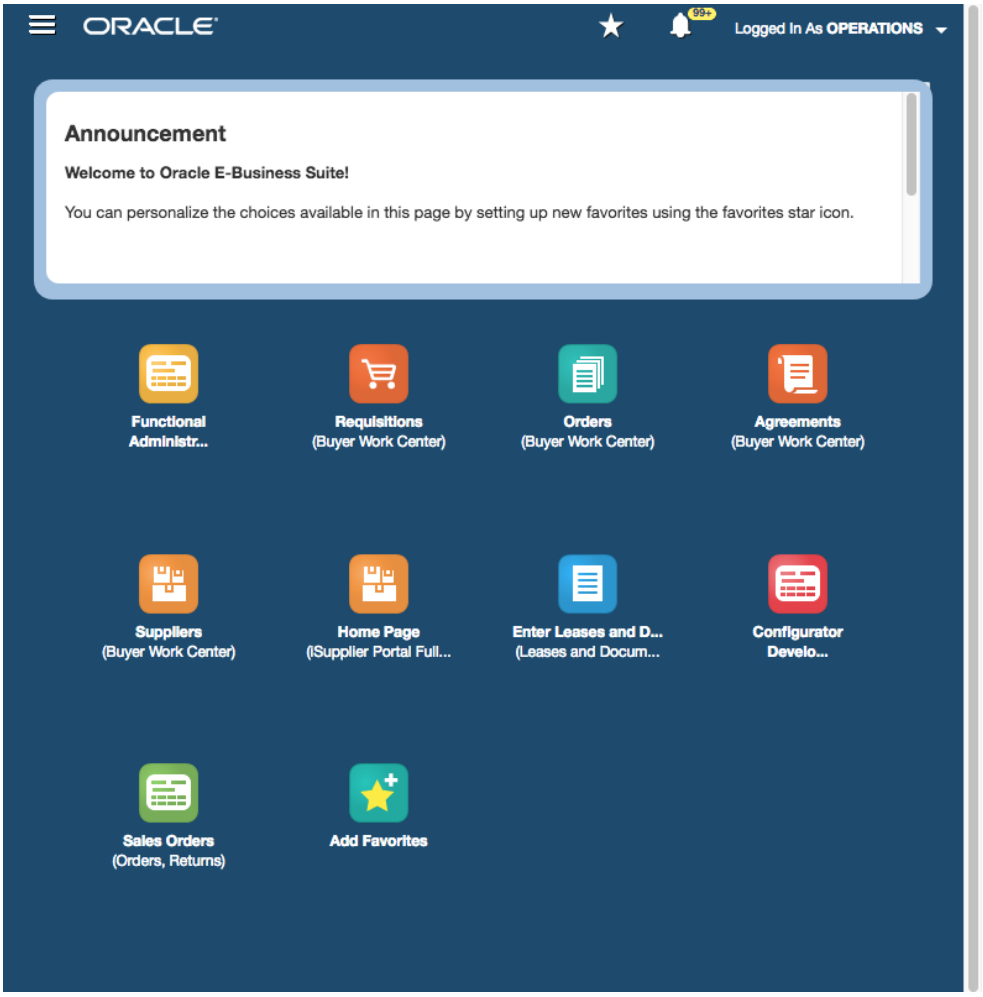
OA Framework developers can use the Masonry layout (and associated tiles) to arrange contents in different devices according to the tile size specifications. If a page has a Masonry layout, the header and footer of that page will automatically be aligned to fit the available space.

The two images below are shown side by side to illustrate the basic principle of a Masonry layout and its component tiles on two different screen shapes.

Desktop Layout	Tablet Layout
	

The following two screenshots show how this is put into practice in OA Framework, with the layout varying according to the available horizontal and vertical space in a browser window. The same principle applies to how the layout changes to make best use of the available display area on different physical devices that may have different screen orientations.





An important associated change is to the use of profile options. Prior to Release 12.2.9, the profile option APPLICATIONS_HOME_PAGE determines the look-and-feel of the Oracle Self-Service Applications Personal Home Page. With the change to use Masonry, the user preference set in the Settings > Preferences page takes precedence over the profile option value. The values set in this page for both home page and icon style are stored in the FND_USER_PREFERENCES table. If no value is set here, the value

from APPLICATIONS_HOME_PAGE profile option is used.

Developer/System Administrator Procedures

Key points:

- A Masonry layout can be added declaratively or programmatically to the page.
- A Masonry layout can be added directly inside a page layout, or inside other regions.
- A Masonry layout allows tiles to be added inside it, to hold the contents.
- If a Masonry layout is a direct child of Page Layout, the page cannot have other children.
    - This is not restricted at design time, in order to allow creation of other child and make the masonry layout move inside the newly created bean as a child.
- Masonry layouts cannot be nested.
- If the content is bigger than the size of the masonry tile, horizontal and/or vertical scroll bars are displayed in the tile to scroll through and see the content.

Visual Design and User Interface

A Masonry layout supports two modes:

- Masonry layout as a direct child of page layout - When directly inside a page layout, it lays out the children according to the device screen
- size. Masonry layout inside other regions - When inside other regions, it lays out the children within the boundaries of its parent container.

User Interface Interactions

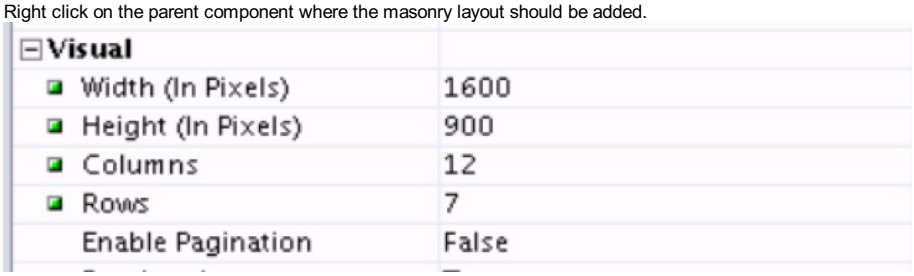
In the UI, the user will see a change in the contents of the masonry layout in any of the following circumstances:

- When the same page is accessed on different devices.
- When the screen orientation is changed.
- When the size of the parent containing the masonry layout is changed (for example inside a splitter, when the splitter is repositioned).

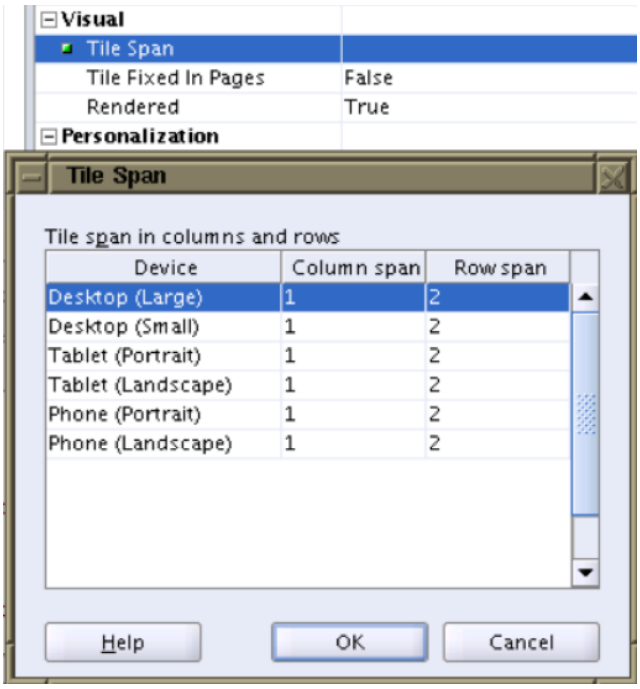
The re-layout of masonry tiles is handled automatically when any of the above events happen. However, when pagination is enabled, the user can navigate through the pages by clicking the pagination bar dots or through gestures (on devices that support gestures).

Declarative Implementation (If Applicable)

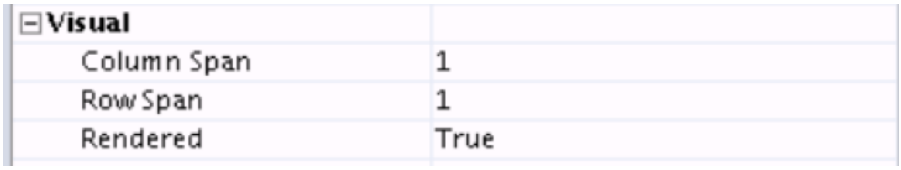
1. Right click on the parent component where the masonry layout should be added.



2. Specify the following properties for the masonry layout.
3. Right click and add a masonry tile inside the masonry layout.
4. Specify the following properties for the masonry tile if the masonry layout is directly under a page layout.



5. Right click and add a masonry tile inside a masonry layout. Specify the following properties for the masonry tile if the masonry layout is not directly under a page layout.



Programmatic Control (If Applicable)

Programmatic implementation of the new feature can be done as shown here:

```
OAMasonryLayoutBean ml = new OAMasonryLayoutBean();
ml.setID("masonrySB"); // This should be set
ml.setWidthInPixels(1200); // Defaults to 1600
ml.setHeightInPixels(750); // Defaults to 900
ml.setLayoutCols(12); // Defaults to 7
ml.setLayoutRows(7); // Defaults to 5
ml.setPaginationEnabled(true); // Defaults to false
```

Personalization Considerations

All properties of the masonry layout and masonry tiles are personalizable.

Keyboard Interactions

As a Masonry layout just lays out the tile contents, keyboard interactions with the tile contents are unchanged. For example, when the focus is on a masonry tile, pressing the arrow keys let the keyboard user navigate across the tiles.

Touch Interactions

When pagination is enabled, the pages can be traversed using gesture support. Other touch interactions with the contents of the tile are unchanged.

Properties of Masonry Layout

Property Code	Property Display Name	Description	Type	Default Value	Personalizable	Translatable
WidthInPixels	Width (In Pixels)	Width (in pixels) specifies the base screen size on which you are designing the layout.	Integer	1600	Yes	No
HeightInPixels	Height (In Pixels)	Height (in pixels) specifies the base screen size on which you are designing the layout.	Integer	900	Yes	No
LayoutCols	Tile Span	Number of columns the width specified above should be divided into.	Integer	7	Yes	No
LayoutRows	Tile Span	Number of rows the width specified above should be divided into.	Integer	5	Yes	No
paginationEnabled	Enable Pagination	Flag to determine if the contents of the layout should be wrapped into pages or not.	Boolean	False	Yes	No
ID	ID	ID of Masonry Layout.	String	N/A	Yes	No

Properties of Masonry Tile (When the Only Child of Page Layout)

Property Code	Property Display Name	Description	Type	Default Value	Personalizable	Translatable
BigDesktopColTileSpan	TileSpan	Number of columns the tile will span in different devices.	String (1 to 9 and also ™)	1	No	No
BigDesktopRowTileSpan		If the row or column span is specified as n, it means the span extends to entire row or column.				
DesktopColTileSpan						
DesktopRowTileSpan						
TabletLandscapeColTileSpan						
TabletLandscapeRowTileSpan						
TablePortraitColTileSpan						
TabletPortraitRowTileSpan						
PhonePortraitColTileSpan						
PhonePortraitRowTileSpan						
TileFixed	Tile Fixed In Pages	When this flag is set to true, the tile would appear on all pages of the layout. (Valid only when pagination is enabled on the masonry layout.)	Boolean	False	Yes	No
ID	ID	ID of masonry layout	String	N/A	Yes	No

Properties of Masonry Tile (When Inside Other Regions)

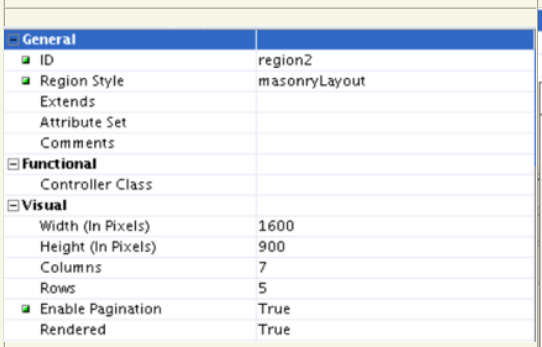
Property Code	Property Display Name	Description	Type	Default Value	Personalizable	Translatable
TileColSpan	Column Span	Number of columns the tile will span.	Integer	1	Yes	No

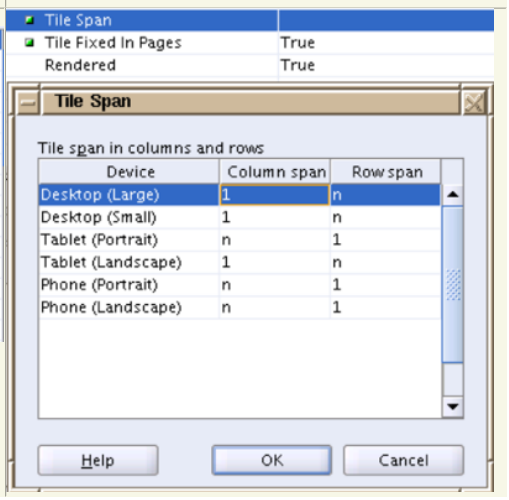
TileRowSpan	Row Span	Number of rows the tile will span.	Integer	1	Yes	No
ID	ID	ID of masonry layout.	String	N/A	Yes	No

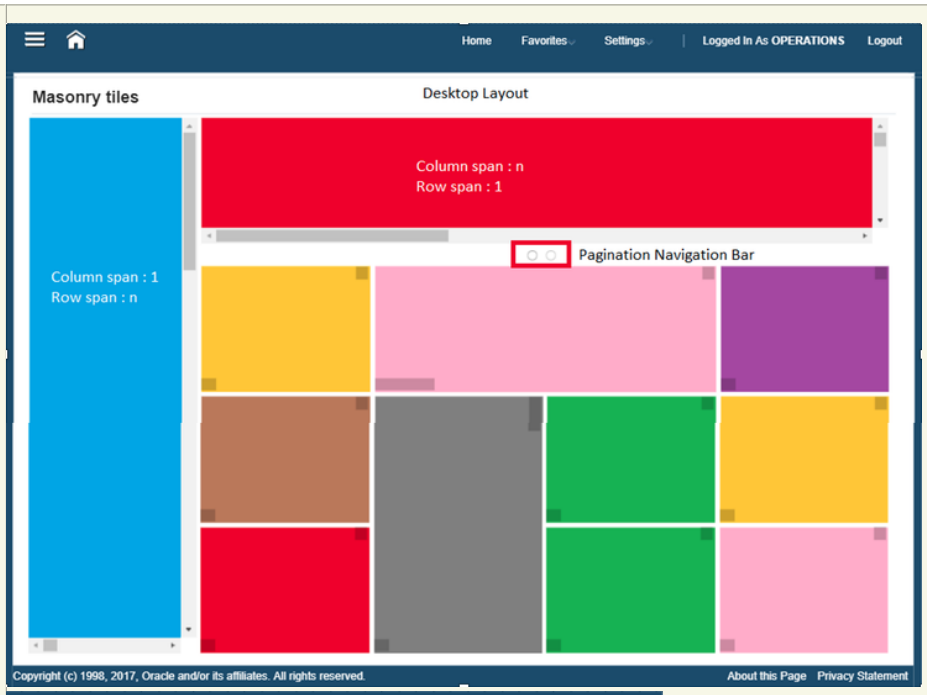
JDeveloper Design Time Example

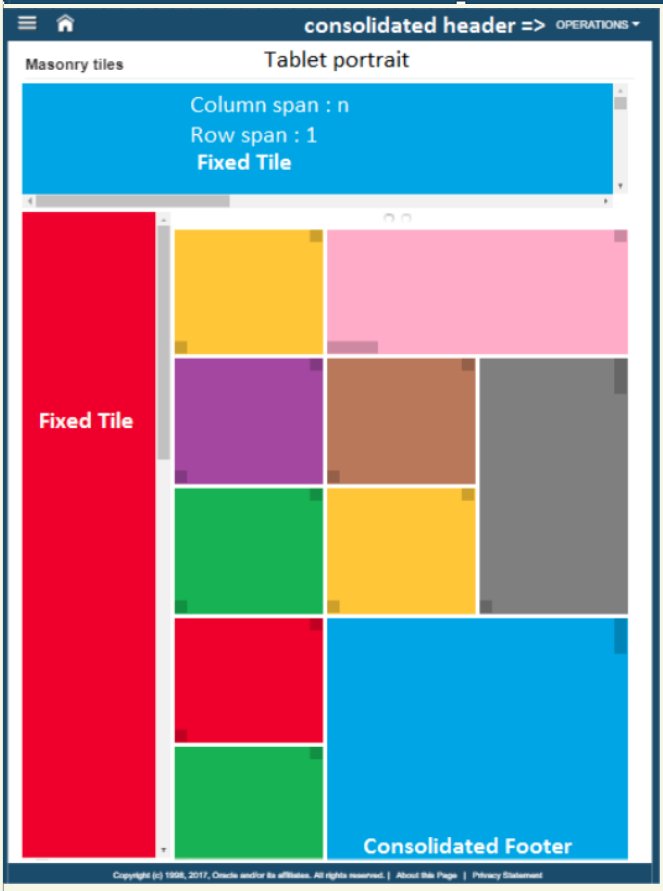
The following example images sum up the principles of Masonry layout in the context of JDeveloper.

Feature Number	Design Time	
1	**Properties of Masonry Layout**	**Tile Span of Masonry Tile (Blue)**









The two example screen layouts shown above represent a desktop and tablet respectively.

Menu Component

Overview

From Release 12.2.4, you can enable a menu component on a page item. The menu component provides a way for you to design an intuitive and space-saving user interface that allows end-users to choose one or more actions to perform on an item. Invoking the menu fires client-side-only events on the menu-enabled item. Select any area outside the menu to close the menu itself.

Currently, you can enable a menu component on a link, image or button item. The menu appears as a pop-up when you select the menu-enabled item and supports two modes of operation depending on its implementation:

- **Selection Disabled** - allows you to choose only one option from the menu list and fires the action associated with that chosen option.
- **Selection Enabled** - allows you to choose multiple menu options and sequentially fires the actions associated with those chosen options. The menu displays a **Check** to the left of an item that is selected. In addition, you can also enable reordering of the items on a menu, as represented by **Up** and **Down** arrows to the right of the menu item.

Contents

This document contains the following topics:

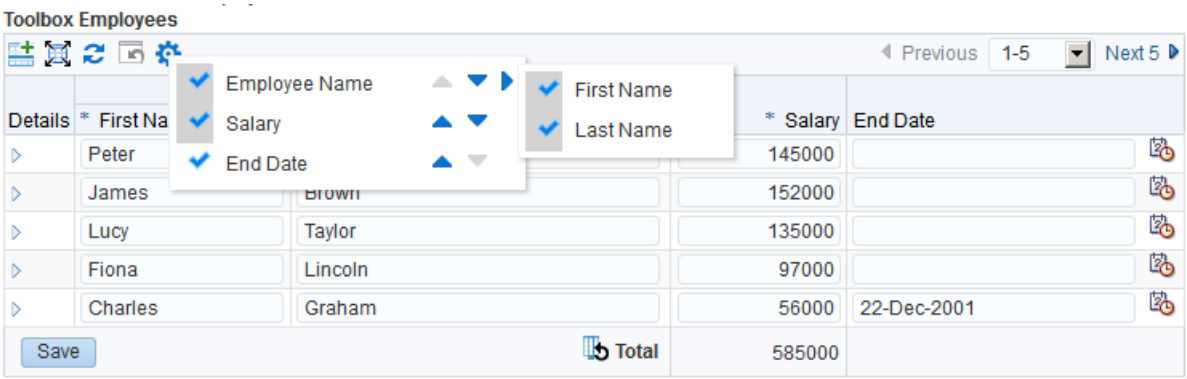
- Overview
  - User Interactions
    - Accessibility Considerations
- Declarative Implementation
- Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

User Interactions

When a user selects a menu-enabled item, a menu pop-up appears. In a Selection Disabled menu, a user may select only one item in the menu, which performs a single action when the menu closes.

Figure 1 shows an example of a menu on the **Table Settings** icon of an advanced table. The menu allows a user to hide/show one column (**End Date**) in the table by selecting or deselecting the menu item. A user may also choose the **Up** or **Down** arrow icons to reorder any of the menu items, which in turn reorder the columns in the table respectively when the user closes the menu. Note that while the menu itself is not Selection Disabled, the menu and submenu contain Selection Disabled menu items.

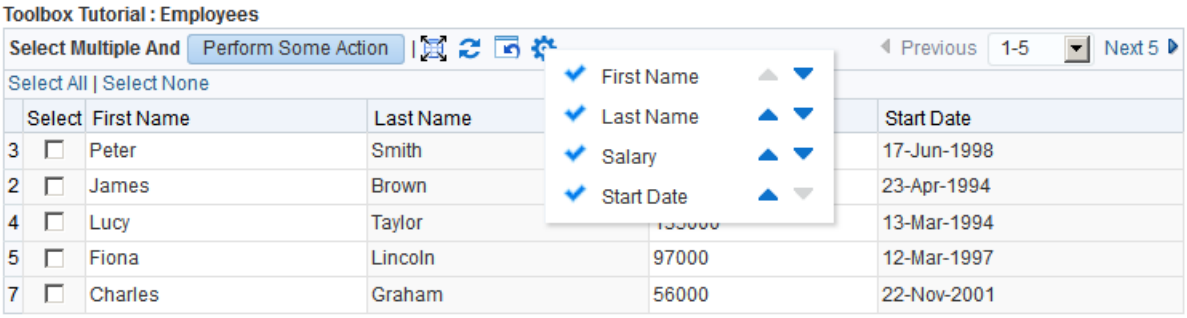
Figure 1: Menu with submenu on Table Settings icon (image) with selection disabled.



In a Selection Enabled menu, a user may select multiple items in the menu, which sequentially fire the selected actions when the menu closes.

Figure 2 shows an example of a Selection Enabled menu on the Table Settings icon of an advanced table. The menu allows a user to hide/show multiple columns in the table by selecting or deselecting multiple menu items. A user may also choose the **Up** or **Down** arrow icons to reorder any of the menu items, which in turn reorder the columns in the table respectively when the user closes the menu.

Figure 2. Menu on Table Settings icon (image) with selection enabled.



Accessibility Considerations

In standard or screen reader accessibility mode, a user may interact with a menu as follows:

Menu invocation

- Press **[Tab]** to navigate through a page, to a menu-enabled item and press the **[Enter]** key to toggle the display of the menu. The focus remains on the menu-enabled item.
- Alternatively, with focus on the menu-enabled item, you may press the **[Down Arrow]** key to display the menu and move the focus onto the first item of the menu.

Navigation within a menu

- Once the focus is on the menu itself, use the **[Up Arrow]** and **[Down Arrow]** keys to navigate between the menu items, "wrapping" at the top and bottom of the menu.
- With the focus on a menu item, press **[Space]** to toggle the state of the menu. For a Selection Enabled menu, this means checking or unchecking the focused menu item. For a Selection Disabled menu, this means firing the action associated with the menu item.
- With the focus on a menu item, press **[Right Arrow]** to invoke the submenu (if present) and move focus to the first submenu item. Press **[Left Arrow]** to close the focused submenu.

Reorder menu items

- With focus on a menu item, press **[Ctrl][Up]** to move menu item up the menu list, keeping focus on the moved menu item.
- With focus on a menu item, press **[Ctrl][Down]** to move menu item up the menu list, keeping focus on the moved menu item.

Navigation out of a menu

- With the focus on a menu item, press **[Esc]** to close the menu and return focus to the menu-enabled item.
- With focus on a menu item, press **[Tab]** to navigate to the next item on the page.
- With focus on a menu item, press **[Tab]** or **[Shift][Tab]** to close the menu and move focus back to the menu-enabled item.

Note: Accessibility behavior may differ based on the language session used. For example, in an Arabic session **[Left Arrow]** functions like the English **[Right Arrow]**.

Declarative Implementation

Specify the following metadata in OA Extension to enable the menu component on an item:

Step 1: In the OA Extension structure pane, for the page that should display the menu, select the region that contains the button, link or image item for which you want the menu to appear.

Step 2: Select the button, link or image item and display the context menu. In the context menu, under **New**, select **menu**.

Step 3: Set the following properties on this new menu region:

- `ID` - Specify an identifier that uniquely identifies the menu in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Enable Selection` - Specify **True** or **False** to indicate whether the menu should operate in selection enabled or selection disabled mode, respectively. The default value is **False**.
- `Enable Reorder` - Specify **True** or **False** to indicate whether the menu can be reordered by the user. The default is **False**.
- `Display as separator property` - Specify **True** or **False** to indicate whether the menu item should be displayed as a separator. The default is **False**.

Step 4: In the OA Extension Structure pane, select the menu region and display the context menu. In the context menu, under **New**, select **menuItem**.

Step 5: Set the following properties on the menu item:

- `ID` - Specify an identifier that uniquely identifies the menu item in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Text` - Enter the text label for the menu item.
- If `Enable Selection` is set to **True** for the parent menu, set these following properties for the menu item:
  - `Selected State` - Specify **enabled** or **disabled** to indicate the selection state of the menu item.
  - `Enable Selection` - Specify **True** or **False** to indicate whether to enable or disable selection only on this menu item, respectively. The default is **True**.
  - `OnSelect` - Specify the function to execute when the user selects this menu item.
  - `Icon URI` - Specify the source of any image you wish to display before the text.
  - `disabled` - Specify **True** to disable the entire menu item. The default is **False**.
  - `onReorderDown` - Specify the function to execute when a user selects the Reorder Down image.
  - `onReorderUp` - Specify the function to execute when a user selects the Reorder Up image.
- If `Enable Selection` is set to **False** for the parent menu, set these following properties for the menu item:
  - `Destination URI` - Specify the target URI of the menu item.
  - `Icon URI` - Specify the source of any image you wish to display before the text.

Step 6: (Optional) To define a submenu, in the OA Extension Structure pane, select the menu item and display the context menu. In the context menu, under **New**, select **menu**. Repeat Steps 4 and 5 above to create the submenu items.

Runtime Control

To programmatically enable a menu on an image, button or link, include the following example code in the `processRequest` method of your controller:

Step 1: Create a menu as an instance of `OAMenuBean` as shown:

```
OAMenuBean menu = new OAMenuBean();
menu.setID("MenuID");
// enable selection - hide/show type menu.
menu.setSelectionEnabled(true);
// enable reordering of menu items within menu
menu.setReorderEnabled(true);
// js function to execute upon submit
menu.setOnSubmit("jsFunction");
```

Step 2: Define the menu's options by creating menu item web beans and adding them as indexed children of the menu:

```
OAMenuItemBean menuItem1 = new OAMenuItemBean();
// selection state of menu item

menuItem1.setSelectedState("enabled");
// text to display for option
menuItem1.setText("Employee");
menuItem1.setID("menuItem1");
// row level action
menuItem1.setOnSelect("jsFunction");
// row level action for Reorder Up
menuItem1.setOnReorderUp("jsFunction");
// disable menu item
menuItem1.setEnabled(false);
// disable only selection of menu item
menuitem1.setSelectionEnabled(false);
// target URL - used in selection disabled menu.
menuItem1.setDestination(url);
// set source for image that appears before text of each option
menuItem1.setIconURI(imgSource);
// set submenu, if any, under menu item
menuItem1.setMenu(menu1);
```

For a Selection Disabled menu, the `setSelectionEnabled` and `setOnReorder...` properties are disabled by default.

Step 3: Enable the menu created in Step 1 on an image, button or link as shown:

```
// Enable the menu on an image
OAImageBean image = (OAImageBean)webBean.findChildRecursive("settingIcon");
image.setMenu(menu);
```

Personalization Considerations

- See a summary of Menu personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guidelines
- Developer's Guide
- Javadoc
    - `oracle.cabo.ui.beans.layout.MenuBean`
    - `oracle.cabo.ui.beans.layout.MenuItemBean`
    - `oracle.apps.fnd.framework.webui.beans.layout.OAMenuBean`
    - `oracle.apps.fnd.framework.webui.beans.layout.OAMenuItemBean`
- ToolBox Tutorial / Sample Library

Message Box

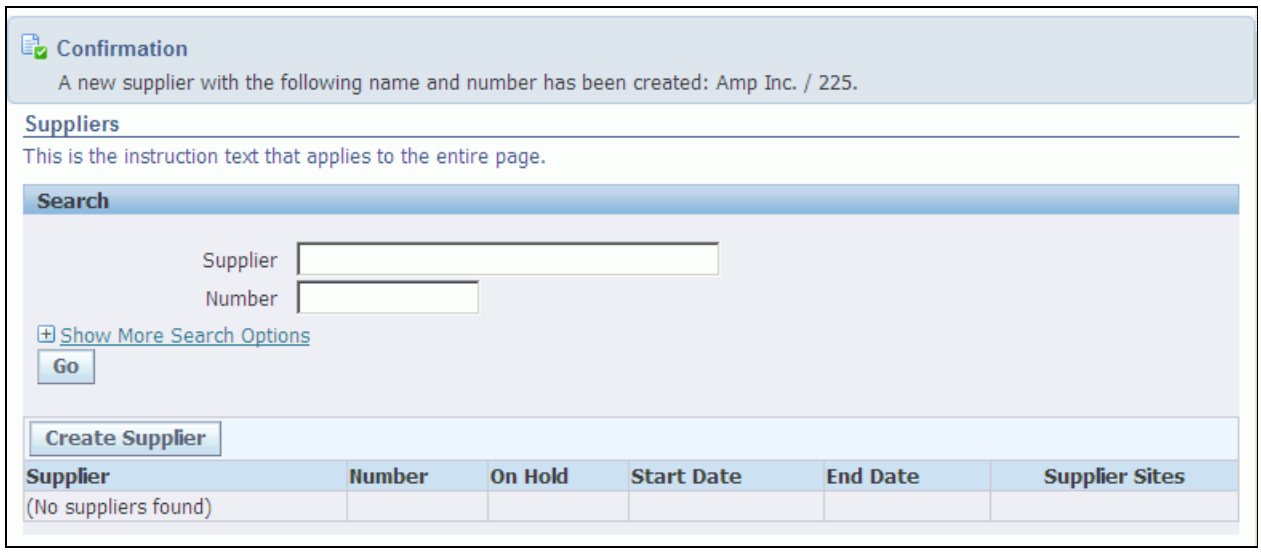
Overview

Per the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Message Box, you can display the following standard kinds of messages at the top of a page:

- Error
- Warning
- Confirmation
- Information

Each message renders with a dark beige background to make it more visible, and a standard icon. For example, Figure 1 shows a confirmation message box:

Figure 1: Example of a confirmation message box displayed at the top of a page



Note if you want to display these messages in a separate page, see the Dialog Page documentation.

Declarative Implementation

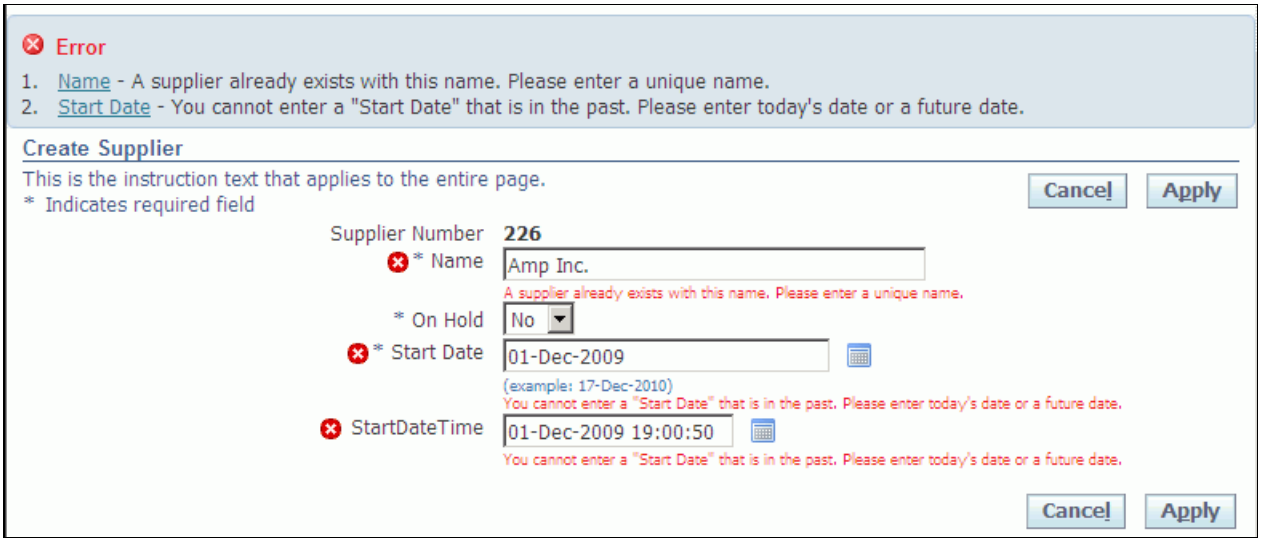
Since messages are displayed in the context of runtime events and circumstances, there is no corresponding declarative implementation.

If you want to display messages that include HTML tags, see the Custom HTML document.

Runtime Control

As described in the Error Handling document, if you throw an OAException (or any of its subclasses), the OA Framework automatically displays an error message at the top of the current page. If you throw row or attribute-level exceptions in your business logic, the error message box includes links to the rows and/or fields which are in error, and displays error icons for these components as shown in Figure 2 below.

Figure 2: Example of an error message box displaying attribute-level validation exceptions thrown in the underlying entity object.



You can also explicitly display a message box of any type using the following code in your controller (this particular example displays a confirmation message after a successful commit).

```
processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Get the purchase order number from the request.

  String orderNumber = pageContext.getParameter("headerId");

  MessageToken[] tokens = { new MessageToken("PO_NUMBER", orderNumber)};
  OAException message = new OAException("ICX", "FWK_TBX_T_PO_UPDATE_CONFIRM", tokens,
   OAException.CONFIRMATION, null);

  pageContext.putDialogMessage(message);

}
```

Note that you construct an `oracle.apps.fnd.framework.OAException` object and set the kind of message you want (other options are `OAException.WARNING`, `OAException.INFORMATION` and `OAException.ERROR`). Then you simply identify this exception for display when the page renders by calling the `OAPageContext.putDialogMessage()` method.

At runtime, the OA Framework constructs and `oracle.apps.fnd.framework.webui.beans.message.OAMessageBoxBean` and adds it to the web bean hierarchy.

Exceptions and Navigation

If -- after you call `putDialogMessage()` in your `processFormRequest()` method -- you want to forward to the current page or another page and display the message at the top of the new target page, you need to call the appropriate `oracle.apps.fnd.framework.webui.OAPageContext forwardImmediately*()` method. The OA Framework immediately stops processing the page and issues a forward before displaying the messages.

Note: The two pages should share the same root application module, and you should retain this AM while forwarding.

If you plan to call the `OAPageContext.setForwardURL()` or `setForwardURLToCurrentPage()` methods *before* throwing an exception, you need to decide whether you want to ignore the messages and proceed with the forward action, or stop and show the messages in the current page. Then, set the `messageLevel setForward*()` method parameter as described in the `OAPageContext` Javadoc.

Multiple Message Type Handling

When you register or throw multiple exceptions, the OA Framework combines them into a single message box using the following rules:

- Since an error is more important than a warning, the message box is titled "Error" if both errors and warnings exist.
- Confirmations and errors cannot be shown together. In this case, the OA Framework simply ignores the confirmation message(s).
- You can, however, show confirmations with warnings. The message box is titled "Confirmation," and it contains both types of messages.

Messages that Reference Web Beans Without Prompts

If your page contains a web bean that does not have a prompt value associated with it and a user enters an invalid value for the web bean, the error message that results will be malformed. For example, if you have a messageTextInput field with no prompt and you enter an invalid value, the error message may display as:

```
Value "A" in "" is not a number.
```

To avoid these malformed messages, use `oracle.apps.fnd.framework.webui.beans.message.OAMessagePromptBean`.

Create an `OAMessagePromptBean` and set:

- `Prompt` - to the alternate prompt that is going to be displayed for the web bean.
- `LabeledNodeId` - to those set of web bean ID's that you want associated with this alternate prompt. (These are the web beans without a current prompt associated with them).

You can associate multiple web bean ID's to the `LabeledNodeId` of this `OAMessagePromptBean`. As a result, all those web beans will be associated with the prompt of the `OAMessagePromptBean`.

The following code example illustrates this:

```
...
OAMessagePromptBean bean = new OAMessagePromptBean();
bean.setID("someID");
bean.setPrompt("Alternative");
bean.setLabeledNodeID("RequiredBeanID");
webBean.addIndexedChild(bean);
...
```

`RequiredBeanID` is the ID of the web bean with which this alternate Prompt is to be associated.

Related Information

- BLAF UI Guidelines
  - Message Box
- Javadoc
  - `oracle.apps.fnd.framework.OAException`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageBoxBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessagePromptBean`
  - `oracle.cabo.ui.beans.message.MessagePromptBean`

Rich Text Editor

Important Announcement

The component richTextEditor in OA Extension is deprecated. Please use messageRichTextEditor instead. The richTextEditor component is non-compliant with BLAF guidelines. The new component, messageRichTextEditor, is compliant with the BLAF and Multi-Lingual guidelines and being a Message component, supports properties like Prompt and Tip. It also provides a superior data validation mechanism.

Overview

The Rich Text Editor component provides rich text editing capability within OA Framework. The component allows users to author, edit and view rich text content in a browser that supports IFRAMEs.

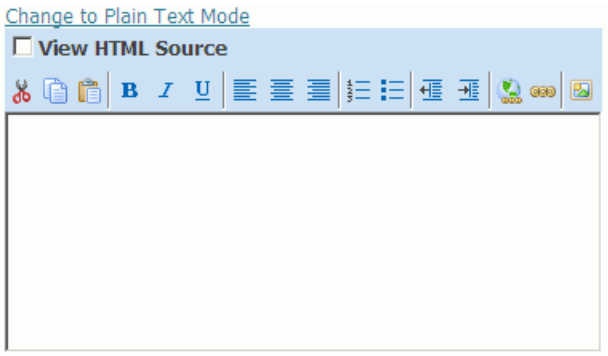
Note The Rich Text Editor area is currently supported for Microsoft Internet Explorer (IE) 5.5 and above on the Microsoft windows platform and for Mozilla, on all platforms where Mozilla 1.6 to 3.5 are available. IE and earlier versions of Mozilla both have support for the IFRAME html tag and expose browser DOM API's through JavaScript, which are used for rendering and manipulating the rich text. On all other platforms and browsers, such as Safari and Mozilla Firefox 3.5 and above, the Cut, Copy and Paste icons do not render due to security restrictions imposed by Safari and Firefox 3.5 and above.

The Rich Text Editor consists of two parts:

1. Tool Bar
2. Editable Area

The tool bar is rendered above the editable area. The user can enter text in the editable area and format it by selecting the text and choosing one of the icons on the Tool bar. The user can also toggle between Rich Text and Normal Text modes or view the HTML source for the text entered in the editor.

Figure 1: Example of a Rich Text Area Displaying a Subset of Toolbar icons



The toolbar contains two rows of editing tools.

The **First Row** of the toolbar may contain the following:

- Font poplist - lets you choose the font for the selected text.
- Font Color poplist - lets you choose the font color for the selected text.
- Font Size poplist - lets you choose the HTML font size for the selected text.
- Checkbox to view HTML source.

The **Second Row** of the toolbar may contain the following:

- Cut - cut the selected text.
- Copy - copy the selected text.
- Paste - paste the previously copied text.
- Bold - make the selected text **Bold**.
- Italic - *Italicize* the selected text.
- Underline - Underline the selected text.
- Align Left - align the text to the left.
- Align Center - align the text to the center.
- Align Right - align the text to the right.
- Number Order List - change the selected text to be an number ordered list.

- Bulleted List - change the selected text to be a bulleted list.
- Decrease Indent - decrease the indentation of the selected text.
- Increase Indent - increase the indentation of the selected text.
- Create Hyperlink - create a hyperlink for the selected text.
- Click-through Destination - build an application-specific URI. It is the responsibility of the calling application to specify a popup page URL to build the URI. The popup page builds the URI and calls `setHref(url)` via the RichTextEditor javascript proxy object to set the URI on the selected text or image in the editable area.
- Upload Image - allows user to upload and embed an image from another source, such as from the Desktop, Content Repository, etc. It is the responsibility of the calling application to specify a popup page URL for the image upload. In the popup page, the user selects the image from the Content Repository or upload it from the desktop. The popup page then calls `insertImageTag(imageSrc)` via the RichTextEditor javascript proxy object to insert the image tag in the editable area.

Usage Notes

The Rich Text Editor (`oracle.apps.fnd.framework.webui.beans.message.OAMessageRichTextEditorBean`) can be used to provide rich text editing capabilities where needed, instead of using `OAMessageTextInputBean`.

Attention: Rich Text Editor cannot be used in search regions or tables.

Attention: Rich Text Editor supports only those tags that are generated by its Rich Text Editor toolbar buttons. Since support for tags generated by copying content from other HTML or text editors may vary based on a browser's capabilities, Rich Text Editor does not support those other tags.

Contents

- Accessibility Compliance
- Bi-direction Support
- AntiSamy Support
- Declarative Implementation
- Runtime Control
  - Instantiate
  - Control Visual Properties
  - Setting the Font Bar
  - Setting the Button Bar
  - Setting Click-Thru URI, Image Upload and Create Hyperlink
  - Public Javascript Functions for the Rich Text Editor
- Personalization Considerations
- Known Issues
- Related Information

Accessibility Compliance

The Rich Text Editor conforms to the Oracle Global HTML Accessibility Guidelines. Accessibility of the Rich Text Editor is determined by the value of the profile option **ICX_ACCESSIBILITY_FEATURES**. This profile value is read by OARenderingContext and made available to the Rich Text Editor. There are three possible Accessibility modes:

- **DEFAULT_MODE** - where strict accessibility compliance is met and the Rich Text Editor is always rendered in Text Mode. This mode results when ICX_ACCESSIBILITY_FEATURES is set to Yes.
- **SCREEN_READER_MODE** - where content is optimized for screen readers and the Rich Text Editor is always rendered in Text Mode. This mode results when ICX_ACCESSIBILITY_FEATURES is set to Screen Reader.
- **INACCESSIBLE_MODE** - where code is optimized to strip out Accessibility-specific constructs.
  In this mode, the Rich Text Editor is rendered in Rich Text Mode, along with a Switch Mode hyperlink, that is based on the `SwitchHyperLink` property, set internally by OA. This mode results when ICX_ACCESSIBILITY_FEATURES is set to No.

Bi-direction Support

Rich Text Editor supports bi-direction rendering and renders icons in the button bar accordingly. The following icons are bi-di supported:

Icon	Regular Icon	Bi-Di Icon
Align Left		
Align Right		
Bullet List		

Number List		
Indent List		
Outdent List		

AntiSamy Support

As of Release 12.2.2, any content you create in the Rich Text Editor now undergoes an Antisamy check. AntiSamy is an API provided by the Open Web Application Security Project (OWASP) to remove malicious pieces of code from HTML files.

OA Framework automatically applies this AntiSamy API when you submit content via the Rich Text Editor. An information dialog displays the following message if an unsupported HTML tag, script, or CSS is found in the submitted content: "Restricted content in the Rich Text Editor has been filtered".

This filtering ensures that only sanitized HTML is submitted. AntiSamy filtering is optional and can be turned off by enabling the profile option **FND_DISABLE_ANTISAMY_FILTER** at the application level. For more information about the AntiSamy check, refer to *Security Configuration Mechanism in the Attachments Feature in Oracle E-Business Suite,* My Oracle Support (formerly Oracle *MetaLink*) Knowledge Document 1357849.1. The profile value **FND_DISABLE_ANTISAMY_FILTER** can be set at the Site, Application, and User levels. The value at the User level takes over the Application level, which in turn takes precedence over the Site level.

Declarative Implementation

To add a Rich Text Editor to a region:

Step 1: Select the region in the OA Extension Structure pane, select **New** > **Item** from the context menu. Name your new item in accordance with the OA Framework naming standards, and set its `Item Style` to **messageRichTextEditor**.

Step 2: Set the following properties for the messageRichTextEditor item.

Property	Meaning
`Rich Text Mode Height`	The display height of the rich text editor in pixels or as a percent (specify as *number*%) of the container.
`Rich Text Mode Length`	The display length of the rich text editor in pixels or as a percent (specify as *number*%) of the container.
`Plain Text Mode Height`	The display height of the text mode editor in bytes.
`Plain Text Mode Length`	The display length of the text mode editor in bytes.
`Font Bar`	Indicates if the font bar is displayed.
`Edit Icons`	Indicates if the cut, copy and paste icons are displayed.
`Style Icons`	Indicates if the bold, italic and underline icons are displayed.
`Alignment Icons`	Indicates if the left, center, and right alignment icons are displayed.
`Bullet Icons`	Indicates if the bullet and numbered list icons are displayed.
`Indentation Icons`	Indicates if the indentation icons are displayed.
`Create Hyperlink Icon`	Indicates if the Create Hyperlink icon is displayed.
`Image Upload URl`	The URI launched from the Upload Image icon. If null, no icon is shown. See the description of the Upload Image Button Bar icon for additional information on how to use this feature.
`ClickThru Destination URI`	The URI launched from the "Click Thru Destination" icon. If null, no icon is shown. See the description of the Click-through Destination Button Bar icon for additional information on how to use this feature.
`Maximum Length`	The maximum length of the data in the view attribute, which is also the maximum number of bytes that can be entered in the editor. A value has to be specified for this property, otherwise no input is taken by the Rich Text Editor. The exception is when the Data Type is CLOB, in which case this property does not need to be set.
`View Instance`	The name of the view object instance that is mapped to the text in the Rich Text Editor.
`View Attribute`	The name of the view attribute that maps to a column in the view object.

Note The `Maximum Length` property sets the maximum length for the datatype VARCHAR2, and the maximum number of bytes that can be entered in the editor. In plain text mode, you are not allowed to type any more bytes beyond this value. In rich text mode, if you enter more

than the maximum number of bytes allowed, you get a JavaScript error when you submit the page.

In addition to the properties listed above, the messageRichTextEditor item supports the following properties:

Property	Meaning
`Required`	Indicates if the user must enter a value.
`Read Only`	Indicates if the value can be changed.
`Prompt`	The text label for the component.
Tip-related properties	To display the short/long tip.

Step 3: Save your work.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. Programmatically created web beans cannot be personalized, reused, or extended easily.

Instantiate

```
OAMessageRichTextEditorBean rteBean = (OAMessageRichTextEditorBean) createWebBean
  (pageContext, RICH_TEXT_EDITOR_BEAN);


//Set bean Id
rteBean.setID("htmlData");

//Set View Usage Name
rteBean.setViewAttributeName("Description");

//Set View Attribute Name
rteBean.setViewUsageName("RichTextEditorVO");

//Set Prompt
rteBean.setPrompt("Prompt");

//Add Bean to the web bean hierarchy
webBean.addIndexedChild(rteBean);
```

Control Visual Properties

All the visual properties can be controlled programmatically.

```
//set width in Text Mode
rteBean.setTextModeDisplayLength(80);

//set height in Text Mode
rteBean.setTextModeDisplayHeight(18);

//set Height in RichText Mode
rteBean.setRichTextModeDisplayHeight("100%");

//set width in RichText Mode
rteBean.setRichTextModeDisplayLength("100%");

//Set Maximum Characters can be entered in both Text and RichText mode
rteBean.setMaximumLength(4000);

//Set flag NOT to render Font Bar
rteBean.setFontBar(false);
```

Setting the Font Bar

You can set the Font Bar declaratively or programmatically:

```
//Set flag to render Font Bar
rteBean.setFontBar(false);
```

If you render the Font Bar, you must also create the poplists (ChoiceBeans) for the Fonts, Font Colors and Font Sizes and set them correctly to the Font Bar. Note that you can only set the Font Size and Font Color poplists programmatically. Also, the Rich Text Editor only supports the font sizes 1 through 7, which correspond to the seven absolute font sizes supported by the HTML <FONT>

tag.

```
//Font Poplist

OAMessageChoiceBean font = (OAMessageChoiceBean) createWebBean
  (pageContext, MESSAGE_CHOICE_BEAN);


// FontListVO should provide the list of supported fonts
font.setPickListViewUsageName("FontListVO");
font.setListValueAttribute("LookupCode");
font.setListDisplayAttribute("Meaning" );
font.setID("font");

//Font Color Poplist
OAMessageChoiceBean color = (OAMessageChoiceBean) createWebBean
  (pageContext, MESSAGE_CHOICE_BEAN);

// FontColorVO should provide the list of supported font colors
color.setPickListViewUsageName("FontColorVO");
color.setListValueAttribute("LookupCode");
color.setListDisplayAttribute("Meaning" );
color.setID("color");

//Font Size Poplist
OAMessageChoiceBean size = (OAMessageChoiceBean) createWebBean
  (pageContext, MESSAGE_CHOICE_BEAN);

// FontSizeVO should provide the list of supported font sizes
size.setPickListViewUsageName("FontSizeVO");
size.setListValueAttribute("LookupCode");
size.setListDisplayAttribute("Meaning" );
size.setID("size");

//Set Font dropdown Bean
rteBean.setFontBean(font);

//Set Color dropdown Bean
rteBean.setFontColorBean(color);

//Set Size dropdown Bean
rteBean.setFontSizeBean(size);
```

Setting the Button Bar

The Button bar is categorize into different groups, such as the Edit Group, Style Group, Alignment Group, etc. Each group can be controlled programmatically.

```
//Set to render Cut, Copy & Paste icons.
rteBean.setEditIconsGroup(true);

//Set to disable Cut, Copy & Paste icons.
rteBean.setEditIconsGroup(false);

//Set to render Bold, Italic & Underline icons.
rteBean.setStyleIconsGroup(true);

//Set to disable Bold, Italic & Underline icons.
rteBean.setStyleIconsGroup(false);

//Set to render align left, center & right icons.
rteBean.setAlignmentIconsGroup(true);

//Set to disable align left, center & right icons.
rteBean.setAlignmentIconsGroup(false);

//Set to render unordered and ordered Bullet list icons.
rteBean.setBulletsIconsGroup(true);


//Set to disable unordered and ordered Bullet list icons.
rteBean.setBulletsIconsGroup(false);
```

```
//Set to render indent & outdent icons.
rteBean.setIndentationIconsGroup(true);

//Set to disable indent & outdent icons.
rteBean.setIndentationIconsGroup(false);

//Set to render Hyperlink icon.
rteBean.setHyperlinkIcon(true);

//Set to disable Hyperlink icon.
rteBean.setHyperlinkIcon(false);

//Set to render Click Thru Destination icon.
rteBean.setClickThruDestinationUri(true);

//Set to disable Click Thru Destination icon.
rteBean.setClickThruDestinationUri(false);

//Set to render Image upload icon.
rteBean.setImageUploadUri(true);

//Set to disable Image upload icon.
rteBean.setImageUploadUri(false);
```

Setting Click-Thru URI, Image Upload and Create Hyperlink

In order to use the Click-Thru Destination or Upload Image feature of the Rich Text Editor, you must first define a URL for a Javascript popup page that you create:

- If you want to use the Click-Thru Destination feature, your popup page should build a URI and call `setHref(url)` via the RichTextEditor javascript proxy object to set the URI on the selected text or image in the editable area.
- If you want to use the Upload Image feature, your popup page should allow the user to select the image to upload from the desktop.

Use the following API's to set that URL as the destination of the Click-Thru Destination URI or Image Upload URI.

```
String proxy = rteBean.getID() + "_rte";

// Create URL for Click-Thru Destination page and set as Click-Thru
// Destination URI property.

rteBean.setClickThruDestinationUri(ctdUrl);

// Create URL for Image Upload page and set as the Image Upload URI property

String imageUrl =
  "javascript:popUpWindow('OA.jsp?page=/oracle/apps/..../ImageUploadPG',{});";
rteBean.setImageUploadUri(imageUrl);
```

Public Javascript Functions for the Rich Text Editor

`OAMessageRichTextEditorBean` provides JavaScript APIs for editing the text in the rich text editor through the JavaScript class `RichTextEditorProxy`. You can get a handle to the `RichTextEditorBean` proxy object and call appropriate Javascript APIs.

The naming convention for the proxy object is:

```
RichTextEditorBean_ ID + "_rte"
```

For example:

```
String proxy = rteBean.getID() + "_rte";
```

The `RichTextEditorProxy` provides the following Javascript APIs:

- `insertText(value)` - inserts any text at the current cursor position.
- `setValue(data)` - replaces the existing text in editable area.
- `createHyperlink(prompt)` - call when the user clicks on Create Hyperlink Button.
  For example, when a user selects text and clicks on Create Hyperlink Icon, a javascript input box to enter a URL pops up. The URL that is entered is set as the href for the hightlighted text.
- `execHTMLCommand(HTMLCommand)` - performs an HTML command on hightlighted text. This is called when the user selects the Cut, Copy, or Paste, etc. button on the button bar.
- `setFontBarDropdown(dropDownName, dropDownType)` - call this API when a change is made in the Font, Color or Size poplists. This in turn calls the `execHTMLCommand(HTMLCommand)` API to perform HTML commands that change the

font, font color or font size.

- `insertHTMLTag(tagString)` - call this API to insert HTML tags like `<IMG>`, `<BR>`, `<HR>` into the Rich Text Editor. For example, when a user selects the Image Upload icon and uploads/selects an image, use this API to paste the image tag with src attribute into the Rich Text Editor.
- `insertImageTag(imageSrc)` - this Javascript API is a specialized version of `insertHTMLTag(tagString)`. It takes imageSrc as a parameter and inserts the `'<IMG SRC="' + imageSrc + '">'` tag into the current cursor position of the editable area.
- `setHref(url)` - call this API when the user selects some text in the editable area and chooses the Click-Thru-Destination icon. This API sets the calculated URI as the href.

Disabling the Rich Text Editor

To disable the Rich Text Editor, set `READ_ONLY_ATTR` to `Boolean.TRUE`.

Personalization Considerations

- See a summary of Rich Text Editor personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
- Javadoc File(s)
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageRichTextEditorBean`
- Lesson(s)
- Sample Code

Mobile Applications

OA Framework-based applications are certified with the Safari browser on Apple iPad and the Chrome browser on Android tablets. This topic discusses gestures support and the special interactions certain components have with respect to the tablet browsers. OA Framework-based applications are neither certified nor supported on any other mobile device or PDA.

Contents

- Gesture Support
- Caveats On Touch Devices
- Look-and-Feel Optimizations for Touch Devices
- Known Issues

Gesture Support

Overview

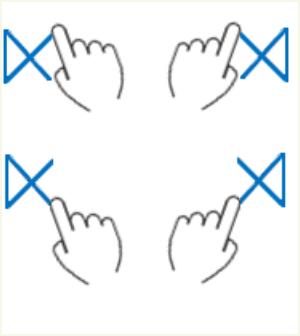
Beginning in Release 12.2.4, OA Framework supports the use of gestures to perform actions on a page to enhance user experiences on touch devices. To take advantage of this support, the iOS device must run Mobile Safari and the Android device must run Google Chrome.

Refer to "Recommended Browsers for Oracle E-Business Suite Release 12", My Oracle Support (formerly Oracle*MetaLink*) Knowledge Document 389422.1 for additional information.

The table below identifies the gestures supported by OA Framework.

Table 1. Visual representation and description of gestures supported by OA Framework.

Gesture	Visual Representation	Description
Single Tap		Briefly touch screen with fingertip.
Double Tap		Readily touch screen twice with fingertip.
Hold / Press / Press and Hold		Touch screen for an extended period of time with fingertip.
Pan / Drag		Move fingertip over screen surface without losing contact. Graphic depicts an example of a drag to the right.
Swipe / Flick		Quickly brush fingertip over screen surface. Graphic depicts an example of a swipe to the right.
Spread		Touch screen surface with two fingers and move them apart without losing contact with the screen.
Pinch		Touch screen surface with two fingers and bring them together without losing contact with the screen.

Check Mark / Tick		For right handed user, touch screen to make a check mark, scaling up while maintaining direction sense with right hand. For left handed user, touch screen to make a check mark, scaling up while maintaining direction sense with left hand.
X- Mark / Cross		Starting with the upper point of the X, touch screen to trace the X pattern, scaling up while maintaining direction sense with right hand or left hand, respectively. Starting with a lower point of the X, touch screen to trace the X pattern, scaling up while maintaining direction sense with right hand or left hand, respectively.

OA Framework supports these gestures out-of-the-box for the following use cases:

Table 2. OA Framework supported gestures for specific use cases.

OA Framework Component	Use Case	Description
Table / HGrid	Column Reorder	Perform a **Pan** on a column header to reorder a column on a touch device as compared to clicking the left mouse button and dragging the mouse to reorder a column on a desktop. **Note:** Since Column reorder is a rich table feature, you must first enable Rich table/HGrid interactions to take advantage of this feature.
Table / HGrid	Column Resize	Perform a **Pan** on a column header edge to resize column on a touch device as compared to clicking the left mouse button and dragging the mouse to resize a column on a desktop. **Note:** Since Column resize is a rich table feature, you must first enable Rich table/HGrid interactions to take advantage of this feature.
Table / HGrid	Column Hide/Show	Perform a **Hold** on a column header to reveal the hide/show icon on a touch device as compared to a mouse hover on a desktop. **Note:** Since Column hide/show is a rich table feature, you must first enable Rich table/HGrid interactions to take advantage of this feature.
Table	Pull to Refresh	Perform a **Pan** on the first set of records in the **top-to-down** direction to refresh a table. The **Pan** moves the table down and displays a **"Pull down to refresh"** prompt along with an icon. As you move the table down and it crosses a certain threshold, the prompt changes to **"Release to refresh"**. Once you release and end the **Pan**, OA Framework displays a **Loading** icon as it refreshes the table. **Note:** Since refreshing a table is a rich table feature and **Pan**, for navigation, is a touch device-related feature, you must enable both Rich table/HGrid interactions and gesture support take advantage of "pull to refresh".

Table	Navigation	Perform a **Swipe-Up** or **Swipe-Down** to navigate to the next or previous set of rows in a table, respectively, as compared to clicking the **Next** or **Previous** button with a mouse on a desktop. Also If you are displaying the first set of records in a table, performing a **Pan** in the **top-to-down** direction refreshes a table while performing a **Pan** in the **down-to-top** direction displays the next set of records. Similarly, if you are not displaying the first set of records, then performing a **Pan** in the **top-to-down** direction displays the previous set of records. **Note:** Since **Swipe** and **Pan,** for navigation, are touch device-related features, you must first enable gesture support take advantage of this feature.
Train	Navigation	Perform a **Swipe-Left** or **Swipe-Right** to navigate to the next or previous step of a train, respectively, as compared to clicking the **Next** or **Previous** button with a mouse on a desktop. **Note:** Since **Swipe,** for navigation, is a touch device-related feature, you must first enable gesture support take advantage of this feature.

In addition to the above use cases, you may also associate additional gestures at the page level. For example, you may attach a gesture such as a Tick/Check Mark to a Submit action. When a user performs a gesture on the touch device, the name of the gesture appears temporarily in a small pop-up at the bottom right corner of the device screen. Gestures such as tick and cross appear in this pop-up only if they are associated to components on the page.

Contents

- Enabling Gesture Support
- Declarative Implementation
- Runtime Control
- Usage Restrictions
- Personalization Considerations
- Known Issues
- Related Information

Enabling Gesture Support

To enable gesture support on touch devices, you must set these profile options as follows:

- **FND: Enable Touch Gestures** set to **True** at the Site, Application and/or User level. This profile is set to **True** by default.
- **FND: Enable Rich Table Interactions** set to **True** at the Site, Application and/or User level to allow users to perform rich interactions using gestures on Touch Devices.This profile is set to **True** by default.
- **Oracle Applications Look and Feel / APPS_LOOK_AND_FEEL** set to **Alta Look and Feel** to display the rich table features. Note that this profile option by default is not set (null), which implies the Alta Look and Feel.

You may also set the profile option **FND: Enable Touch Gesture Diagnostics** to **True** to enable a gestures diagnostic pop-up to help you debug gesture issues that arise.

Declarative Implementation

To associate a gesture at the page level, you must identify the following:

- A web bean to attach a gesture. This web bean recognizes the gesture if the gesture is performed within the web bean's physical scope in an HTML page.
- A target web bean which contains, as one of its attributes, the routine to trigger upon recognition of the gesture.
- An attribute of the target web bean, which holds the definition of the routine to trigger upon recognition of the gesture.

Note: Currently, the pageLayout region is the only web bean to which you can declaratively associate a gesture.

In the Oracle JDeveloper OA Extension Property Inspector:

Step 1: In the OA Extension structure pane, for the page that you want to associate a gesture, select the **pageLayout** region.

Step 2: Right-click and select **New** > **gestureMap** from the context menu. JDeveloper creates a new gestureMap named child under

the pageLayout region and creates a new gesture under gestureMap.

Step 3: Select the new gesture and set these properties as follows:

- `ID` - Specify an identifier that uniquely identifies the gesture in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Gesture Name` - select a supported gesture from the pull-down list to attach to the pageLayout region. For example, TAP.
- `Target Bean ID` - specify the ID of the target web bean that contains, as one of its attributes, the routine to be triggered upon gesture detection. Note the target web bean must exist in the current page. For example, submit1, which is the ID of the submitButton target web bean.
- `Target Bean Attribute` - select the attribute of the target bean that holds the routine definition. For example, onclick, which contains the definition of the routine to trigger upon recognition of the Tap gesture.

Step 4: You should define a separate gesture under gestureMap for each gesture you wish to associate to the web bean. To define another gesture, select **gestureMap** in the OA Extension structure pane, right-click and select **New** > **gesture**. Repeat Step 3 to set the properties for the gesture.

Runtime Control

To programmatically associate a gesture to a Page Layout region, you can use the following set of supported gesture APIs:

Gesture	API
Hold	`addGesture(GestureMap.Gestures.HOLD, "targetBeanId", "targetBeanAttribute")`
Swipe	`addGesture(GestureMap.Gestures.SWIPE, "targetBeanId", "targetBeanAttribute")`
Swipe-Up	`addGesture(GestureMap.Gestures.SWIPE_UP, "targetBeanId", "targetBeanAttribute")`
Swipe-Down	`addGesture(GestureMap.Gestures.SWIPE_DOWN, "targetBeanId", "targetBeanAttribute")`
Swipe-Left	`addGesture(GestureMap.Gestures.SWIPE_LEFT, "targetBeanId", "targetBeanAttribute")`
Swipe-Right	`addGesture(GestureMap.Gestures.SWIPE_RIGHT, "targetBeanId", "targetBeanAttribute")`
Pan-Start	`addGesture(GestureMap.Gestures.DRAG_START, "targetBeanId", "targetBeanAttribute")`
Pan	`addGesture(GestureMap.Gestures.DRAG, "targetBeanId", "targetBeanAttribute")`
Pan-End	`addGesture(GestureMap.Gestures.DRAG_END, "targetBeanId", "targetBeanAttribute")`
Check Mark/Tick	`addGesture(GestureMap.Gestures.TICK, "targetBeanId", "targetBeanAttribute")`
X-Mark/Cross	`addGesture(GestureMap.Gestures.CROSS, "targetBeanId", "targetBeanAttribute")`
Tap	`addGesture(GestureMap.Gestures.TAP, "targetBeanId", "targetBeanAttribute")`
Double- Tap	`addGesture(GestureMap.Gestures.DOUBLE_TAP, "targetBeanId", "targetBeanAttribute")`

The following example shows how you can associate a Check Mark / Tick gesture to a page:

```
OAPageLayoutBean pageLayoutBean = (OAPageLayoutBean)webBean;
   webBean.getGestureMap().addGesture(GestureMap.Gestures.TICK,
   "id_of_target_button", UIConstants.ON_CLICK_ATTR.toString()));

   //"webBean" is the web bean to which the Tick gesture is being attached.
   //"id_of_target_button" is the ID of the web bean that contains the information of
   //the action to taken, on detection of the Tick gesture.
   //"ON_CLICK_ATTR" is the attribute of target web bean that contains the
   //definition of action to be taken.
```

Usage Restrictions

- OA Framework only supports gestures on web beans that extend the OAWebBeanContainer Class.
- In the case of conflicting gestures, the default implementation of the gesture always overrides any new gesture association. For

example, OA Framework supports Swipe-Left and Swipe-Right to navigate between pages of a train. If you attempt to associate the Swipe-Left and Swipe-Right gestures at the page level to some other action, the default actions of navigation within a train will override any new actions you try to associate with these gestures.
- OA Framework supports out-of-the box navigation within a train using the Swipe-Left and Swipe-Right gestures only when a navigation bar web bean is present on the page under a page layout region's page button bar web bean.

Personalization Considerations

- None

Known Issues

- Gesture recognition does not work properly on Android Chrome browser versions 32 and above.

Related Information

- BLAF UI Guideline(s)
- Javadoc File(s)
    - `oracle.cabo.ui.collection.GestureMap`
    - `oracle.cabo.ui.collection.EventDefinition`
    - `oracle.cabo.ui.laf.base.xhtml.GestureRendererUtils`
- Lesson(s)
- Sample Code

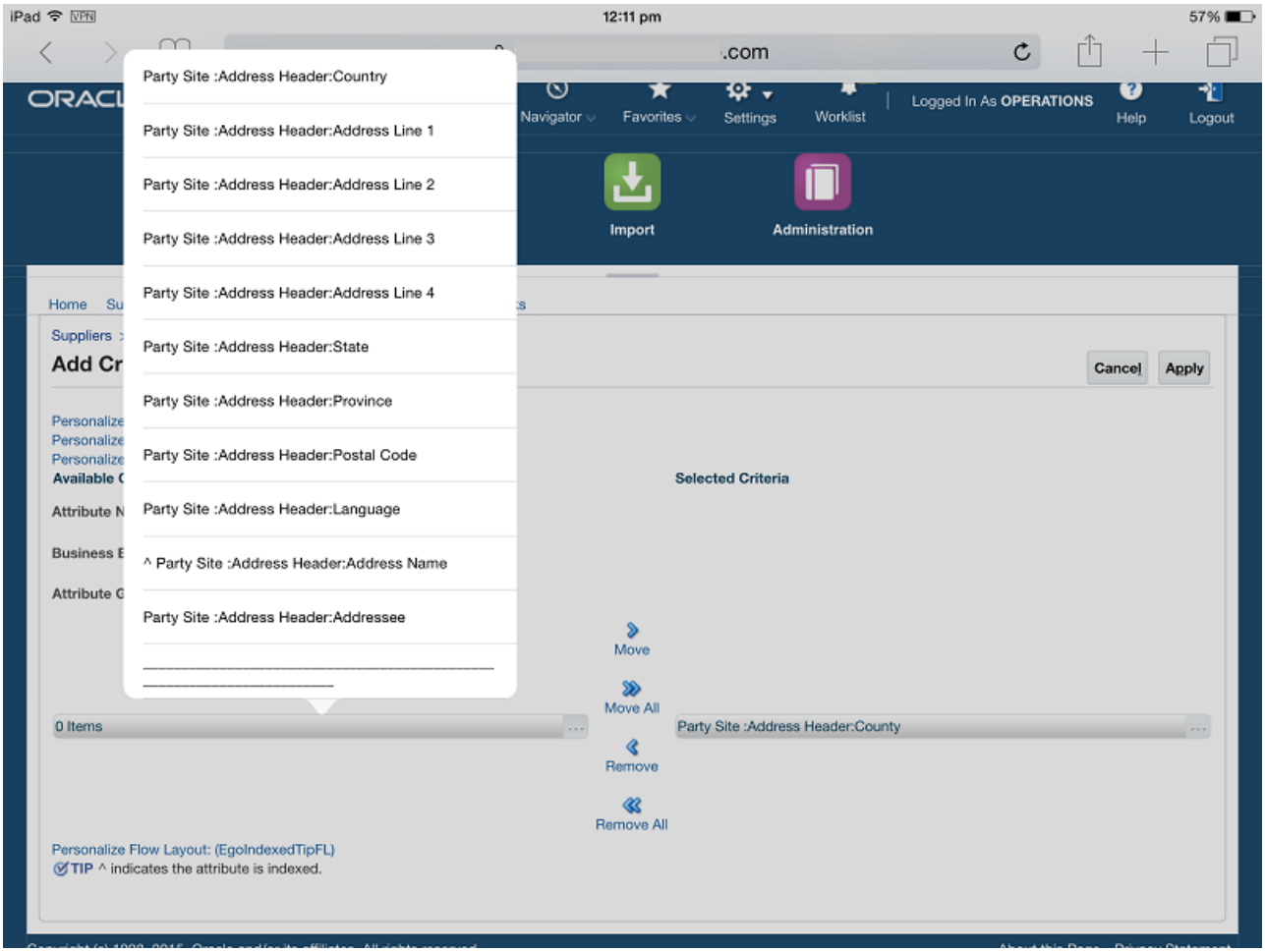
Caveats On Touch Devices

Visual Appearance

Shuttle

On an iPad, the shuttle lists appear as poplists with a menu pop-up to show the full list of values in each list of the shuttle.

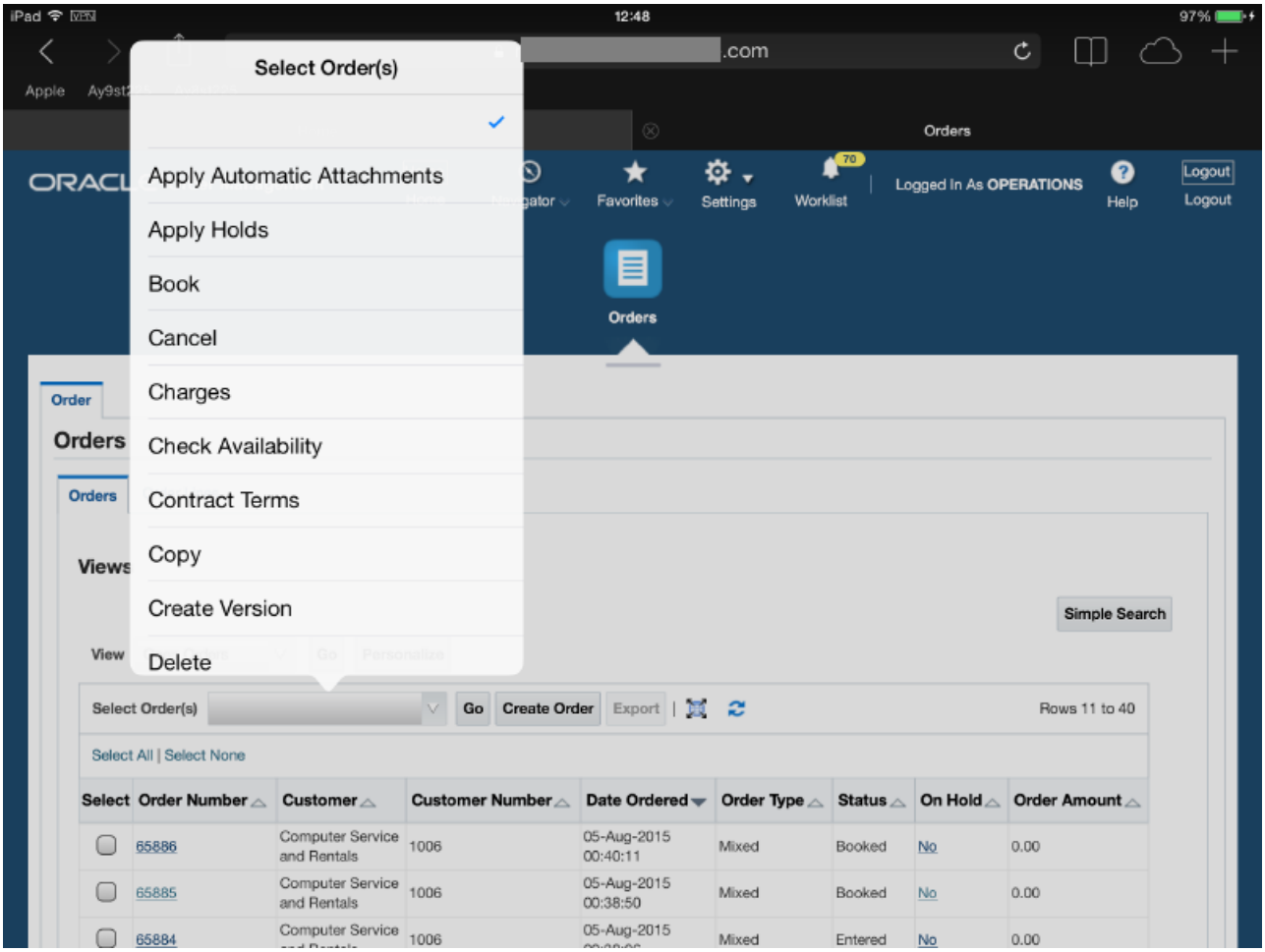
Figure 1. Shuttle list in Release 12.2.5 on the iPad.



Poplist

On an iPad, the poplist appears above the poplist field, and may extend beyond the page as shown in the figure below.

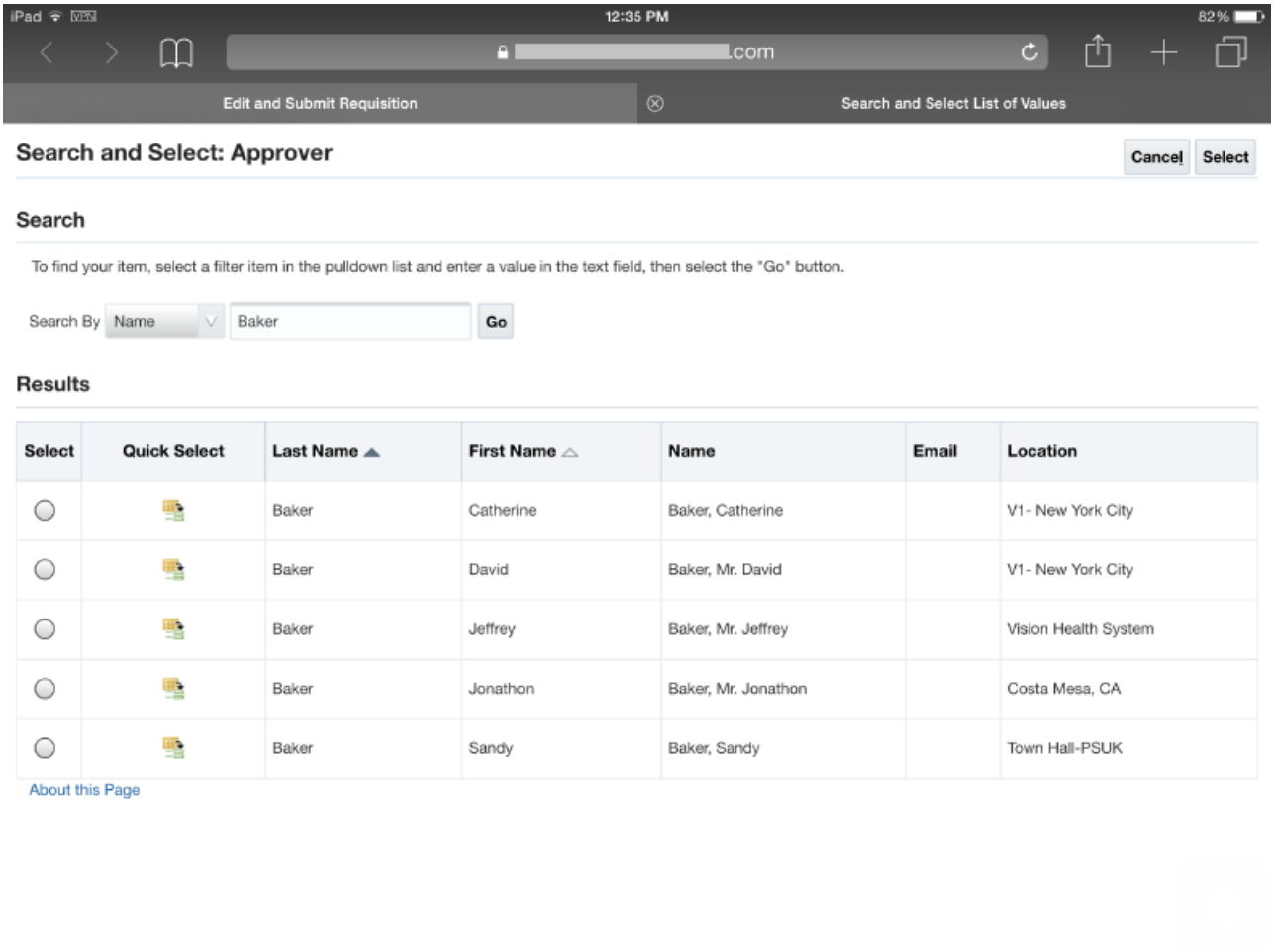
Figure 2. Poplist in Release 12.2.5 on the iPad.



List of Values (LOV)

The List of Values (LOV) component opens as a separate browser tab on tablets. This is because LOVs are modal browser windows and the display of such windows as a separate tab is expected behavior on tablet browsers such as Safari on IOS.

Figure 3. LOV open in a separate browser tab.



Touch Interactions

Accessing Inline Attachments

To access an inline attachment:

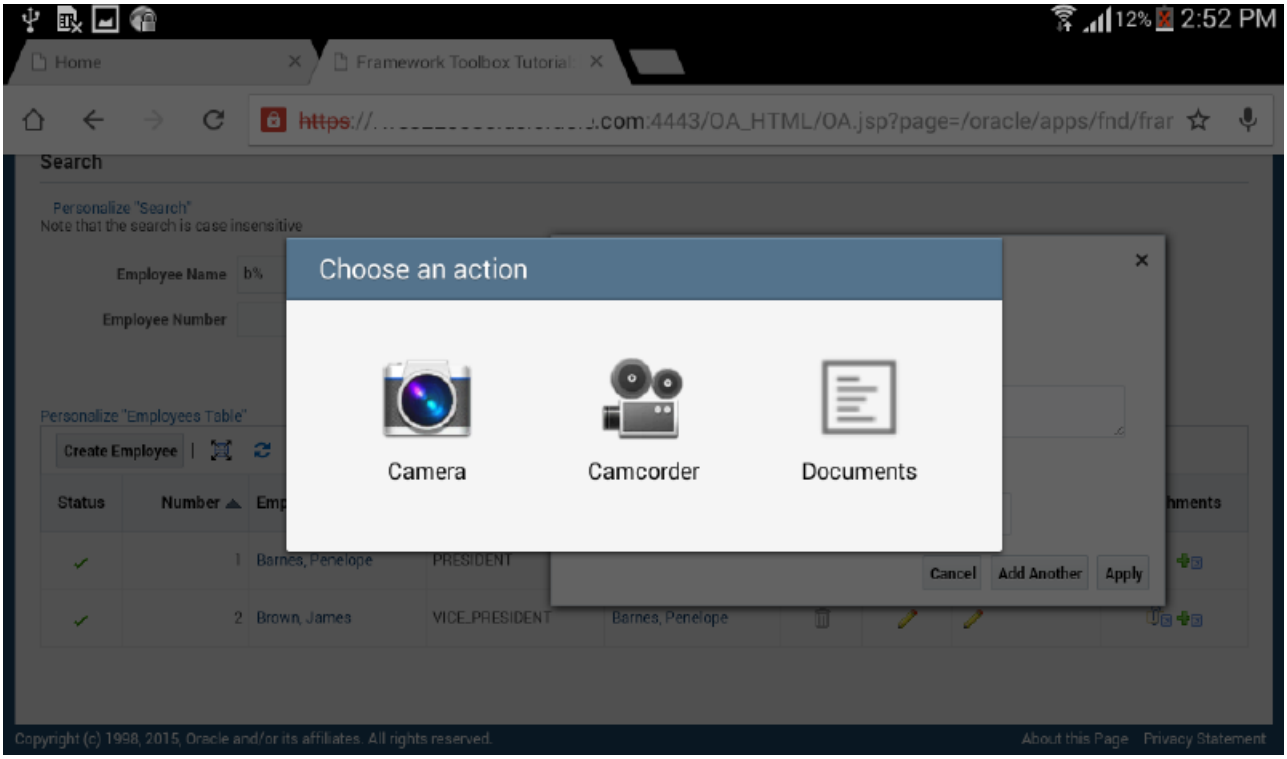
1. Tap on the **Add** or **View** attachment icon to open the inline pop-up window.
2. While the inline pop-up window is open, tap on the **Add** or **View** attachment icon again to redirect to the full Add Attachment or View Attachment page.

Adding Just-in-time Photos as Attachments

Users may take just-in-time photos of business objects with the cameras on their mobile devices and add those photos as attachments in Oracle E-Business Suite. In the Add Attachment or Update Attachment inline pop-up window, set the **Attachment Type** as **File**, then select the **Choose File** button to either select an existing file from the mobile device, or take a new photo with the camera on the mobile device and add that new photo as the attachment.

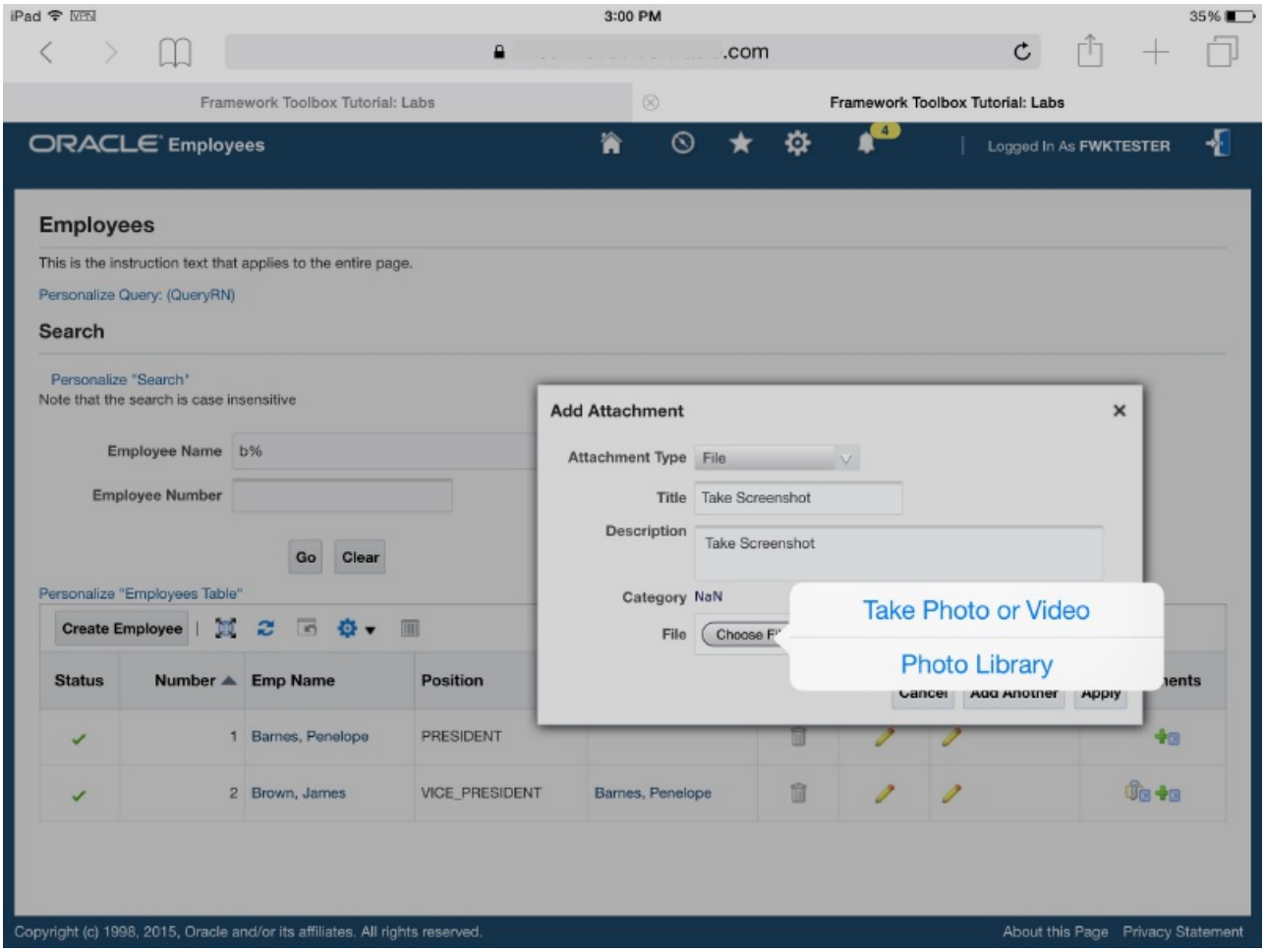
On Android devices, a **Choose an action** pop-up appears that allows you to select an existing document as the attachment file or select the camera or camcorder function to take a new photo or video to save as the attachment file.

Figure 4. Selecting the Choose File button in the Add Attachment pop-up on an Android device.



On iPads, you can select an existing file from the Photo Library or take a new photo or video to save as the attachment file. The Safari browser on iOS allows access only to whatever file attachments the platform supports for Safari. Currently, iOS allows access only to image files in the Photo gallery and camera. It does not support access to other file types such as PDF or text.

Figure 5. Selecting the Choose File button in the Add Attachment pop-up in Release 12.2.5 on an iPad.



Accessing Pop-ups

To open a pop-up window, tap on the item for which the pop-up is enabled. Items that may be pop-up-enabled include text (messageStyledText), images, links, and buttons. This initial tapping action is equivalent to the desktop behavior of hovering over an item to display the pop-up window. If the item has a destination URL set on it, then tapping on the item again while the pop-up is open navigates you to the destination page.

As of Release 12.2.5, you may drag the pop-up title bar to move the pop-up, and for a read-only pop-up, you may tap on any area of the browser window outside of the pop-up to automatically close the pop-up.

Disabled Form Functions in the Home Page

Form functions do not run on the touch devices supported by Oracle E-Business Suite.

If the profile **Self Service Personal Home Page Mode** is set to **Framework Tree,** the following behavior will be observed on the Oracle E-Business Suite Home page for the touch device:

- If a responsibility's submenu contains only form functions, that submenu does not render.
- if a responsibility's submenu contains a secondary submenu that contains only form functions, that secondary submenu does not render.
- If a responsibility contains only form functions, a "No pages found" message displays when you select the responsibility.
- Form functions set as favorites do not render in the **Favorites** drop down list on the Universal Global Header.

If the profile **Self Service Personal Home Page Mode** is set to **Framework Simplified,** the following behavior will be observed on the Oracle E-Business Suite Home page for the touch device:

- Favorites icons representing form functions do not render.
- Form functions set as favorites do not render in the **Favorites** drop down lists on the Universal Global Header.

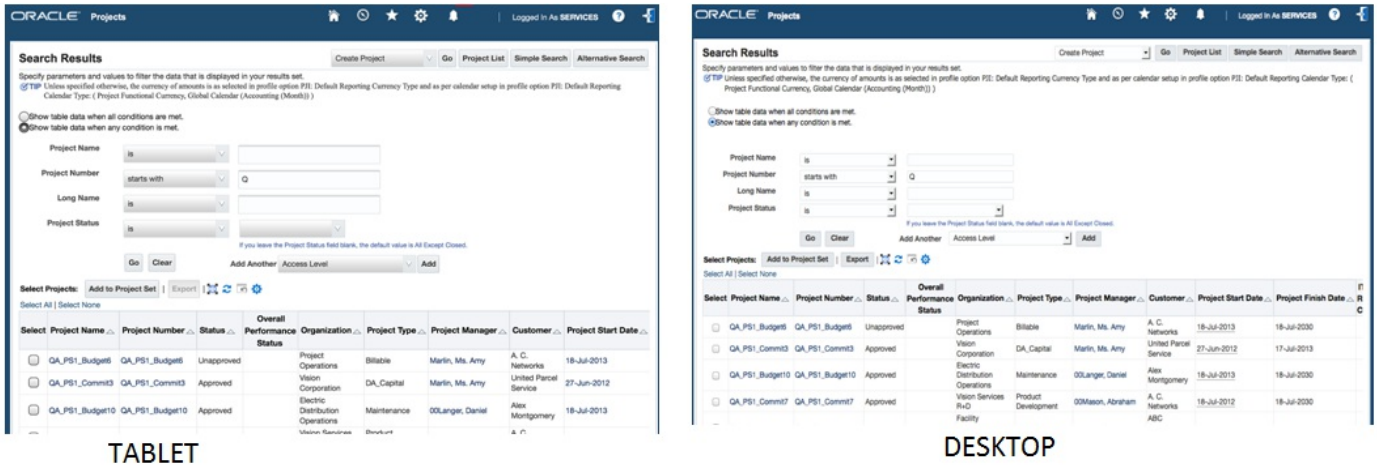
Look-and-Feel Optimizations for Touch Devices

OA Framework pages render with an optimized version of the Alta Look-and-Feel on touch devices. This optimized version contains

specific variations from the desktop version that enhance the usability of OA Framework-based pages on touch devices. The following table lists components that vary from the desktop version while rendering on touch devices:

Component	Variation
• messageTextInput • messageChoice • messageLOVInput • Buttons • Date Picker	Increased component height to allow for easier tapping into component field.
Table (Classic and Advanced)	Increased height to allow easier performance of rich interactions such as column reorder, resize, show / hide.
messageComponentLayout	Increased spacing between rows of the messageComponentLayout.

Figure 6. The optimized variations of the Release 12.2.5 Look-and-Feel between a tablet device and desktop:



Known Issues

- See a summary of key issues with suggested workarounds if available.

Notifications (Workflow Worklist)

Overview

The Oracle Workflow Worklist lets users view, respond to and maintain notifications. This document describes how to add the shared Worklist region to a page in your product. As shown in Figure 1 and Figure 2 below, you can add the Worklist in summary form to a page with other content, or you can include the full Worklist in its own page.

For additional information about the Workflow product, see the *Oracle Workflow Developer's Guide*, the *Oracle Workflow Administrator's Guide*, the *Oracle Workflow User's Guide* and the *Oracle Workflow API Reference*.

Figure 1: Example of the Worklist shared region added to a product Home page.

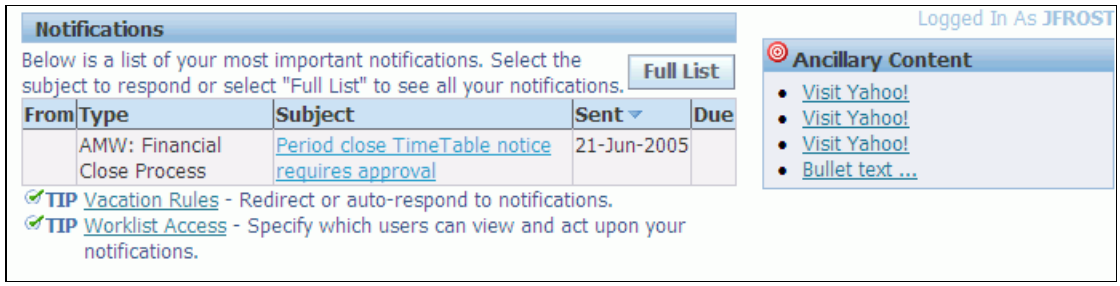
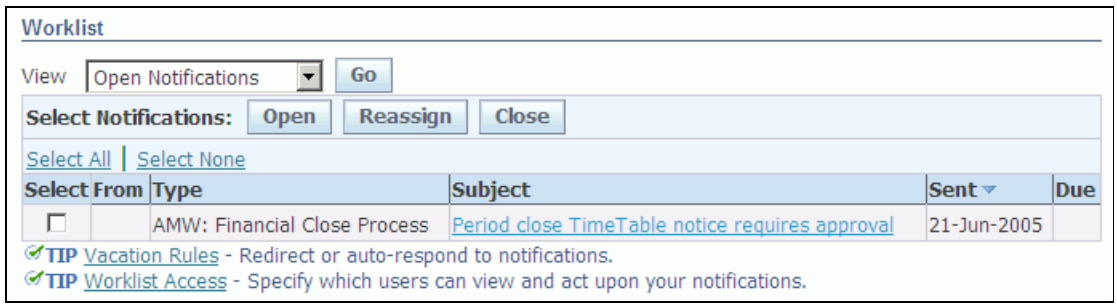


Figure 2: Example of the Worklist in its own page.



Implementation

To add a shared Workflow Worklist region to your page:

Step 1: Select the region to which you want to add the Worklist, right-click and select **New > Region**. Give the region a standards-compliant `ID`.

Step 2 (optional): If you want to include a header above the Worklist region as shown in Figure 1 above, set the `Region Style` to **header** and set the `Text` property to the value you want to display as the header text. Select the header region, right-click and select **New > Region**. Give this region a standards-compliant `ID` (the "Workflow Worklist" text in Figure 2 is set in the `Page Title` property in the **pageLayout** region).

Tip: If you need to reduce the size of the header text from its default rendering size, add the following `processRequest()` call to `setSize()` in a controller associated with the header region:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
 ((OAHeaderBean)webBean).setSize(1);
}
```

Step 3: Select the region that will extend the shared Notifications Worklist region and set its `Extends` property to `/oracle/apps/fnd/wf/worklist/webui/AdvancWorklistRG` (if JDeveloper displays a warning message regarding scope restrictions, simply acknowledge it and proceed). Note that the `Region Style` is automatically set based on the region that you are extending.

Step 4 (optional): If you are adding the Worklist region to a page where you want it to display in summary form as shown in Figure 1, you must add a **WFHomeWorklist** parameter to the request with a value of **Y**. For example, you could add a controller to your Worklist region (or to a ancester region) with the following `processRequest()` logic:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
```

```
{
  super.processRequest(pageContext, webBean);
  pageContext.putParameter("WFHomeWorklist","Y");
}
```

In this case, Oracle Workflow automatically handles the selection of the Full List button for you by rendering the standard **AdvancedWorklist** page with your menu/footer.

If you opt to display the Worklist region in its summarized form, *and you want to drilldown to a custom Worklist page when the user selects the Full List button*, then you must add the following logic to a controller associated with your Worklist region or an ancestor region:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);


  // You must set a session value WFFullListPage to the fully qualified
  // name of your custom full Worklist page.
  pageContext.putSessionValue("WFFullListPage",
          "/oracle/apps/fnd/framework/toolbox/tutorial/webui/ToolboxWorklistPG");

}
```

Previously, you had to also add the following logic to ensure that Oracle Workflow correctly rendered a "Return to" link with an appropriate destination in any Worklist pages displayed while the user was navigating in that module. Now, Oracle Workflow automatically configures the the "Return to" link to point to the page where the Worklist is embedded. If you need to override this behavior for any reason, you can also leverage the following example code.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);


  // You must set a session value WFWorklistPage to the fully qualified
  // name of the page that includes the shared Worklist.
  pageContext.putSessionValue("WFWorklistPage",
          "/oracle/apps/fnd/framework/toolbox/tutorial/webui/HomePG");
}
```

Personalization Considerations

- See a summary of Worklist personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guidelines
  - Notification Page Templates
- OA Framework Developer's Guide
  - OA Framework File Standards (Naming, Package Structure and Standard Content)
  - Worklist Personalization Considerations
- OA Framework ToolBox Tutorial / Sample Library
  - See the Home Page lab.
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.WorklistPG`
- Oracle Workflow Documentation
  - Oracle Workflow Developer's Guide
  - Oracle Workflow Administrator's Guide
  - Oracle Workflow User's Guide
  - Oracle Workflow API Reference

Page Access Tracking

Overview

Page Access Tracking allows administrators to track application usage statistics and perform web site traffic analysis. The Page Access Tracking infrastructure transparently (with negligible performance overhead) captures each user click with associated application rich context information for web-based Oracle E-Business Suite built on OA Framework or JTF technologies.

The Page Access Tracking administration UI provides a single location for administrators to enable and disable Page Access Tracking, configure the granularity to which application context information is captured, and view various reports on the gathered data. Some examples of the reports include:

1. Access reports for a given application, responsibility and/or user across OA Framework, JTF and Forms tech stacks.
2. Page Performance reports per mid-tier node.
3. Page access flow chart for a given user session.
4. Search reports based on several filter criteria.

Page Access Tracking reports aggregate data gathered for JTF and OA Framework-based applications with Sign-on audit data gathered for Forms-based applications. This enables administrators to view usage flow paths that span all three technology stacks.

Please see Knowledge Document 402116.1 for more information about this feature.

Deployment

Page Access Tracking is shipped with JTT.E and OA Framework to track CRM Applications and OA Framework Applications. The Configuration and Reports UI is currently shipped in JTT.E as part of the CRM HTML Admin Console. If JTT.E is installed in the environment, a system administrator can navigate to the UI from the CRM HTML Admin Console. Refer to Knowledge Document 402116.1 for details on navigating to the Configuration and Reports UI. If JTT.E is not installed in the environment, the Configuration and Reports UI will not be available. Page Access Tracking can still be configured to track OA Framework Applications by setting the profiles described below.

Please see Knowledge Document 402116.1 regarding Sign-on Audit Configuration.

Profile Options

There are five Profile Options used to configure Page Access Tracking. If JTT.E is installed in the system, configuring these five profiles can be done via the Configuration UI as described above. You should also refer to the Knowledge Document 402116.1 for further details on how to configure Page Access Tracking.

If JTT.E is not installed, these profiles can be set via the Oracle E-Business Suite System Profile Options form:

- **JTF_PF_MASTER_ENABLED**
  - Site level Profile
  - Master on/off switch for Page Access Tracking.
  - Possible values: **true**/**false**
- **JTF_PF_ENABLED**
  - Site/Application/Responsibility/User level Profile
  - Determines if Site/Application/Responsibility/User has Page Access Tracking turned on/off.
  - Possible values: **true**/**false**
- **JTF_PF_LEVEL**
  - Site/Application/Responsibility/User level Profile
  - As a bitmask this profile determines what information to log when Page Access Tracking is on for a particular access.
  - Possible values:

Value	Information Logged
22	Session Information (Client Browser, Language, and HTTP Header)
118	Session Information and Cookies
254	Session Information, Cookies, and URL Parameters
126	Session Information, Cookies, and all Parameters

- **JTF_PF_FLUSH_INTERVAL**
  - Site level Profile
  - Determines how often Page Access data is flushed to the database (sec).
  - Default value of **120** sec is used if Profile not set.
- **JTF_PF_BUFFER_SIZE**

- Site level Profile
- Determines how often Page Access data is flushed to the database (number of page accesses).
- Default value of **20** page accesses is used if Profile not set.

Personalization Considerations

- None

Known Issues

- None

Related Information

- Knowledge Document 402116.1

Page Contents Bottom Line (the 'Ski')

The "ski" graphical component that renders above all page footer links to identify the end of a page is obsolete as of Release 12.

Page Footer

Overview

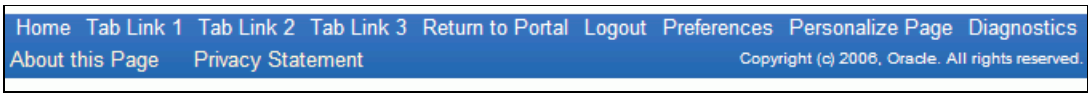
Per the BLAF UI Guideline: Page Footer specification, the page footer marks the end of a page's contents. A footer includes the following components:

- A link for each of the top-level application tabs and global buttons
- Oracle copyright information
- (Optional) A privacy statement link
- (Optional) An "About" link to information about the current application/page

Tip: For information on adding action/navigation buttons and a "Return to..." link above the footer, see Buttons (Action/Navigation) and Buttons (Link) respectively.

The footer content is truly the last content that a page includes, and as such, it renders below the page contents and any associated action/navigation buttons or the "Return to..." link. A footer example including all optional components is shown below:

Figure 1: Example of a Page Footer Including All Optional Components



Declarative Implementation

The links that duplicate your top-level tabs and global buttons render automatically in all OA Framework pages.

Standard Copyright and Privacy

To include the standard Oracle E-Business Suite copyright and privacy statement links, simply set the `Auto Footer` property to **True** on your **pageLayout** region.

Custom Copyright and/or Privacy

If you want to include custom privacy statement or copyright, ensure the **pageLayout** region's `Auto Footer` property is set to **False**.

To add a custom copyright:

Step 1: Select your **pageLayout** region in the Structure pane, right-click and select **New > Copyright**. JDeveloper creates a **messageStyledText** item for you (note that you cannot change the style).

Step 2: Specify your item's `ID` in accordance with the OA Framework File Standards (Naming, Package Structure and Standard Content) and set the `Prompt` property to the copyright text, or use a Message Dictionary message.

To add a custom privacy statement:

Step 1: Select your **pageLayout** region in the Structure pane, right-click and select **New > Privacy**. JDeveloper creates a **link** item for you.

Step 2: Specify your item's `ID` in accordance with the OA Framework File Standards (Naming, Package Structure and Standard Content), set the `Text` property to the link label and specify the `Destination URI` for your privacy document.

"About" Page

Each OA Framework page automatically includes an "About" link in the footer if the Diagnostics or Administrator Personalization features are enabled. See Discovering Page, Technology Stack and Session Information.

Runtime Control

Although you should never manipulate footer elements programmatically as this precludes personalization, there are methods in the `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean` to get and set the text for the privacy statement and copyright links. You can also set the privacy statement target.

Note: The footer components are added to your page as special named children, not indexed children. That means you can't get a handle to them using `findIndexedChildRecursive` in the `oracle.apps.fnd.framework.webui.OAPageContext` class as you would for the regions and items that comprise the main page content. Use `findChildRecursive()` instead.

Personalization Considerations

- None

Known Issues

- None

Related Information

- BLAF UI Guidelines
  - Page Footer
- Developer's Guide:
  - Buttons (Action/Navigation)
  - Buttons (Links)
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`
  - `oracle.apps.fnd.framework.webui.OAPageContext`
- ToolBox Tutorial / Sample Library

Page Layout (How to Place Content)

Overview

This document is intended to help you understand and control the behaviors of the different layout regions in OA Framework, and suggests layout solutions for common page designs.

Contents

- Region Style: header
- Region Style: messageComponentLayout
- Region Style: stackLayout
- Region Style: flowLayout
- Region Styles: tableLayout / rowLayout / cellFormat
- Region Styles: defaultSingleColumn / defaultDoubleColumn
- Common Layouts and Suggested Solutions
  - Page Start and Page End
  - Search and Results Page
  - Single Column Display-Only Details Page
  - Single Column Update Page
  - Master-Detail Page w/ Adjacent Headers
  - Home Page
  - Multiple Items with a Single Prompt
  - Image and Text in a Table Column

Region Style: header

The `oracle.apps.fnd.framework.webui.beans.layout.OAHeaderBean` (**header** region style) renders text and a decorative line that spans the region underneath the header text as shown in Figure 1.

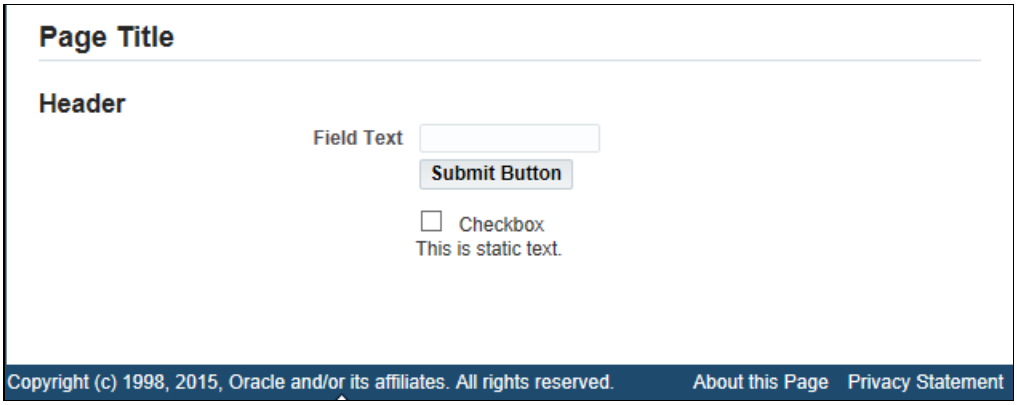
Most regions and items do not automatically indent when added to a header. If, for example, you add a button item to a header region, the button start-aligns immediately beneath the header as shown below. Similarly, if you add a header to a header, the parent header appears as text with a blue background and the child header appears immediately beneath, aligned as underlined text. Only third-level nested content automatically indents from the top parent header.

Figure 1: Example of a header region with items added directly to it. Note that the items are not indented.



Since headers don't automatically indent their content, add an intermediate region to the header to achieve the appropriate indentation. The messageComponentLayout region, discussed next, is the recommended solution for this. Figure 2 shows the change in component placement when we add a **messageComponentLayout** region to the header, and then add the items to the **messageComponentLayout**.

Figure 2: Example of a header region with a messageComponentLayout added to it to hold the items.



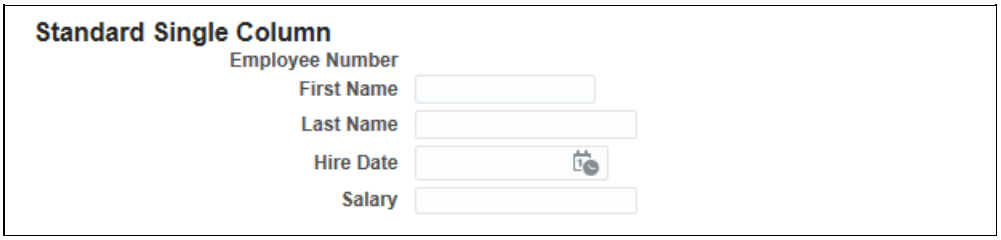
To display regions or items that are not to be indented, such as instruction text and tables, add them directly to your header region.

See Headers and Subheaders for additional information about this component.

Region Style: messageComponentLayout

The `oracle.apps.fnd.framework.webui.beans.layout.OAMessageComponentLayoutBean` (`messageComponentLayout` region style) can be used to quickly and correctly position components into an N-column grid as shown in Figures 3 and 4 below. Both the component spacing and tab sequence (down and then across for multiple columns) fully comply with the BLAF UI guidelines.

Figure 3: Single column messageComponentLayout.



Tip: For those of you who are familiar with the spacing in the OADefaultSingleColumnBean and OADefaultDoubleColumnBean regions, the **messageComponentLayout** yields a very different result (fields are far more indented). The **messageComponentLayout** spacing is correct per the current UI guidelines.

Figure 4: Double column messageComponentLayout.



With this component, you can control the number of rows (items) that you display in a column, as well as the number of columns. You can define as many columns as you need -- although there is a practical UI usability limit. Typically, you should display no more than 3 columns.

Declarative Implementation

Step 1: (optional) To use the **messageComponentLayout** to position content immediately beneath a header as shown in Figure 5 below, first add a **header** region to your page as the **messageComponentLayout** is not capable of rendering the header text and line. Set the **header** region's `Text` property as needed.

Figure 5: Single column messageComponentLayout used in a manually created search region with a header and instruction text.



Step 2: (optional) To get instruction text to render above the fields in the **messageComponentLayout**, as shown in Figure 3, add the corresponding item to the *containing* region.

Note: If you add the instruction text to the **messageComponentLayout**, it will render indented and aligned with the fields.

If you have a **header** region above the **messageComponentLayout**, create the following structure:

```
header
| -- staticStyledText
| -- messageComponentLayout
```

Step 3: Add a new region to your page, set its `Style` to **messageComponentLayout** and assign it a standards-compliant `ID`.

Step 4: Select the **messageComponentLayout** region, right-click and select **New** in the context menu. JDeveloper displays all the **message*** beans, including a **messageLayout** region, for quick selection. To add any **non-message*** beans to your **messageComponentLayout**, such as a **submitButton** or a region, first add the **messageLayout** region, then select the **messageLayout** region and add your item. The page structure required to achieve the layout shown in Figure 1 is:

```
header
| -- staticStyledText
| -- messageComponentLayout
     | -- messageLovInput
     | -- messageChoice
     | -- messageCheckBox
     | -- messageLayout
          | -- submitButton
```

Tip: All of the items that you add to the **messageComponentLayout** region are added as indexed children. By default, the **messageComponentLayout** renders its items in a single column.

Tip: Refer to Aligning Attachment Prompts in a messageComponentLayout for guidance on how to align the prompt of a **attachmentLink** or **messageInlineAttachment** item style with respect to other elements in a **messageComponentLayout** region.

Step 5: (optional) To obtain a multi-column layout, configure the `Columns` and `Rows` properties according to the following rendering rules.

- The `Columns` property controls the maximum number of columns you want to render.
- The `Rows` property controls how many items should render in any given column before populating the next column. However, this property is ultimately governed by the `Columns` property. If, for example, you set the `Rows` property to **2** and the `Columns` property to **2**, but you add **6** fields to the **messageComponentLayout** region, your layout will appear as two columns with the first containing objects 1 - 3 while the second contains objects 4 - 6.

Figure 6: messageComponentLayout with Rows = 2, Columns = 2 and 6 fields.



If you set the `Columns` property to **3** and the `Rows` property to **2** for the same 6 fields, the resulting layout will render as three columns with the first column containing objects 1 and 2, the second column containing objects 3 and 4, and the third column containing objects 5 and 6, as shown in Figure 7 below:

Figure 7: messageComponentLayout with Rows = 2, Columns = 3 and 6 fields.



Note: You must set *both* the `Columns` and `Rows` properties to achieve the multi-column layout. If you set only the `Columns` property, your items will render in a single column regardless of what value you specify.

Tip: If you set the `Rendered` property of an indexed child to **False**, the remaining items collapse vertically to absorb the missing component. See the Dynamic User Interface documentation for information about controlling item rendering at runtime. For example, if you hide the Field 4 item in the layout shown above, the result renders as:

Figure 8: messageComponentLayout with Rows = 2, Columns = 3 and 6 fields with Field 4's Rendered property set to False.



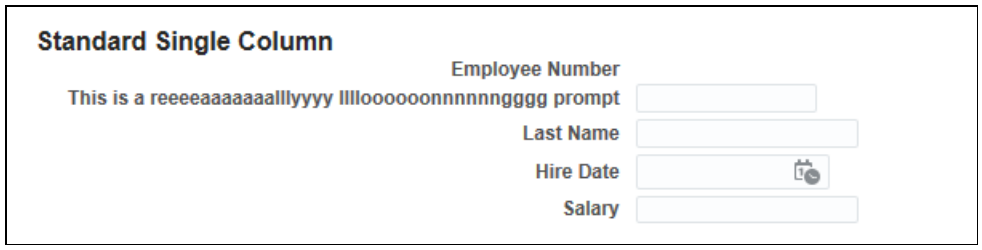
Step 6: (optional) To change the default layout, you may set the `Width`, `Prompt Width`, and `Field Width` properties as needed to achieve the desired result. These values may be set as absolute pixels (**100**) or as a percentage (**100%**).

Note: Since the default spacing complies with the UI guidelines, confer with the UI team to verify that your overrides are desirable and correct.

- The `Width` property controls the amount of available space used by the entire region.
- The `Prompt Width` property controls the *preferred* width of the prompts. If specified as a percentage, the `Prompt Width` plus the `Field Width` should add up to 100%, regardless of the number of columns. If the `Prompt Width` is specified as a percentage, OA Framework derives the `Field Width` if it is not specified.
- The `Field Width` property controls the *preferred* width of the fields. If specified as a percentage the `Prompt Width` plus the `Field Width` should add up to 100%, regardless of the number of columns. If the `Field Width` is specified as a percentage, OA Framework derives the `Prompt Width` if not specified.

Note: If prompt or field lengths exceed the *preferred* amount of space requested by these properties, the preferences will be ignored and the layout adjusted so the values render properly. For example, in Figure 9, the `Prompt Width` property is set to 15%, however, the very long prompt exceeds this space allocation so the layout is adjusted to accommodate it.

Figure 9: Example of a very long prompt exceeding a Prompt Width value of 15%.



Step 7: (Required only if you are using the **messageComponentLayout** in a **search** region) The BLAF UI guidelines stipulate special spacing parameters for components in **search** regions. Although you can override the individual `Width`, `Prompt Width` and `Field Width` properties to satisfy this requirement, you should instead set the `Search Region` property to **True**. OA Framework automatically adjusts the default spacing to correctly position your fields.

Note: You can also call the `setSearchRegion(true)` method on your `OAMessageComponentLayoutBean`.

Aligning Attachment Prompts in a messageComponentLayout

The following instructions describe how to place an **attachmentLink** or **messageInlineAttachment** item style in a **messageComponentLayout** region so that it's prompt aligns properly with respect to other elements of the layout region.

Step 1. Create a **messageLayout** region as a child of the **messageComponentLayout** region.

Step 2. In the **messageLayout** region, set the `Prompt` property value to **Attachments** or some other value.

Step 3. Create an **attachmentLink** or a **messageInlineAttachment** item under the **messageLayout** region.

Step 4. For proper horizontal alignment of the attachment item, set the `Prompt` property of the **attachmentLink** or **messageInlineAttachment** item as null.

Step 5. By default, the **messageLayout** region 'top' aligns its prompt and the **attachmentLink** and **messageInlineAttachment** items 'middle' align their prompts. The only way to resolve these vertical alignment differences is by programmatically setting the vertical alignment of the **messageLayout** region, as shown in the code example below.

```
OAMessageLayoutBean msgLayout =
  (OAMessageLayoutBean)webBean.findIndexedChildRecursive("<MessageLayoutBeanID>");
msgLayout.setAttributeValue(V_ALIGN_ATTR, "middle");
```

Runtime Control

There is no common reason to interact with a **messageComponentLayout** programmatically, except to call the `setSearchRegion(true)` method as described in Step 7 above or to correct the vertical alignment of an **attachmentLink** and **messageInlineAttachment** item in a **messageLayout** region.

Personalization Considerations

- None

Known Issues

- See a summary of key Page Layout issues with suggested workarounds if available.

Region Style: stackLayout

The `oracle.apps.fnd.framework.webui.beans.layout.OAStackLayoutBean` (**stackLayout** region style) is a simple utility layout that adds regions and items in a vertical "stack" without any indentation. For example, in Figure 10, a **stackLayout** region is added directly to a **pageLayout** region, and the items are added to the **stackLayout**.

Figure 10: Stack region with items added directly beneath the pageLayout region of a page.



Since most items need to be indented, you'll typically use a **stackLayout** only for cases where you want to explicitly position regions and/or items vertically without the indentation. For example, in the Home page example shown in Figure 17 below, we use a **stackLayout** to position the two content containers on the right side of the page.

Note: Do not use the deprecated `oracle.apps.fnd.framework.webui.beans.layout.OADefaultStackLayoutBean` (**defaultStack** region style) or the `oracle.apps.fnd.framework.webui.beans.layout.OADefaultFormStackLayoutBean` (**defaultFormStack** region style).

Declarative Implementation

Create a new region, assign it a standards-compliant `ID`, set its `Region Style` to **stackLayout**, and add your items.

Personalization Considerations

- None

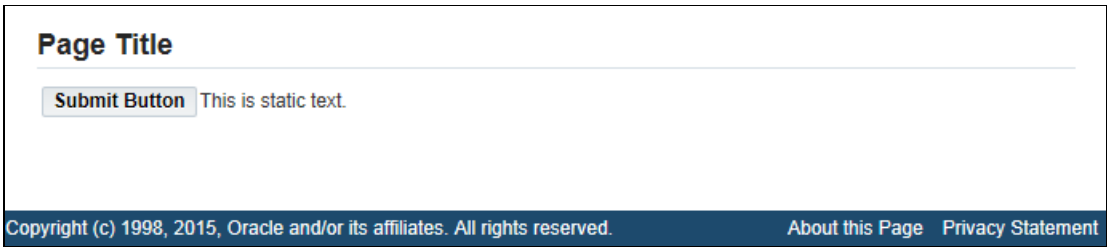
Known Issues

- None

Region Style: flowLayout

The `oracle.apps.fnd.framework.webui.beans.layout.OAFlowLayoutBean` (**flowLayout** region style) is another simple utility layout that adds regions and items in a horizontal flow. The items that you add are not indented, and OA Framework does not add any horizontal space between items.

Figure 11: Example of items in a flowLayout region.



Note: Certain components "wrap" when added to a **flowLayout**. Instead of rendering horizontally, they render vertically. All the **message*** items exhibit this behavior because they are contained in a **tableLayout** region, and a **tableLayout** region by itself does the same thing.

See Image and Text in a Table Column for a good use case for a **flowLayout**.

Declarative Implementation

Create a new region, assign it a standards-compliant `ID`, set its `Region Style` to **flowLayout**, and add your items.

Personalization Considerations

- None

Known Issues

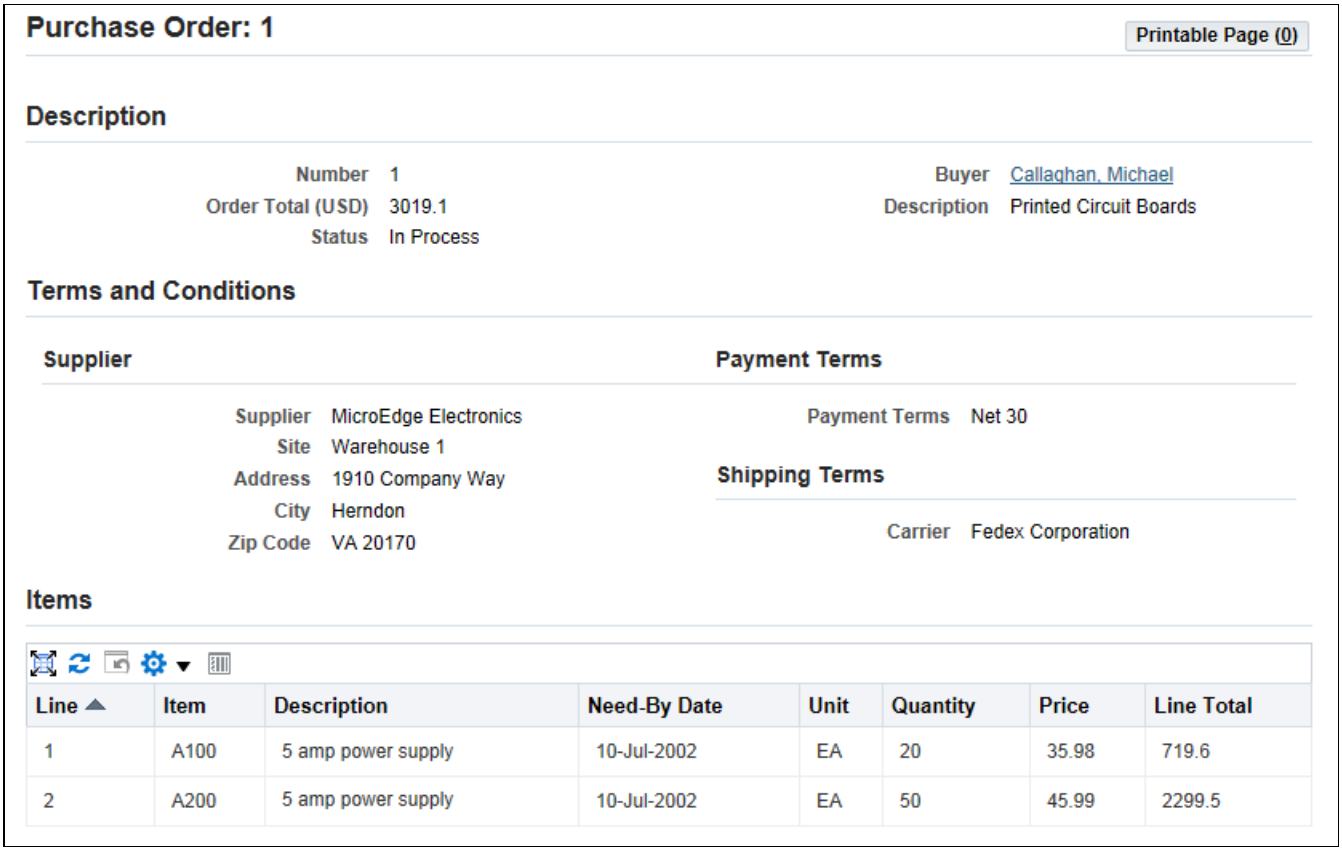
- None

Region Styles: tableLayout / rowLayout / cellFormat

A **tableLayout** used with the **rowLayout** and **cellFormat** regions, affords fine-grained control over your user interface. You typically use this when you need to accomplish something that you can't achieve using the other layout regions.

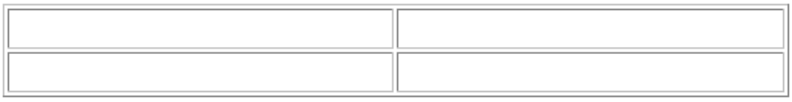
Ultimately, any complex HTML user interface is implemented as a series of tables within tables. If you're unfamiliar with HTML, it's often helpful to conceptualize your layout in a tool like Macromedia Dreamweaver (or any WYSIWYG HTML editor). For example, assume you need to implement the following adjacent headers beneath the Terms and Conditions header:

Figure 12: Example of a tableLayout implementation.



To conceptualize this in an HTML editor, we'll start by creating a simple 2 row, 2 column table in Dreamweaver. The result is shown in Figure 13.

Figure 13: 2 row, 2 column table in Dreamweaver after initially creating it.



Now, we'll add some content roughly where we want it to render, but we won't set any formatting properties. Figure 14 shows a result that is close to our layout, but the formatting is off: the Shipping Terms doesn't render next to the Supplier content even though we have room, and all the headers are "floating" in their respective cells.

Figure 14: 2 row, 2 column table in Dreamweaver after adding content to three of the four cells.



The first thing to fix is the location of the Shipping Terms header. To force this to render beside the Supplier header, we merge the Supplier cell with the empty cell below it by creating a "row span." After making this change, our table appears as shown in Figure 15.

Figure 15 : 2 row, 2 column table in Dreamweaver after adding adding a row span format to Supplier cell.

Supplier Header ----------	Payment Terms Header ----------
Field Field Field Field Field	Field
	Shipping Terms Header ---------- Field

Next, we fix the table cell content alignment by setting their vertical alignment properties to "Top." As you can see in Figure 16, we've arrived at a layout that mirrors what we want to achieve in our UI.

Figure 16 : 2 row, 2 column table in Dreamweaver after adding adding a rowspan format to Supplier cell and setting the vertical alignment of all the cells to the Top.

Supplier Header ----------	Payment Terms Header ----------
Field Field Field Field Field	Field
	Shipping Terms Header ---------- Field

The corresponding HTML looks like this (to highlight the table structure, cell content has been replaced by ellipses):

```
<table>
  <tr valign="top">
    <td rowspan="2">
    ... <!-- Supplier cell -->
    </td>
    <td valign="top">
    ... <!-- Payment terms cell -->
    </td>
  </tr>
  <tr>
    <td valign="top">
    ... <!-- Shipping terms cell -->
    </td>
  </tr>
</table>
```

To recap:

- We created a table with two columns and added two rows.
- We then merged two of the cells by setting a rowspan of 2 to end up with three content cells. Two of these cells are part of the first row: "Supplier" and "Payment Terms." "Shipping Terms" is part of the second row (when you merge cells, the resulting cell belongs to whatever the *top row* was before you did the merge).
- All three cells have their vertical alignment set to "Top."

For the table to fill the available horizontal space, set its width property to 100%.

The next section describes how to create a generic **tableLayout** region, and then follows with the specific implementation of this example.

Declarative Implementation

To add a basic **tableLayout** to your page:

Step 1: Create a new region, assign it a standards-compliant `ID`, and set its `Region Style` to **tableLayout**.

- For the tableLayout to fill whatever space it's rendering in, set its `Width` to 100%. if you don't do this, you may find that your content doesn't start/end align as you might expect it to since this region does not automatically expand to fill available space. You can also specify a number of pixels, such as 300, or a smaller percentage, if necessary.

Step 2: Select your **tableLayout** region, right-click and select **New > Region**. Assign this region a standards-compliant `ID` and set its `Region Style` to **rowLayout**. Repeat this step for all rows that you want to add.

Step 3: Select your **rowLayout**, right-click and select **New > Region**. Assign this region a standards-compliant `ID` and set its `Region Style` to **cellFormat**. Repeat this step for each row in your **tableLayout**.

Step 4: Select your **cellFormat**, and add whatever region or item you want to display.

- If you need to control the alignment of the web bean displayed within the **cellFormat**, set its `Vertical Alignment` and `Horizontal Alignment` properties as needed. For example, if you want a text field to be start justified at the top of the **cellFormat** region, set the `Vertical Alignment` to **top** and the `Horizontal Alignment` to **start**.

  Note: This does NOT control the alignment of content *within* the web bean. For example, if the start aligned text field's `Data Type` is **NUMBER**, the text field value will be end aligned.

- To change the column and/or row span for the **cellFormat** (as in the example below), set the `Column Span` and `Row Span` properties to a whole number.

- If you add two or more **cellFormats** to a **rowLayout** and add a subheader under each **cellFormat**, a continuous underline will appear across the resulting adjacent subheaders. To delineate each of the subheaders so that they display separate underlines, add a dummy **cellFormat** between the two adjacent **cellFormats**.

Guidelines for using tableLayout, rowLayout and cellFormat

1. A **tableLayout** must always have only **rowLayouts** or message web beans as its children.
2. A **rowLayout** should always have **cellFormats** as children. Furthermore, **rowLayouts** always need to be under a **tableLayout**.
3. A **cellFormat** can have any other web bean except rowLayout and cellFormat under it.

Any variation of these guidelines can lead to cases where a browser may not render the page correctly. Please refer to the OA Framework view coding standards for more information.

Example Implementation

Now that you know how to create a **tableLayout** and its contents, let's look at the example that we started working through above and translate the HTML into an OA Framework layout.

Step 1: Create a **tableLayout** region and set its `Width` to **100%**.

Step 2: Add two **rowLayout** regions to the **tableLayout** region.

Step 3: Add two **cellFormat** regions to the top row (these will be used for the Supplier and Payment Terms content). Add one **cellFormat** region to the bottom row (this will be used for the Shipping Terms content).

Step 4: Select the Supplier **cellFormat** and set its `Row Span` property to **2**. Set its `Vertical Alignment` property to **top** and its `Horizontal Alignment` property to **start**.

Step 5: Select the Payment Terms **cellFormat** region and set its `Vertical Alignment` property to **top** and its `Horizontal Alignment` property to **start**. Repeat for the Shipping Terms cell.

Step 6: Add the content that you want to render in each cell. For this particular layout, the structure appears as follows -- structurally similar to the HTML we created above.

```
pageLayout
| -- header ("Description")
|    | -- messageComponentLayout
|         | -- messageStyledText ("Number")
|         ...
| -- header ("Terms and Conditions")
|     -- tableLayout
|        | -- rowLayout
|             | -- cellFormat
|                  | -- header ("Supplier")
|                       | -- messageComponentLayout
|                            | -- messageStyledText ("Supplier")
|                            ...
|             | -- cellFormat
|                  | -- header ("Payment Terms")
|                       | -- messageComponentLayout
|                            | -- messasgeStyledText ("Payment Terms")
|             | -- rowLayout
|                  | -- cellFormat
|                       | -- header ("Shipping Terms")
```

```
|                              | -- messageComponentLayout
|                                   | -- messageStyledText ("Carrier")
| -- header ("Items")
     | -- table
```

See the Home Page example below for another **tableLayout** use case.

Personalization Considerations

- None

Known Issues

- None

Region Styles: defaultSingleColumn / defaultDoubleColumn

Before the **messageComponentLayout** region was introduced, the **defaultSingleColumn** (`oracle.apps.fnd.framework.webui.beans.layout.OADefaultSingleColumnBean`) and **defaultDoubleColumn** (`oracle.apps.fnd.framework.webui.beans.layout.OADefaultDoubleColumnBean`) regions were the primary recommended tools for positioning **message*** components in single and double column layouts. Given that these components incorporate complex, fixed rules for item placement, and they don't fully comply with the current UI spacing guidelines, use the **messageComponentLayout** region instead for new development.

These layout beans have been deprecated in favor of the `OAMessageComponentLayoutBean`.

Common Layouts and Suggested Solutions

If you look at the templates section in the Oracle Browser Look-and-Feel (BLAF) UI Guidelines, you'll note that most pages use a combination of layouts as illustrated here.

For specific component-level implementation details, see the other topics in Chapter 4. Also see the OA Framework ToolBox Sample Library for additional examples.

Page Start and Page End

For the purposes of adding content to a page, it is important to understand that each page has three potential content areas beneath the menu:

- "Start" named child -- set by calling `pageContext.getPageLayoutBean().setStart(<a Bean>)`. See the `oracle.apps.fnd.framework.webui.OAPageContext` Javadoc.

  Note: In American applications, this renders on the left side of the page.

- "End" named child -- set by calling `pageContext.getPageLayoutBean().setEnd(<a Bean>)`.

  Note: In American applications, this renders on the right side of the page.

- Everything Else -- this is the content that you add as children of your **pageLayout** region. If you don't explicitly set the "start" and "end" components then "everything else" fills up all the horizontal area in your page.

See the Home page example below. The "Ancillary Content" and "Even More Content" content containers have been added to the "End" named child, and the "Search / Quick Links" region has been added to the "Start" name child. In Figure 18, we don't set the "End" named child, and in Figure 19, we don't set the "Start" named child.

Figure 17: Example of a Home page with "Start" and "End" content in addition to the regular pageLayout content.

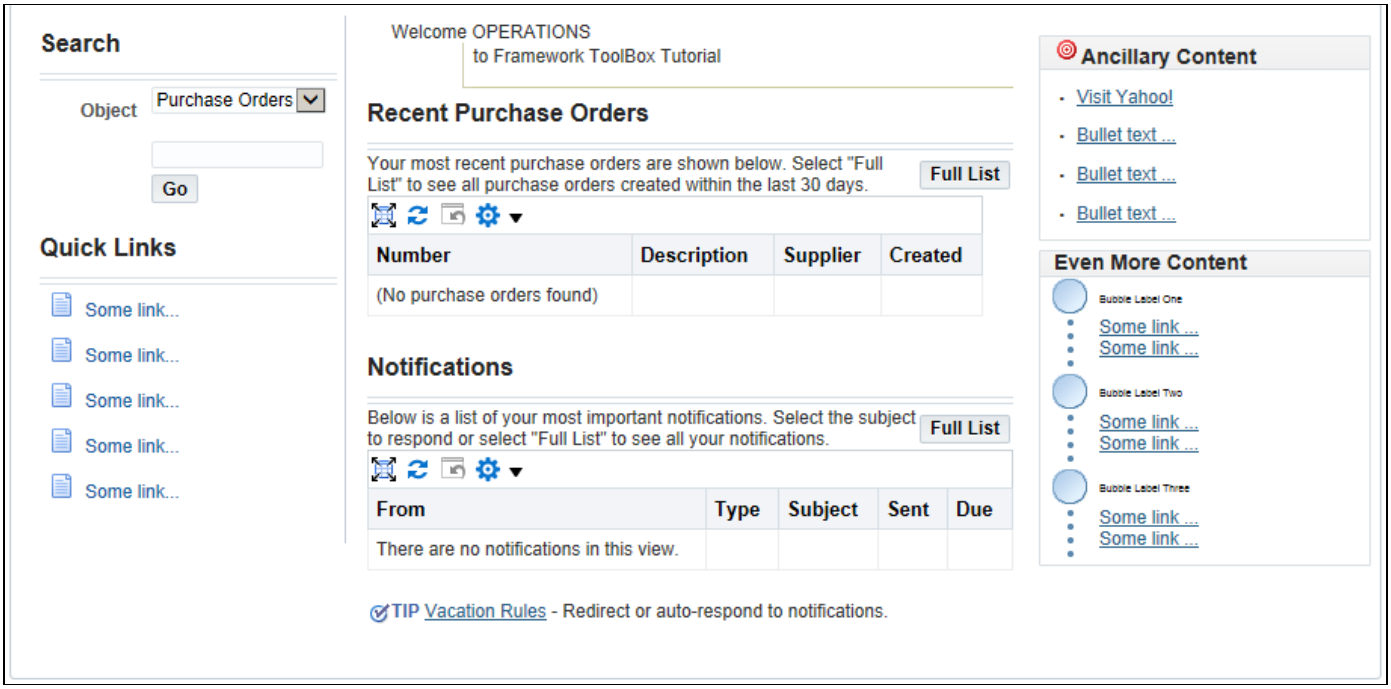


Figure 18: Example of the same Home page with only "Start" and pageLayout content.

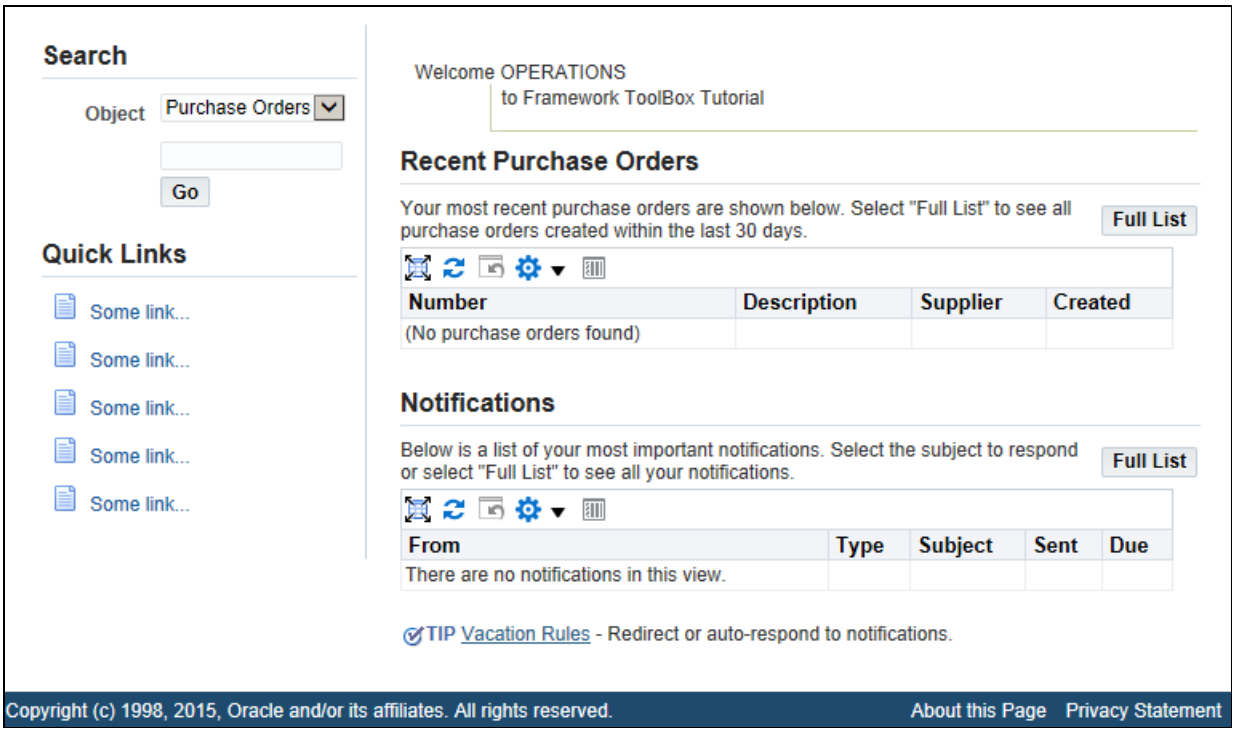
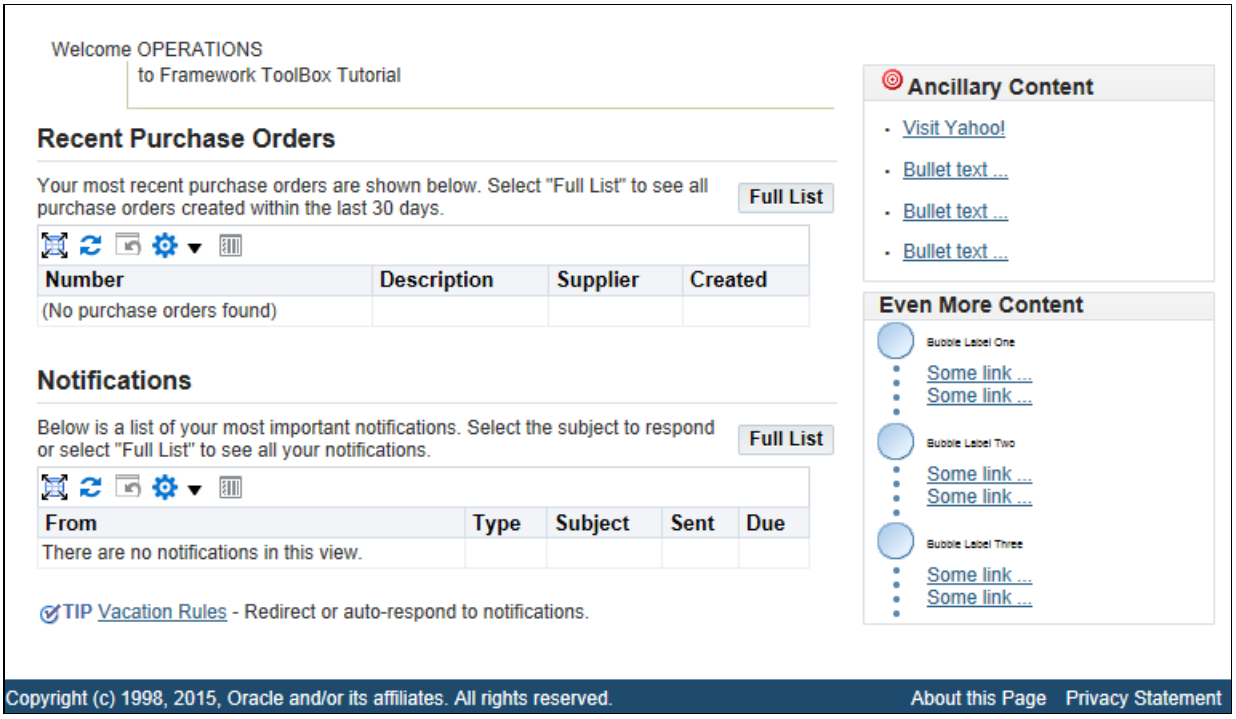


Figure 19: Example of the same Home page with only "End" and pageLayout content.

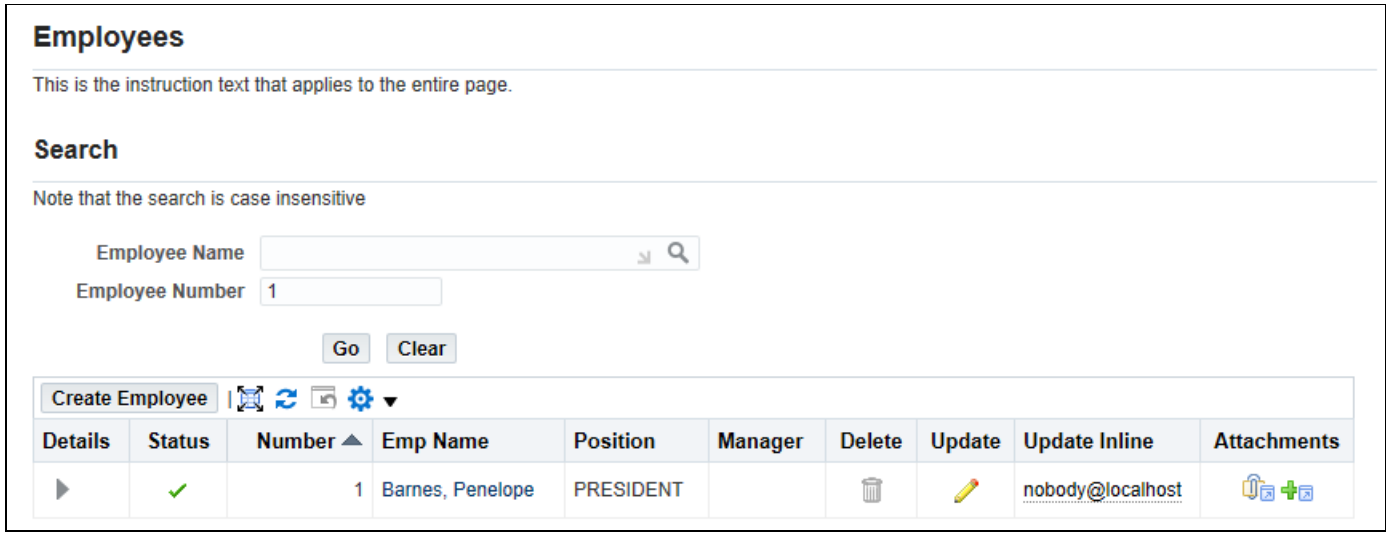


To learn how to implement this Home page, see the OA Framework ToolBox Tutorial (`oracle.apps.fnd.framework.toolbox.tutorial.webui.HomePG` in Tutorial.jpr). Also see Content Containers. Also see the ToolBox Tutorial Home Page lab.

Search and Results Page

To create a basic search page like the following, see the ToolBox Tutorial Search lab. This lab also shows how to add an Advanced Search and User-Personalizable Views. Also see the Search document.

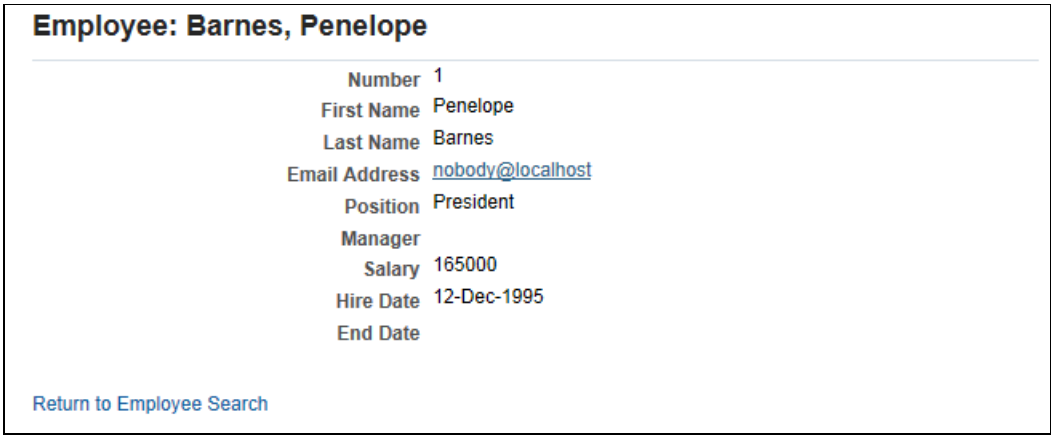
Figure 20: Search Page Configured with just a Simple Search.



Single Column Display-Only Details Page

To create a basic single row display-only details like the following, see the ToolBox Tutorial Drilldown to Details lab.

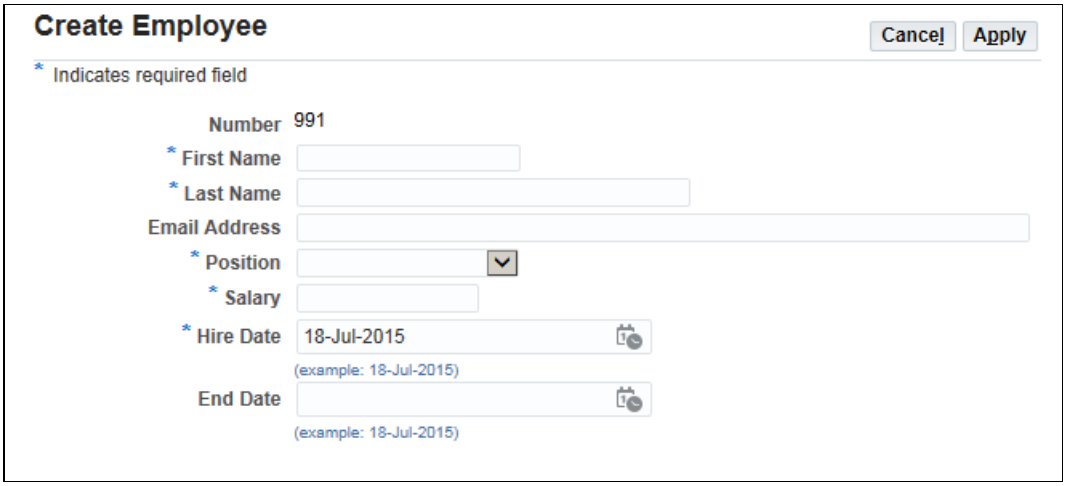
Figure 21: Display-only single row details page.



Single Column Update Page

To create a basic single row update page like the following, see the ToolBox Tutorial Create lab.

Figure 22: Updateable single row Create page.



Master-Detail Page w/ Adjacent Headers

See the tableLayout example shown in Figure 12 above.

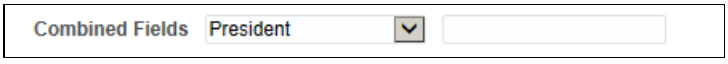
Home Page

See the Page Start and Page End topic above for pointers to examples.

Multiple Items with a Single Prompt

To display multiple items side-by-side with a single prompt as shown in Figure 23, follow these instructions.

Figure 23: Example of a poplist and a text field rendered side-by-side with the same prompt.



Step 1: Create a **messageComponentLayout** region.

Step 2: Add a **messageLayout** region to your **messageComponentLayout** region. Specify the `Prompt` value for the messageLayout region, which serves as the prompt for the related items. (In Figure 23, this value is "Combined Fields").

Step 3: Add a **rowLayout** region to the **messageLayout** region.

Step 4: Add two **cellFormat** regions to the **rowLayout** region.

Step 5: Add your items to the **cellFormat** regions. Do not specify prompts for any of these items, but do set `Additional Text` values for accessibility (per the OA Framework View Coding Standards, all fields must have prompts or additional text values).

Step 6 (optional): If one or more of the items that you added to the **messageLayout** region are required:

- Set the **messageLayout** region's `Required` property to **yes** (you need to set this programmatically). This ensures that the required indicator renders next to the **messageLayout** prompt.

Step 7: Add a controller to your region with the following code to identify the `IDs` for the items that you added in Step 3. This ensures that any Javascript client errors raised for these items display the `Prompt` that you specified in Step 2 (as if the combined fields were a single component).

Note: The list of `IDs` is space-delimited, and is specified as the `LABELED_NODE_ID_ATTR` attribute value. (This cannot be set declaratively).

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{

  super.processRequest(pageContext, webBean);

   // Configure the combined fields with a single prompt so any client side
   // validation errors for the poplist or the field are displayed for the
   // messageLayoutBean prompt as if this were a single component.
   OAMessageLayoutBean msgLayout =
     (OAMessageLayoutBean)webBean.findIndexedChildRecursive("CombinedFieldsLayout");

   // Set the LABELED_NODE_ID_ATTR attribute to a space-delimited list
   // of IDs associated with the messageLayout region.
   msgLayout.setLabeledNodeID("CombinedPoplist CombinedTextField");
}
```

Figure 24: Example of a Javascript client validation error that references the Prompt associated with the messageLayout region holding the related items (see Figure 23 for the UI beneath the error dialog).

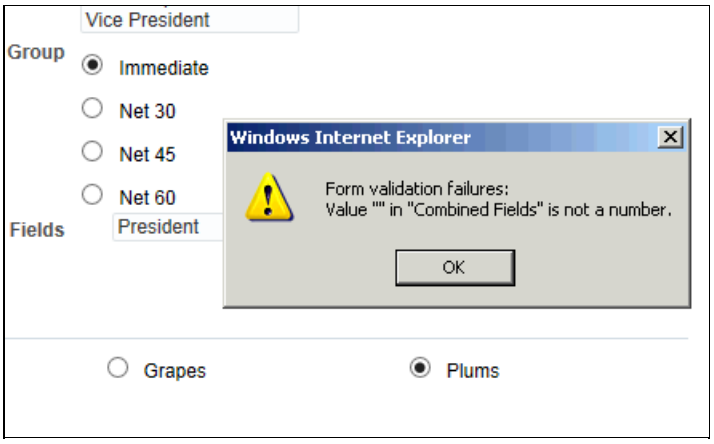


Image and Text in a Table Column

To add an image with text to a table column, create a **flowLayout** region and add your **image**, and **messageStyledText** or **staticStyledText** items.

Tip: If the image displays conditionally, such as when using it to denote comparison values, use a SPEL binding on the image to control its `Rendering` property. See Dynamic User Interfaces for additional information.

Related Information

- OA Framework Developer's Guide
  - Headers and Subheaders
  - Dynamic User Interfaces
  - Content Containers
  - Search
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAHeaderBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAMessageComponentLayoutBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAMessageLayoutBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAStackLayoutBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAFlowLayoutBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OADefaultSingleColumnBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OADefaultDoubleColumnBean`

    - oracle.apps.fnd.framework.webui.beans.layout.OADefaultStackLayoutBean
    - oracle.apps.fnd.framework.webui.beans.layout.OADefaultFormStackLayoutBean
    - oracle.apps.fnd.framework.webui.beans.layout.OATableLayoutBean
    - oracle.apps.fnd.framework.webui.beans.layout.OARowLayoutBean
    - oracle.apps.fnd.framework.webui.beans.layout.OACellFormatBean
- OA Framework ToolBox Tutorial / Sample Library
    - Search lab
    - Drilldown to Details lab
    - Create lab
    - Home Page lab

Page Security

Overview

This document describes how to build applications that grant selected access to individual pages.

Security Example

To put the various aspects of page security in context, this document uses the following example application to illustrate the steps that you need to take in your applications. It is the same example that we used in Chapter 3: Menus and Page Security with one additional special page (the Benefits Registration Page).

Page	Description	Benefits Manager Access?	Employee Access?
Benefits Registration Page	User login page to access the benefits application. This page also allows a user to register himself.	**Yes**	**Yes**
Administer Benefits	View, update, approve and discontinue benefits.	**Yes**	No
Create Benefit	Create a new benefit.	**Yes**	No
My Benefits	View current benefit selections and make new selections as appropriate. • This page can be accessed from the "Workflow Notifications Page". • This page has a "Create" button to launch the "Create Benefit" page.	**Yes**	**Yes**
Update Beneficiaries	Update designated beneficiaries.	**Yes**	**Yes**

Contents

- Step 1: Create the Navigation Menu
- Step 2: Create Permissions
  - Permissions to secure Component Access
    - Case: Page supports auto responsibility setting
    - Case: Shared/reusable page that needs a specific responsibility context
- Step 3: Create Grantees or Roles
- Step 4: Create Permission Sets
- Step 5: Create Grants
- Step 6: Extract Seed Data

Prerequisite Reading

Please read the following before proceeding:

- Chapter 3: Menus and Page Security
- Chapter 4: Tabs / Navigation
- Chapter 4: Component-Level Function Security (Dynamic User Interface)

Step 1: Create Navigation Menu

As described in Chapter 3: Menus and Page Security and Chapter 4: Tabs / Navigation, you should begin with a navigation menu including all the pages that anyone could access from your application's menu. In our example, we would create a function for each of the five pages described above, and add them to tabs, and then the Home Page top-level menu.

Tip: The advantage of having just one navigation menu is that there is just one file to ship, patch and upgrade when the navigation within your application changes and/or you add more pages.

Note that the **Grant Flag** for all of the functions should be unchecked (or set to **N**). One of the primary goals of the Oracle E-Business Suite security model is to separate navigation from security. And as you notice, the above navigation menu just defines the navigation or hierarchy for your application. To ensure the right authorization for your pages, you would then proceed with steps outlined in this document.

Note: Previously, a navigation menu's function **Grant Flags** were checked (or set to **Y**). These functions could then be accessed only by those responsibilities that had access to the navigation menu.

Step 2: Create Permissions

Just like the Navigation functions, **permissions** are FND form functions, but in this context, they are used exclusively for application security. Permissions created for an application fall under the following categories:

- All navigation functions used in your navigation menu -- you can reuse the functions that you created for your navigation menu as permissions.
- Permissions to secure component access -- all your UI components including the page itself can be secured using permissions.

Securing Page Access

You should use the following rules to set permissions to secure your pages:

- **Rule 1**: If a page can be accessed from a menu, then use the navigation function of that page to set permissions. This permission is required to identify if your page is accessible from the menu.

  In our example, we can use the Navigation Functions that we created for the five pages above as permissions. There is no need to create additional permissions.

  Note that you do not need to specify a navigation function or permission for every page in your application. You should specify a navigation function only if is a part of your navigation menu.

- **Rule 2**: If a restricted page can be accessed from another page using a link or a button, then associate a permission with the `Rendered` property of the link or button to hide the link or button from the appropriate users. Create a permission with a name that describes the rule you want to implement.

  In our example, the Create button on the My Benefits page should be displayed only to managers. So, create a permission **MANAGER_RENDERED_ONLY** for this purpose. We will learn about how to associate this permission with the appropriate grants in the subsequent steps. Note that showing a restricted access message on the Create Benefits page when a non-manager selects the Create button is an inappropriate design. You should instead secure the entry points to the restricted page.

- **Rule 3**: If you want to switch or set responsibility seamlessly

  To secure your page access, you should set the function SPEL expression on the `Rendered` property of your **pageLayout** region (see Chapter 4: Component-Level Function Security for information about how to do this). Previously, you would do this using the `Function Name` property on your **pageLayout** region.

  You should do this if your page falls under one of the cases below:

  Note: **GUEST** is a special seeded user in the Oracle E-Business Suite Users form.

  In our example, the Benefits Registration Page is an example of a Guest user page. To create a Guest user page:

  Step 1: Set the Security Mode property of your page to **Self secured**.

  Step 2: Implement the `validateParameters()` method in your controllers to protect the integrity of the URL.

  Tip: Note that the above two steps are required because your page should exist outside a user's session, and is therefore "bookmarkable".

  Step 3: Set a permission on the rendered attribute of your page layout region using a function security SPEL expression.

  ```
  ${oa.FunctionSecurity.BENEFITS_GUEST}Rendered
  ```

  The OA Framework requires no authentication and authorization to run a page that is secured using a GUEST user-granted permission.

  Note: Although a GLOBAL-granted permission would seem logical to use, it actually is not appropriate in this case. All users **except** the GUEST user have access to pages that have a GLOBAL grant. In addition, GLOBAL users must first log in and be authenticated in order to access those globally-granted pages.

  Case 2: Page Supports Automatic Responsibility Setting

  The OA Framework **requires** a responsibility context to be set while running your page, unless it is globally granted. A

responsibility context is established when you launch your page after picking a responsibility from the E-Business Home page.

In order to set a responsibility, you should associate a permission (function name) with the `Rendered` property of your **pageLayout** region using a SPEL expression that allows the OA Framework to establish the responsibility context automatically. The OA Framework will try to establish your responsibility context using the following rules:

- If the permission is associated with just one of your responsibilities, then that is set as your responsibility context.
- If the permission is associated with more than one of your responsibilities, then the OA Framework picks the responsibility that is associated with the same organization as your current organization to display the page. If a match is not found, or if an organization is not available, then the OA Framework chooses your first responsibility associated with the permission.

  You have a responsibility context switcher on the page to switch to another responsibility (associated with the permission) if you like.

- If the permission is not associated with any of your responsibilities, then the OA Framework prohibits you from accessing the page.
- As we stated earlier, if the permission is associated with a GUEST user grant, then the OA Framework requires no responsibility context. In our example, Benefits Registration Page is granted to the GUEST user. So, you can display that page without picking a responsibility.
- If your page has no permission set on its rendered flag, then the OA Framework displays a responsibility context switcher of all your responsibilities and then picks the first responsibility to display the page.

  You can use the responsibility context switcher to switch to another responsibility if you like.

  Tip: If your page is bookmarkable then it most likely falls either under Case 1 or Case 2 or both, unless you want to prompt the user to login and/or choose a responsibility.

Case 3 : Shared/Reusable Page that Needs a Specific Responsibility Context

This is an extension of Case 2. If your page is used in a cross application flow where each application has its own security context, then you should secure your page with a permission. The OA Framework uses this permission to identify your responsibility required for rendering the page, and makes a responsibility switch if necessary.

In our example, the My Benefits page can be launched from the Workflow Notifications Page. Let's assume that the Workflow page needs a Workflow responsibility and the My Benefits page needs a Benefits responsibility. When you navigate to the Benefits page from the Workflow page, you want to switch to the workflow responsibility automatically without prompting the user.

You can do so by associating a permission with the rendered attribute My Benefits page. As we discussed in Case 2 above, the OA Framework uses this permission to set or to switch to the Benefits responsibility automatically.

You should also handle the case where a user can revisit the Workflow page from the Benefits page. Since the workflow page needs a Workflow responsibility, you should set a permission on its rendered attribute as well. This permission will then be used to switch to the Workflow responsibility automatically.

Step 3: Create Grantees or Roles

Previously, a responsibility was the primary mechanism for grouping users into role-based sets. You would then assign menus to responsibilities, and create security rules by excluding individual menu functions from your responsibility. At runtime, the current responsibility, organization and security group together comprised the security context.

The concept of responsibility has been expanded to a more generic **role**. Users can belong to one or more roles. All users assigned to a particular responsibility are also assigned to a corresponding **role**. Security rules are based on **permission grants** instead of function exclusion rules. At runtime, these grants are evaluated for the current security context, which now includes roles (also known as a "**grantee**") in addition to responsibility, organization and security group.

A grantee can either be a **user** (FND_USER)**,** or a **user group**(also known as **role**), or **"global"**. **User** identities are created in FND_USERS, and should map one-to-one with individual humans or systems. Users can belong to **groups** or **roles** that are formed by grouping organizational or position relationships modeled in products such as Human Resources. Roles are defined in WF_ROLES. Although its membership is not explicitly populated, there is a **Global** group which includes "everyone". Ideally, users would belong to predefined Workflow roles, so you don't have to group them again.

Note: The GLOBAL role actually includes everyone except the GUEST user.

You need three user roles for our example above: one that groups all managers into a manager role, one that represents the GUEST user and a third that groups all employees. Since all employees includes everyone, you can use a **GLOBAL** role for this purpose.

Alternately, you can create a responsibility that is assigned to all managers, and use that for your grants setup.

Note: When using a Proxy User, there is a change in the UserInfo Named Child.
For the proxy user we add a StackLayout region as a userInfo named child of the PageLayout.
If the product team has added any other region as an userInfo named child, then we add that region under the StackLayout region.
Because of this change, for a user with proxy privilege, the call to the api getUserInfo() on the OAPageLayoutBean will return a OAStackLayoutBean instead of the region which the Product team added.
Hence, product teams should modify their code to reflect this change.
i.e., for a user with proxy privileges, the getUserInfo() api will return a OAStackLayoutBean and the userInfo region added by the product team will be an indexed child of the OAStackLayoutBean.

Step 4: Create Permission Sets

Permission Sets are implemented as menus, but they are exist solely to group a flat list of permissions into sets for easy access granting.

Ideally, you should group permissions that are required by a role into one or more permission sets.

You need three permission sets for our example:

- A Manager permission set for all the tasks to which only managers should have access. This includes:
  - The navigation functions Administer Benefits, and Create Benefit.
  - The **MANAGER_RENDERED_ONLY** permission that we created in Step 4 associated with the Create Button.
- A GLOBAL permission set with permissions that are accessible by everyone. This includes:
  - The navigation functions Benefits Registration Page, My Benefits and Update Beneficiaries.
- A GUEST permission set for the publically accessible page

Here are the step by step instructions to create the manager permission set:

Step 1: Start Oracle E-Business Suite Release 12 in your development environment and log in as a user that has the System Administrator or Application Developer responsibility.

Step 2: Select either the System Administrator or the Application Developer responsibility.

Step 3: Select the Navigator to choose the Menus form from the **Application > Menus** menu (or, it might also be in the **Top 10** list). As described above, permission sets are flat menus.

Step 4: To create a new permission set, simply start entering values in the blank fields. Set values for the following properties. Note that you can accept all the other default values.

- `Menu` - this is the unique developer key for your permission set. In our example, we'll call this the BENEFITS_MANAGER_PS (the product short code followed by a descriptive name for the permission set).
- `User Menu Name` - this is the unique, user-friendly version of the permission set that displays when administering. In our example, we'll call this Benefits Manager Permission Set.
- `Description` - provides a short description of the permission set.
- `Menu Type` - You can choose any menu type for permission sets, but for convenience and easy access, we recommend that you use the SECURITY menu type.

Step 5: Add all permissions for this permission set as individual menu entries. Note that since the same form is used for creating a navigation menu (as we saw in Chapter 4: Tabs/Navigation) and a permission set, there are some UI- or navigation-only properties for menus that are not applicable to permission sets. This includes `sequence` and `prompt`. You can enter any value for these two properties. The properties that should have valid values are:

- `Submenu` or `Function` - the permission name to be used in the permission set. In our example, **MANAGER_RENDERED_ONLY** is one of the permissions that you should specify here.
- `Description` - a brief description of the permission.
- `Grant Flag` - as stated in the Navigation Menu section, this should be unchecked. When unchecked, the OA Framework will evaluate your permission set based on its grants and your runtime security context. We will create the appropriate Grants in the next step.

Step 6 : To save your work, select **File > Save** from the menu.

Step 5: Create Grants

A **Grant** defines security rules that allows only certain users of your system access to specific functions or pages within your

application. A **grant** gives a **grantee** access to the permission sets described above. In simple terms, grants link your grantees to your permission sets.

You need three grants for the example above:

- A Manager Grant to associate the manager permission set with the manager role.
- An Employee Grant that is associated with your Global permission set with a global grantee.
- A GUEST Grant that is associated with your GUEST permission set with a GUEST grantee.

In addition to specifying a grantee, you could also restrict your grant further with additional security context. This includes the current user's responsibility, organization and security group. So, for example, to restrict the manager grant to a specific organization, you can associate an organization context with the grant.

Note that grants, just like responsibilities, are mostly created at the customer site. You create the permission sets, and the customer then creates the WF roles and creates grants to associate these roles with your permission sets.

Instead of granting the manager permission set to the manager role, you can grant it to a global grantee. You can then restrict it to managers alone by associating a security context with the responsibility to which only managers have access. However note that the OA Framework recommends the use of role based grants instead of responsibilities.

At runtime, a user is granted access to a page if the permission associated with the page is granted access to the current user's security context. The user's security context as described above includes the user's role, responsibility, organization and security group.

To create the Manager grant, we:

Step 1: Login to your development database and choose the Functional Administrator responsibility. This takes you to the Applications Administration Home Page.

Step 2 : Navigate to the **Grants** sub tab under **Security** tab. Select the **Create Function Grant** button. Note that since the grant that we are creating is associated with a permission, it is also called a Functional Grant. You can use the page navigation buttons to complete the steps below.

Step 3 : Choose the grantee for your grant. You have the following options:

- You should choose the All Users option to create a global grant that is accessible by everyone.

  Tip: You should choose this option for the Employee Grant.

- You should choose the Group of Users option to create a grant that is associated with a specific role.

  In our example, you should choose this option, and pick the manager role that you created from the LOV.

- You should choose the Single User option to create a grant that is associated with just one user.

  Tip: You should choose this option for the GUEST Grant.

Step 4: Associate your grantee with the permission set by picking it from the LOV. In our example, you should pick the Benefits Manager Permission Set that you just created.

Step 5: Pick any additional security context to associate with this grant. In our example, as we stated above, to restrict the manager grant to a specific organization, pick the organization from the LOV.

Step 6: Pick the start date to activate your grant and optionally specify an end date.

Step 7: Select **Finish** to create your grant.

The Grants model is a very versatile model for modeling any security requirement of your application. The above steps to create your security rules by using roles, permission sets and grants is just one aspect of what it offers. You can find more information on its full list of features from the *Oracle E-Business Suite Security Guide*.

Step 6: Extract Security Seed Data

To deploy your security data to a different database (or to ship it if you are an Oracle E-Business Suite internal developer) you must extract your seed data using the generic Oracle E-Business Suite loader FNDLOAD. You can find more information on FNDLOAD from the *Oracle E-Business Suite Security Guide*.

The syntax you use for FNDLOAD is

```
FNDLOAD username 0 Y mode configfile datafile [ entity [ param ... ] ]
ORACLE password:
```

where
`username` is APPS username
`mode` is either `UPLOAD` or `DOWNLOAD`
`configfile` is the configuration file
`datafile` is the data file
`entity` is an entity name, or `-` to specify all values in an upload
`param` is a `NAME=VALUE` string used for parameter substitution

The following table describes which configuration file to use for the entities you want to extract. The configuration files are published in `$FND_TOP/patch/115/import`.

Entity Name	Description	Configuration File
FND_RESPONSIBILITY	To extract the responsibilities that you created with your navigation menu.	afscursp.lct
MENU	Extracting a menu will also extract any submenus or functions associated with it. In our example, use this to extract the navigation menu.	afsload.lct
GRANT	Extracting the grant will also extract its permission set and its permissions. In our example, use this to extract the manager and the employee grants.	afsload.lct.

For example, to extract Grants seed data, you would specify:

```
FNDLOAD <username> 0 Y DOWNLOAD
  $FND_TOP/patch/115/import/afsload.lct somefile.ldt
  GRANT GNT_MENU_NAME=the_menu_under_which_grants_exist
```

Page Stamps

Overview

As described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Page Stamps, page stamps let you display view-only, contextual information in a consistent fashion. Specifically, page stamps fall into the following categories:

- User login and connection information (for example, Logged In As **JFROST**)
- Status or task-related information (for example, **22-Oct-2001 10:15 AM** [*current date and time*], Status **Approved**, Sales Target **$65,000,000** and so on)

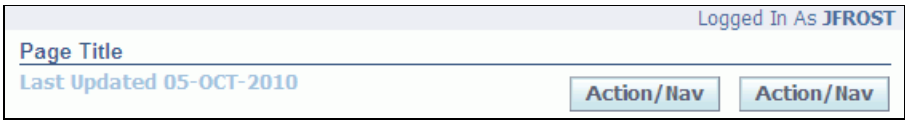
Page stamps are placed on the page according to the type of stamp and relative importance to the user's task. Possible locations (fully described in the UI guideline) include:

- User Info ("Log in Stamp")
- Footnote Stamp
- Primary Page Stamp
- Section Stamp

User Info ("Log in Stamp")

If present, the user/connection stamp displays in the upper right-hand corner (in an American UI) of your page as shown in Figure 1:

Figure 1: Example of a Log in stamp.



Implementation

The user info login stamp is a special, named child of the `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`. The user info stamp must be added programmatically as shown.

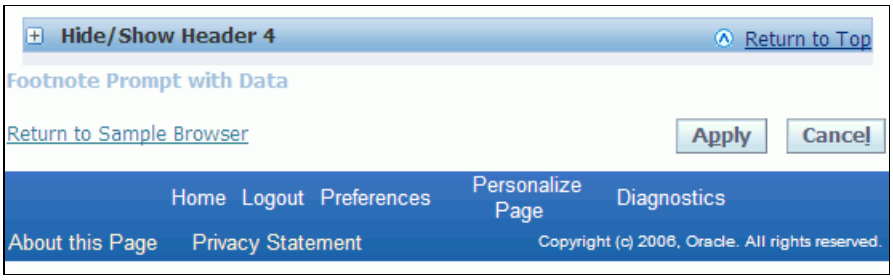
```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   super.processRequest(pageContext, webBean);

   // Create a flow layout region to hold the prompt and data components.

   OAFlowLayoutBean userInfo = (OAFlowLayoutBean)createWebBean(pageContext,
                                       OAWebBeanConstants.FLOW_LAYOUT_BEAN,
                                       null, "userInfo");
   OAStyledTextBean infoPrompt = (OAStyledTextBean)createWebBean(pageContext,
                                       OAWebBeanConstants.STYLED_TEXT_BEAN,
                                       null, "infoPrompt");
   OAStyledTextBean infoText = (OAStyledTextBean)createWebBean(pageContext,
                                      OAWebBeanConstants.STYLED_TEXT_BEAN,
                                      null, "infoText");

   userInfo.addIndexedChild(infoPrompt);
   userInfo.addIndexedChild(infoText);

   // Set the content for the prompt and the user name based on the current
   // user. Note that the prompt should be sourced from Message Dictionary and
   // not hard-coded as shown. Also note the inclusion of white space after
   // The "User Name" text to ensure that the two Strings don't render right
   // next to each other (the flowLayout region is whitespace sensitive).

   infoPrompt.setText("User Name ");
   infoText.setText(pageContext.getUserName());

   // Set the following styles to achieve the required look.
   infoPrompt.setCSSClass("OraPageStampLabel");
   infoText.setCSSClass("OraPageStampText");

   // Set the user info component on the page layout bean.
   OAPageLayoutBean pageLayout = (OAPageLayoutBean)webBean;
   pageLayout.setUserInfo(userInfo);
```

}

Footnote Stamp

If present, a footnote renders at the bottom of your page above the page contents bottom line as shown in Figure 2.

Figure 2: Example of a footnote.



Declarative Implementation

The footer page stamp is a special, named child of the `OAPageLayoutBean`. To add it to your page:

Step 1: Select the **pageLayout** region in the Structure pane, right-click and select **New > footnote**. JDeveloper will create a **footnote** node including an item.

Note: The UI Guidelines allow only one footnote; you can't add a region of multiple footnotes.

Step 2: Name your item in accordance with the OA Framework Package / File / Directory standards.

Step 3: Set the `Item Style` to **formattedText**. Set the `Text` property to the content you want to display in the footer stamp. For example: **Footnote Prompt with Data**. Note the addition of the bold tag around the data portion.

Step 4: Set the `CSS Class` to **OraPageStampText**.

The resulting footnote renders as shown in Figure 2 above.

Runtime Control

To create and set the footer stamp programmatically, follow the User Info procedure described above. The `OAPageLayoutBean` accessors for this named child are `get/setFootnote()`.

Primary Page Stamp

If present, a primary page stamp renders at the top of your page immediately below the page title as shown in Figure 1 (see the "Last Updated" stamp).

Note: In addition to page stamps, this component should be used to place any content that should be in parallel with page-level action/navigation buttons. See Figure 3 and the Contextual Information document for examples of this.

Figure 3: Example of a page stamp component used to render instruction text and a required field indicator.



Declarative Implementation

The primary page stamp is a special, named child of the `OAPageLayoutBean`. To add it to your page:

Step 1: Step 1: Select the **pageLayout** region in the Structure pane, right-click and select **New > pageStatus**. JDeveloper will create a **pageStatus** node including a **flowLayout** region.

Step 2: Name your region in accordance with the OA Framework Package / File / Directory standards.

Step 3: Set the `Region Style` as appropriate to achieve the desired layout (see Page Layout (How to Place Content) if you need help with standard layouts), and add the region and item(s) you want to display. For example, to achieve the layout shown in Figure

3, you simply need to add a **staticStyledText** item to the page status **flowLayout** followed by a region that extends `/oracle/apps/fnd/framework/webui/OAReqFieldDescRG` (since `OAReqFieldDescRG` is a **tableLayout** region, remember to set your region's `Width` to **100%** so it aligns properly to the start).

Runtime Control

To create and set the page status stamp programmatically, follow the User Info procedure described above. The `OAPageLayoutBean` accessors for this named child are `get/setPageStatus()`.

Section Stamp

Section stamps are merely items that render immediately beneath a header region (you may include up to 3 stamps beneath each header). To create a section stamp, add an item to your header and configure it as shown for a footnote.

Personalization Considerations

- None

Known Issues

- None

Related Information

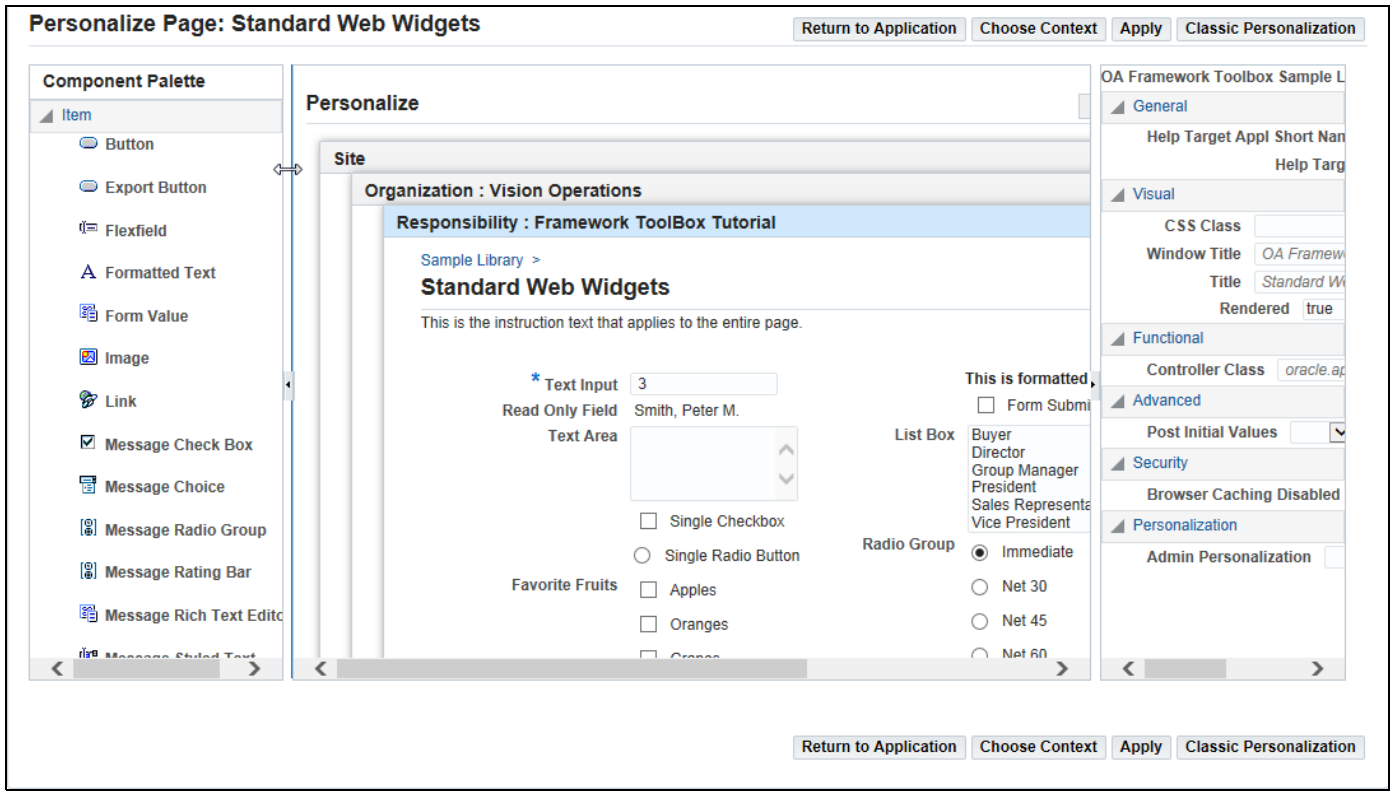
- BLAF UI Guidelines
  - Page Stamps
- OA Framework Developer's Guide
  - Contextual information
  - Page Layout (How to Place Content)
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`
- OA Framework ToolBox Tutorial / Sample Library
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.BasicStructPG`

Panel Splitter

Overview

Beginning in Release 12.2.6, you can create a component in OA Framework called a panel splitter region. This component enables you to design Oracle E-Business Suite pages in such a way that a single region may be split into two separate resizable and collapsible regions, called panels. You may add a panel splitter region to any container region as a child. Beneath the panel splitter region, you create two named children called first panel and second panel. As described in the Alta UI Guideline: Panel Splitter specification, the position of the splitter bar decides the area occupied by the two panels. You may adjust the splitter bar using the drag event of a mouse or by touch, on a touch device. You may collapse any one of the two panels, allowing the other panel to occupy the entire space of the region. If you need to further divide a page, you may create a second level panel splitter region, such that you have a panel splitter inside a panel splitter. You may divide a page by nesting panel splitters to any level to structure a page.

Figure 1: Example of a horizontal panel splitter within a horizontal panel splitter, as implemented in the Personalization Workbench UI.



You may only move the splitter bar inside the main panel splitter boundary. You may select the arrow icon in the splitter bar to collapse the panel that is pointed to by the arrow. You may define the panel that collapses at design time.

Contents

- Declarative Implementation
- Runtime Control
- Accessibility Support
- Personalization Considerations
- Known Issues
- Related Information

Declarative Implementation

Use the Oracle JDeveloper OA Extension Property Inspector to add a panel splitter component to any container region.

Step 1: In your pageLayout region, select the region to which you want to add a panel splitter component. Select **New** > **region** from the context menu.

Step 2: Set the `Region Style` property for this new item to **panelSplitter**.

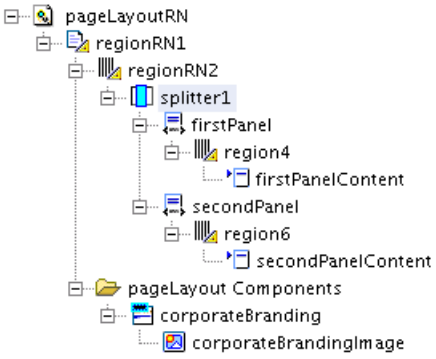
Step 3: Use the Property Inspector to set the properties on the **panelSplitter** region, specifically (where * indicates a required property):

- `ID*` - Specify an identifier that uniquely identifies the region in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Attribute Set` - specify an attribute set if you want to maintain a standard look and feel for your item.
- `Orientation*` - specify the orientation of the split panels. The orientation specifies whether the panels are stacked horizontally or vertically.The default is **horizontal**.
- `Splitter Bar Offset Position*` - specify the position of the splitter bar with respect to the first panel, as a percentage. The default is **50**.
- `Collapsed` - specify the initial state of the collapsible panel. The default is **False**.
- `Collapsible Panel*` - specify the panel that the user can collapse or expand. Valid values are **first** or **second**. The default is **first**.
- `Rendered` - specify whether to render the panel splitter region. The default is **True**.

Step 4: Under the named child **firstPanel**, create your first panel content.

Step 5: Under the named child **secondPanel**, create your second panel content.

Figure 2: Example of a panel splitter region structure as displayed in JDeveloper.



Runtime Control

At runtime, OA Framework instantiates an `oracle.apps.fnd.framework.webui.beans.layout.OAPanelSplitterBean.`

The following sample code demonstrates how to create a panel splitter web bean programmatically:

```
OAWebBeanFactory fac = pageContext.getWebBeanFactory();
 OAPanelSplitterBean panelSplit =
  (OAPanelSplitterBean)fac.createWebBean(pageContext,
  PANEL_SPLITTER_BEAN, null, "Split1");
```

You may also set panel splitter properties as follows:

```
panelSplit.setAttributeValue(POSITION_ATTR, "30%");
//Defines the position of splitter in percentage with respenct to the first panel

panelSplit.setAttributeValue(ORIENTATION_ATTR, ("horizontal"));
//Defines the orientation on which container needs to be divided, Horizontal/Vertical

panelSplit.setCollapsiblePanel("second");
//Panel that supports expand and collapse

panelSplit.setAttributeValue(ID_ATTR, "mainPanel");
//ID of splitter on page
```

To add panel content to the panel splitter region:

```
panelSplit.setFirstPanelContent(contentLT);
panelSplit.setSecondPanelContent(contentRT);
```

To add the panel splitter to an existing region:

```
stackBean.addIndexedChild(panelSplit);
```

Accessibility Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may

still use the panel splitter component as follows.

Keyboard Support

Use the following keys to interact with the panel splitter region:

- **[Tab]** - navigates focus onto the collapsible/expandable icon.
- **[Left Arrow]** or **[Down Arrow]** - with focus on the collapsible/expandable icon, moves the position of the splitter bar left or down.
- **[Right Arrow]** or **[Up Arrow]** - with focus on the collapsible/expandable icon, moves the position of the splitter bar right or up.
- **[Enter]** - with focus on the collapsible/expandable icon, expands or collapses the collapsible panel.

Personalization Considerations

- See a summary of Panel Splitter personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key issues with suggested workarounds if available.

Related Information

- UI Guideline(s)
  - Alta UI Guideline: Panel Splitter
- OA Component Reference
  - For a description of all the possible properties that you can set for the **panelSplitter** region type, see the OA Component Reference in the JDeveloper online Help. To access, select **Help > Help Topics** from the main menu. In the **Help Navigator** window, expand the **OA Component Reference** and **OA Region Styles** nodes.
- Javadoc
  - oracle.apps.fnd.framework.webui.beans.layout.OAPanelSplitterBean
- OA Framework ToolBox Tutorial / Sample Library
  - None

Popups

Overview

Note: This feature is supported by Oracle E-Business Suite Release 12.1.2 and higher, and by a subset of the browsers certified with Release 12. For Microsoft Internet Explorer (IE), this feature is only supported in version 7.0 and higher.

A popup component is a small window that allows a user to display and update contextual information. A popup window can render any OA Framework region. There are two types of popups:

- **Embedded popup** - The popup is embedded within the base page web bean hierarchy. It is an extension of the base page, sharing the same page state. Its contents are pre-fetched as part of the base page request.
- **Parameterized popup** - The popup is a separate standalone region that is not embedded within the base page. Its contents are fetched and rendered on demand.

Pop-ups are available on the following OA Framework components:

- Text (messageStyledText)
- Image
- Link
- Button

Upon hover-over or selection on any of these components, the popup renders, either displaying read-only contextual information or allowing inline entry of transactional data.

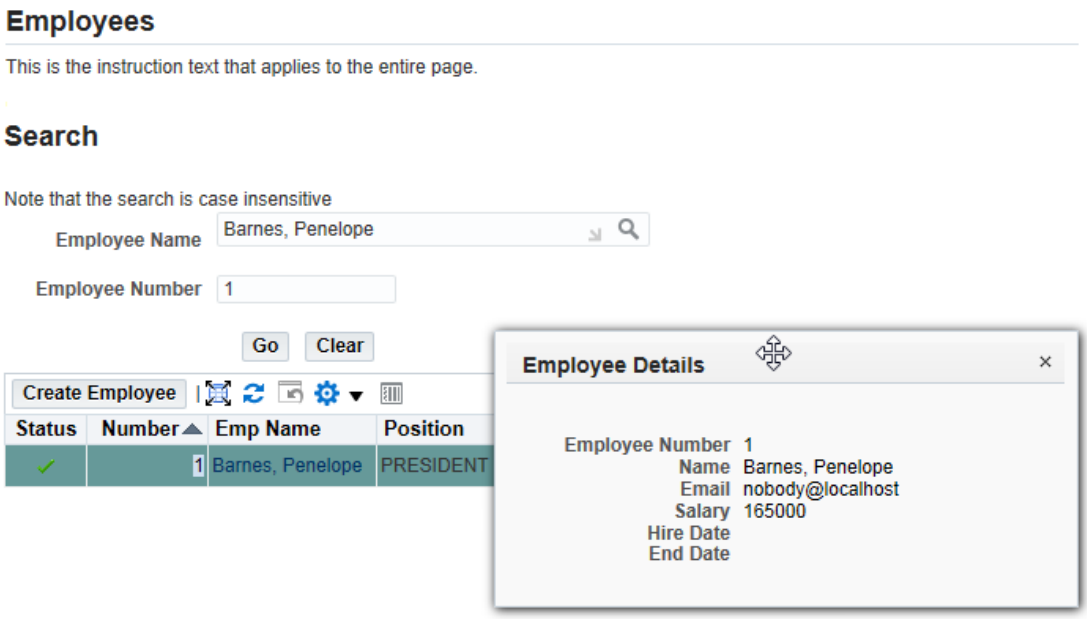
The following table summarizes guidelines for the type of interaction and visual indicator that should appear when you add a popup to any of the above components:

Type of Pop-up	Base Component	Interaction	Indicator
Information Pop-up (Read-only)	Link, Text	Hover-Over	Dotted Underline
		Click	Normal Link
	Button	Hover-Over	Not Recommended*
		Click	Not Required
	Image (Icon)	Hover-Over	Visual Indicator**
		Click	Not Required
Dialog with Transaction (Editable)	Link, Text	Hover-Over	Dotted Underline
		Click	Normal Link
	Button	Hover-Over	Not Recommended*
		Click	Not Required
	Image (Icon)	Hover-Over	Visual Indicator**
		Click	Not Required

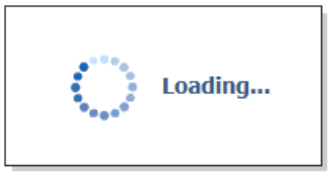
* There should not be any hover-over interaction for a button since the purpose of a button is for a user to click on it to perform an action. If you have a need for a popup to appear over button, then you should change the button to a link based on the intended task.

**** If you add a popup to an image component, you must use an image that includes a visual popup indicator.

Figure 1: An example of a popup window that appears when hovering over the Emp Name column. Note that as of Release 12.2.5, when you select the popup title header, a "move" cursor appears, allowing you to move the popup window to a new position.



Note: As of Release 12.2, an indicator appears when you trigger a parameterized popup, signifying background processing as the popup loads:



Contents

This document contains the following topics:

- Overview
  - User Interactions
    - Accessibility Considerations
- Declarative Implementation
- Runtime Control
- Usage Restrictions
- Personalization Considerations
- Known Issues
- Related Information

User Interactions

When you enable a popup, you can configure the parent messageStyledText, image, link, or button component to trigger the popup display from a mouse click, a keyboard selection event or a mouse hover-over event. The popup renders below and to the right of the parent component unless there is insufficient space to display the popup in that area. If that is the case, the popup renders to the left or above the parent component, as necessary.

The popup window includes a title header bar and a **Close** window icon. You may also close the po-up either by selecting the **Close** icon or by any action within the popup that dismisses the popup. For a read-only popup, starting in Release 12.2.5, you may select on any area of the browser window outside of the popup to automatically close the popup.

If a user modifies the popup's content and then selects the **Close** icon, the following warning appears, "The changes you have made in this popup have not been saved. If you continue, the changes will be discarded. Do you wish to continue?" If the user selects **Yes**, any changes made in the popup are discarded, and the popup closes. If the user selects **No**, the popup remains open and retains any previous changes.

You can handle user interactions for embedded and parameterized popups as follows:

- **All-Submit** - Any "Submit" button within the embedded or parameterized popup would submit the popup data and the base page data together. The popup closes and the base page submits after this event.

- **Self-PPR** - Define partial page refresh events on the popup to refresh the popup window alone. The popup remains displayed after a self-PPR event.
- **GET** - All GET requests triggered within the popup (links, buttons) redirects the user to the target page within the popup.

Note: You can use a form submit event within a parameterized popup to refresh items on the base page. On the form submit, capture the popup's `submitButton` event by calling the following from the base page's processFormRequest method:

```
pageContext.getParameter("popupEvent")
```

Then using the captured `submitButton` event, re-execute the query for the base page to refresh its data. You can use this event handling capability to write other code specific for the parameterized popup.

When a parameterized popup is opened, the user can still interact with the base page. In Release 12.2.9, with the popup in popup feature, it is possible to open a second-level parameter popup on top of the first level parameter popup. When the second-level popup is opened, the user can only interact with the first-level parameter popup, and will not be able to interact with the base page. Only after closing the second-level popup will the user again be able to access the base page.

Note: Refer to Caveats on the Apple iPad in the Mobile Applications topic for more information about popup behavior on the Apple iPad platform.

Accessibility Considerations

In standard or screen reader accessibility mode, a user may interact with popups as follows:

- **Pop-up invocation** - A user can tab to a popup-enabled item on a base page and invoke the popup by pressing the [Enter] key. A popup can only be invoked upon hover over if the Accessibility mode is set to None.

  Note: In screen reader accessibility mode, a user can not invoke a popup by hovering over a popup-enabled Link, Button, or Image (Icon) item. However, a user can invoke a popup by hovering over a popup-enabled messageStyledText item in any accessibility mode

- **Navigation** - When a popup displays, the initial focus is on the **Close** icon. if there is no **Close** icon, the initial focus is on the first focusable item within the popup. A user can navigate to items within the popup by using the **[Tab]** or **[Shift]**+**[Tab]** keys. Focus is held within the popup.

  For modal popups, focus remains within the popup until the user cancels or submits the popup.

  For non-modal popups, the user may select the **F6** key to toggle focus between the base page and the popup.

- **Close** - A user can close a popup by pressing the [Enter] key when the focus is on the **Close** icon or by pressing the [Esc] key when the focus is within the popup. Once the popup is closed, the focus returns to the base page's popup-enabled item.
- **Other** - Based on accessibility guidelines, the OA Framework development team advises against enabling messageStyledText items as popups. If this is unavoidable, then for accessibility modes, set the `Destination URI` property on the messageStyledText item to **#** so that it renders as a link, making it focusable.

Declarative Implementation

To enable a popup on a component:

Step 1: In Oracle JDeveloper OA Extension, create a standalone region containing the items that you want to render within the popup.

Step 2: In the OA Extension structure pane, for the page that should display the popup, select the region that contains the item for which you want the popup to appear.

Step 3: Create a new region and set its `Style` property to **popUp**.

Step 4: Set the following properties on this new popUp region:

Property	Type	Description
`ID`	String	Specify an identifier that uniquely identifies the popup in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
`Title`	String	Specify the text to appear as the title for the popup window. **Note**: As per accessibility guidelines, you must specify a title for a popup window, especially if it is read-only. If a title is not specified, the title bar containing the **Close** window (**X**) icon does not render.
`Region`	String	Specify the fully-qualified path of the standalone region to be rendered as a popup.
`Type`	String	Specify a popup type of either **embedded** or **parameterized**. The default is **embedded**.
`Width`	Number	Specify the width of the popup window (in pixels).

Height	Number	Specify the height of the popup window (in pixels). **Note:** Set the height to a minimum value of 50 to ensure the popup window accommodates its popup content. If the height of the popup content exceeds the height of the popup window, a vertical scroll bar renders. Increase the height of the popup window to eliminate the rendering of a vertical scroll bar.
Enable Auto Resize	Boolean	As of Release 12.2.5, specify whether the popup window should automatically resize to fit the current popup content. If this property is set to **True**, the Width and Height properties are ignored.
Parameters	String	Specify any base page parameters to pass to the popup region, if the popup-type is parameterized.
Modal Enabled	Boolean	Specify whether the popup should be a modal popup. If set to True, when the modal popup renders, users will not be able to interact with the base page.
Read Only	Boolean	Specify whether the popup should be read-only.
Show Notch	Boolean	As of Release 12.2.5, specify whether the popup should display a notch above its window (as in a contact card), pointing to the item that launches the popup.

Note: If you add an embedded popup under a table region, then you should not set the AM Definition property on the standalone popup region.

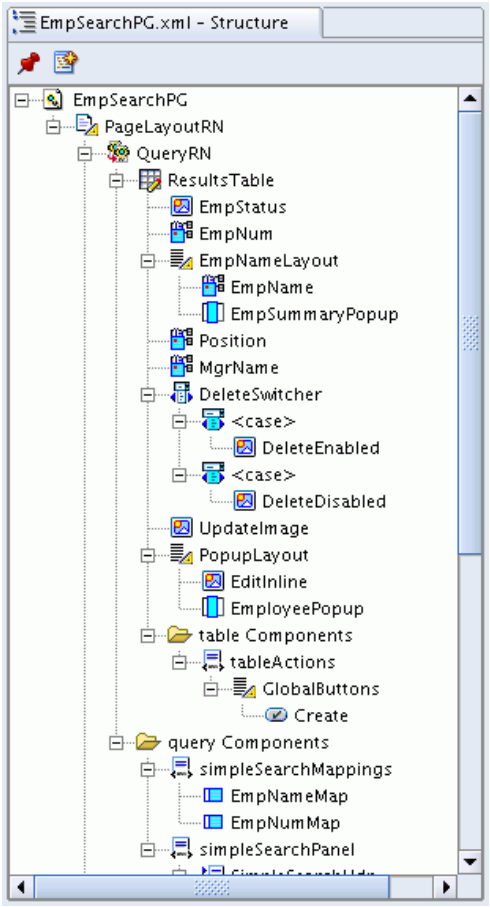
Step 5: In the OA Extension structure pane, select the item (either messageStyledText, image, link, or button) on which the popup should be shown and set the following properties:

Property	Type	Description
PopupId	String	Specify the identifier of the popUp item to be rendered for this component.
PopupRenderEvent	String	Specify the type of event that will launch the popup. A valid value is either **onClick** or **onHover**. The default is **onClick**.
PopupEnabled	boolean	Specify whether to enable or disable the popup. A value of **true** enables the popup. The default is **false**.
ReadOnly	boolean	Specify whether the popup is read only. If it is read-only, when the mouse no longer hovers over the popup, the popup closes. A value of **true** sets the popup as read-only. The default is **false**. **Note:** If a user's Accessibility mode is set to Standard Accessibility, a title bar is displayed in the popup, allowing the user to select the **Close** icon in the title bar to close the popup. The popup does not close automatically when the mouse no longer hovers over the popup.

By setting these properties, you add the popup as an indexed child under the same parent layout component as the component that invokes the popup.

Note: If you add a popup under a table region, be sure to set the Visual Prompt property on the parent item (column) on which the popup should be shown, otherwise the column will render with a blank header. For example, in the popup shown in Figure 1, to ensure the first column of the results table displays "Employee Number", set the Visual Prompt property on PopUpRN, as shown in Figure 3, to "Employee Number".

Figure 3: An example of the page structure for the popup window shown in Figure 1.



Runtime Control

Parameterized Pop-ups

To programmatically add a parameterized popup to a component:

Step 1: Create an OAPopupBean. For example:

```
OAPopupBean popupBean1 =(OAPopupBean)createWebBean(pageContext,POPUP_BEAN,null,"myPopup1");
//Set the following properties on the popup
popupBean1.setID("myPopup1");
popupBean1.setUINodeName("myPopup1");
popupBean1.setRegion(popupRegion1);
popupBean1.setHeight("130");
popupBean1.setWidth("320");
popupBean1.setTitle("Test");
popupBean1.setParameters("deptNo={@Deptno}");
popupBean1.setType(PARAMETERIZED_POPUP);
```

Step 2: For the item (messageStyledText, image, link, or button) on which you want to enable the popup, set the following properties, as shown in this example:

```
OAHeaderBean headerBean = (OAHeaderBean)webBean.findChildRecursive("headerRN");
OAMessageStyledTextBean deptBean = (OAMessageStyledTextBean)webBean.findChildRecursive("deptNo");
deptBean.setPopupEnabled(true);
deptBean.setPopupRenderEvent("onHover");
deptBean.setPopupID("myPopup1");
```

Step 3: Add the popup as an indexed child of the region that contains the item on which the popup is enabled.

Note: If you are enabling a popup on an item within a classic table, advanced table or HGrid, then you must add the popup and the item on which it is enabled to a layout and add the layout as the indexed child of the table.

```
headerBean.addIndexedChild(popupBean1);
```

Embedded Pop-ups

To programmatically add an embedded popup to a component:

Step 1: Create an OAPopupBean. For example:

```
OAPopupBean popupBean =(OAPopupBean)createWebBean(pageContext,POPUP_BEAN,null,"myPopup");
//Set the following properties on the popup:
popupBean.setID("myPopup");
popupBean.setUINodeName("myPopup");
String popupRegion= "/oracle/apps/fnd/framework/toolbox/labsolutions/webui/TestEmpDetailsRN"
popupBean.setRegion(popupRegion);
popupBean.setHeight("130");
popupBean.setWidth("320");
popupBean.setTitle("Test");
popupBean.setType(EMBEDDED_POPUP);
```

Step 2: Select the item (messageStyledText, image, link, or button) on which you want to enable the popup, and set the following properties, as shown in this example:

```
OATableBean tableBean = (OATableBean)webBean.findChildRecursive("ResultsTable");
OAImageBean image = (OAImageBean)webBean.findChildRecursive("UpdateImage");
image.setPopupEnabled(true);
image.setPopupRenderEvent("onClick");
image.setPopupID("myPopup");
```

Step 3: Add the popup as an indexed child of the region that contains the item on which the popup is enabled.

Note: If you are enabling a popup on an item within a classic table, advanced table or HGrid, then you must add the popup and the item on which it is enabled to a layout and add the layout as the indexed child of the table.

In this example, the popup is enabled on an image within a table, hence the popup is added as a second level indexed child of the table:

```
OAStackLayoutBean stackLayoutBean = new OAStackLayoutBean();
stackLayoutBean.addIndexedChild(popupBean);
tableBean.addIndexedChild(stackLayoutBean);
```

Auto-Positioning a Pop-up

As of Release 12.2.4, you may use the `OAPopupBean.setAutoPosition("<position>")` API to programmatically define a fixed position for a popup, where `<position>` may be `center`, `top-left`, `top-right`, `bottom-left`, or `bottom-right`. You may retrieve this default position in your code by using `OAPopupBean.POPUP_POSITION_CENTRE`. Alternatively, you can set a fixed absolute position for the popup by using the `OAPopupBean.setPopupPosition(x,y)` API.

Adding a Notch to a Popup

As of Release 12.2.5, to add a notch to a popup window that points to the item that launches the popup, use the following example code:

```
OAPopupBean popupBean=(OAPopupBean)webBean.findChildRecursive("popupId");
popupBean.setNotchShown(true);
```

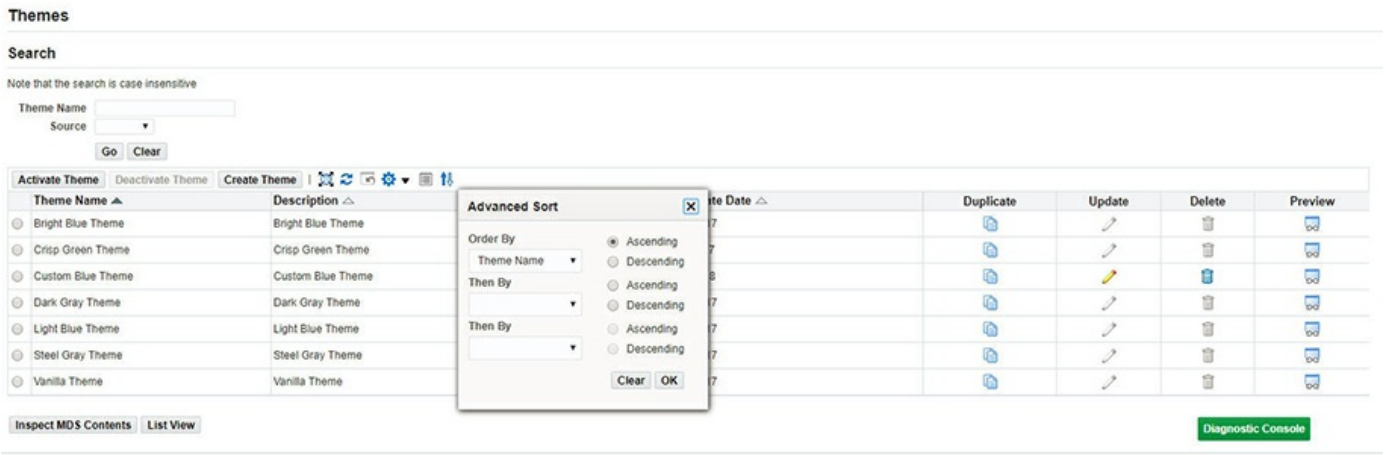
Using an API to open a Popup

As of Release 12.2.7, you can open a popup window with an API.

1. There is an existing popup region with popup ID "myPopup".
2. For the item (messageStyledText, image, link, or button) on which you want to enable the popup:
    a. Set "Popup Render Event" and "Action Type".
    b. Ensure "Popup ID" is null and "Popup Enabled" is false.
3. In processFormRequest method, call pageContext.openPopup("myPopup") or pageContext.openPopup(OAPopupBean).

Using Table Sort Popup

As of Release 12.2.9, a new icon in the table control bar opens a popup that allows users to perform an advanced table sort. Up to three rows are available to specify the sort criteria and sort direction. A 'Clear' button in the popup allows the user to reset the sort choices.



Usage Restrictions

- OA Framework currently does not support the rendering of these components in a popup region:
  - switchers
  - HGrids
  - graphs
  - Gantt charts
  - table-in-table user interfaces
  - inline attachments
  - message rich text editor
- Do not add a popup directly as an indexed child of a classic table, advanced table, or HGrid. The popup and the item on which it is enabled should be added to a layout, and the layout should be added as the indexed child of the table.
- If you enable a popup on an item in a classic table, advanced table, or HGrid, you must either define the popup region or the enclosing layout as a tableLayout or a messageComponentLayout to ensure that the item prompt renders within the popup.
- If you add a popup under a column in a classic or advanced table, the popup itself cannot contain another classic or advanced table. To build this type of user interface, you should use the table-in-table feature of classic or advanced tables.

Personalization Considerations

- See a summary of Popup personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None.

Related Information

- BLAF UI Guideline
  - none
- Javadoc File
  - oracle.apps.fnd.framework.webui.beans.layout.OAPopupBean
  - oracle.apps.fnd.framework.webui.beans.OAImageBean
  - oracle.apps.fnd.framework.webui.beans.message.OAMessageStyledTextBean
  - oracle.apps.fnd.framework.webui.beans.nav.OAButtonBean
  - oracle.apps.fnd.framework.webui.beans.nav.OALinkBean
- OA Framework Component Reference (Oracle JDeveloper 10*g* Online Help)

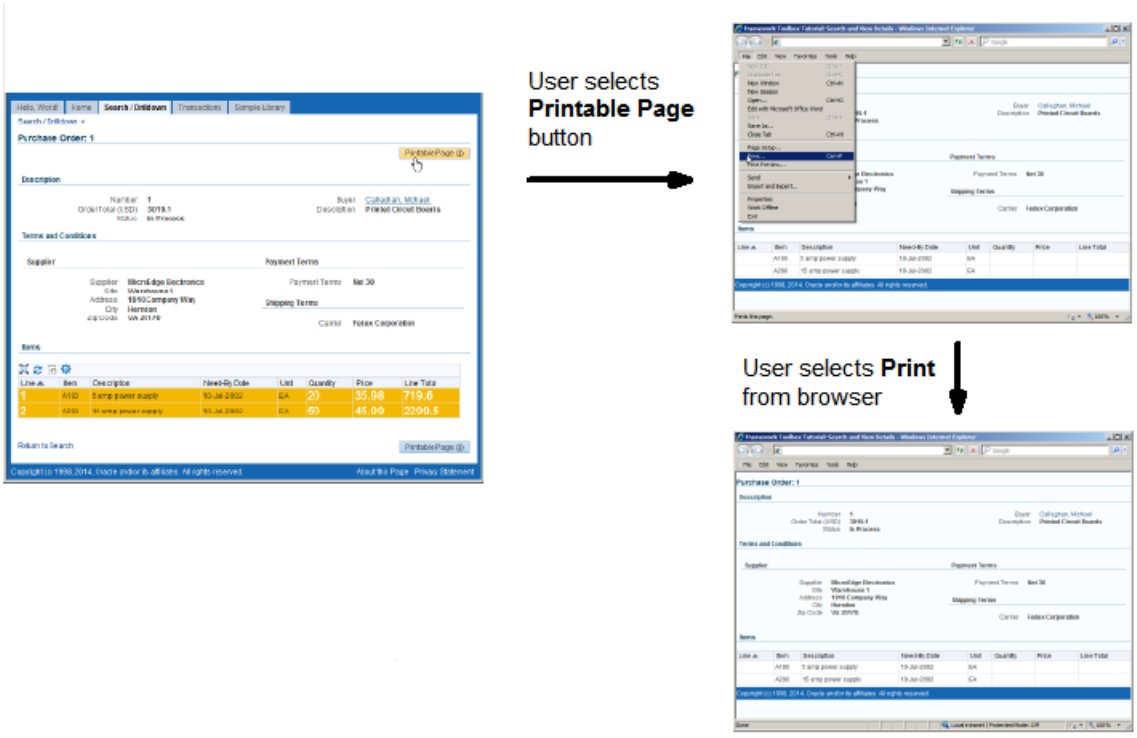
Printable Page

Overview

In accordance with Oracle Browser Look-and-Feel (BLAF) UI Guidelines: Preview and Printable Page Flows, any OA Framework page can be rendered in an optimized state for printing. Printable pages do not include any navigation controls or Personalization links (if Personalization is enabled), and they should open in a new browser window.

Users access the printable version of a given page by accessing a **Printable Page** button as shown in the following flow diagram.

Figure 1: Example "Printable Page" flow



Note: New support for the CSS style OraWrapPreText, which wraps long text entered in a TextArea, now prevents text from being truncated when printed.

Note: As of Release 12.2.4, control bar icons in tables and advanced tables do not render in the printable page. If the table or advanced table displays a Total column, the recalculate icon next to the total renders in the printable page, but is disabled.

Declarative Implementation

On any page that should have a printable state:

Step 1: Add a page-level **button** as described in the Buttons (Action/Navigation) document.

Step 2: Assuming your button label is the standard "Printable Page," apply the OA Framework attribute set `/oracle/apps/fnd/attributesets/Buttons/PrintablePage`.

Step 3: Set the button's `Destination URI` property to point to the current page and set the UIX facet parameter (`OARF`) to `printable` as shown in the following example (See Controlling UIX Rendering Output for additional information about facets). Remember to retain your application module.

```
OA.jsp?page=<CURRENT_PAGE>&retainAM=Y&OARF=printable
```

Note: Earlier versions of OA Framework recommended giving the printable page submit button the `ID` **lcxPrintablePageButton**. OA Framework detects this `ID` and behaves appropriately when the user presses the button. Although this still works, new code leverages the facets approach instead.

Step 4: Set the button's `Target Frame` property to **_blank** to ensure that the printable page opens in a new window.

Tip: If your page requires a request parameter so it can build properly when OA Framework renders the page (for example, if you need to pass a primary key to a details page from a summary page), your Printable Page button needs to put this value on the

request when it is selected. In the ToolBox Tutorial `orace.apps.fnd.framework.toolbox.tutorial.webui.PoDetailsPG`, we show how to bind the button to a view instance and use a URL token to pick up the primary key value.

Runtime Control

If you need to make any supplemental changes to the page over and above the standard OA Framework changes for a printable page, add the following code to the `processRequest` method in a controller associated with the base page.

```
import oracle.apps.fnd.framework.webui.laf;


if (LafUtils.isPrintableFacet(pageContext))
{
   // Printable page mode processing
}
else
{
  // Normal page processing
}
```

Personalization Considerations

- See a summary of Printable Page personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key printable page issues with suggested workarounds if available.

Related Information

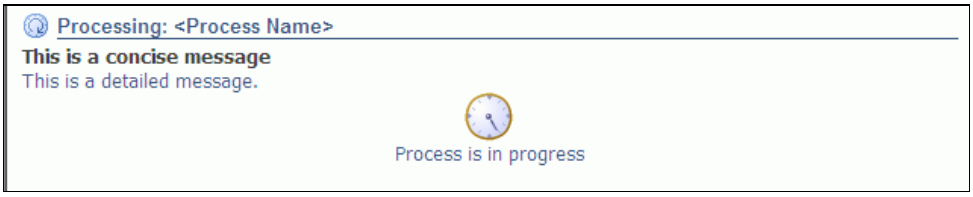
- BLAF UI Guidelines
  - Preview and Printable Page Flows
- OA Framework Developer's Guide
  - Controlling UIX Rendering Output
  - Buttons (Action/Navigation)
- OA Framework ToolBox Tutorial / Sample Library

Processing Page

Overview

Per the Oracle Browser Look-and-Feel (BLAF) UI Guidelines: Processing Templates, you can display a special "Processing" page whenever the user should be told that a long-running process is working in real-time in the background (note that this is appropriate only if the user cannot do anything else while the process is running).

Figure 1: Processing page content (without incremental status)



Note: For Release 12 Back button support, none of the Global buttons or links, except **Home/Return to Portal**, **Logout** and **Close Window** display in the "Processing" page.

Declarative Implementation

Currently, there is no declarative support for this feature.

Runtime Control

To associate a processing page with a long-running process, complete the following two steps.

Note: The current OA Framework Processing page implementation has the following two restrictions:

- Once the processing starts, the background process cannot be cancelled (so no Cancel button displays as shown in the BLAF guideline).
- Only the simple, static processing page (without information about incremental status) is currently supported.

Step 1: Create a Controller for the Processing Page

Create a controller to be used by the Processing page. This controller should actually start the long-running process (in the `processFormRequest` method as shown below); and specify the destination page to display when processing completes.

Note: The target post-processing display page should include a "Confirmation" message so users aren't left to wonder if the process completed successfully.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
   super.processFormRequest(pageContext, webBean);


   // This is where you would launch your process.


   ...

   catch (SQLException e)
   {
    pageContext.putDialogMessage(e)
    pageContext.forwardImmediately(//forward to calling page with retainAM=Y)
   }


   // Finally, tell the OA Framework what page to display when the processing completes
   // (this assumes you display a "Confirmation" message at the top of the target page).
   // Note that you can also display a "Confirmation" dialog instead.

   pageContext.forwardImmediately("<TARGET_PAGE_FUNCTION_NAME>",
                                  OAWebBeanConstants.KEEP_MENU_CONTEXT,
                                  null,
                                  null,
                                  true,
                                  OAWebBeanConstants.ADD_BREAD_CRUMB_YES);
}
```

Fowarding to a Page with Different AM

If you are forwarding to a page that does not share the same application module (AM) as the calling page, the dialog message specified in the error handling section of the code example above will not appear. Instead, you should include the following code in the `processFormRequest` method of the processing page controller:

```
pageParams.put("DisplayErrorMsg", "Y");
pageContext.setForwardURL("FUNC_NAME",
                          KEEP_MENU_CONTEXT,
                          null,
                          pageParams,
                          true,
                          ADD_BREAD_CRUMB_YES,
                          OAException#ERROR);
```

In addition, add the following code to the `processRequest` method of the controller for the forwarded page:

```
String displayErrorMsg = pageContext.getParameter("DisplayErrorMsg");
if ( displayErrorMsg != null && "Y".equals(displayErrorMsg) )
{
 OAException errMessage = new OAException(...);
 pageContext.putDialogMessage(errMessage);
}
```

Step 2: Add Processing Page Access to Launching Page

In the page that initiates the long-running process, add code to instantiate and navigate to the processing page as shown in the example below.

```
import oracle.apps.fnd.framework.webui.OAProcessingPage;
...


public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
   super.processFormRequest(pageContext, webBean);


   // This example assumes a submit button named "StartProcess" initiates the long-running
   // process.
   if (pageContext.getParameter("StartProcess")!= null)
   {
     // Create the processing page and pass it the fully qualified name of the controller that
     // you created to launch the process.
     OAProcessingPage page =
        new OAProcessingPage("oracle.apps.fnd.toolbox.samplelib.webui.processCO");

     // The concise message is displayed in bold and should briefly describe the process.
     // NOTE: never hard-code text display values as shown here.  Always use message dictionary.
     page.setConciseMessage("This is the concise processing page message.");

     // The detailed message displays below the concise message in plain text.  It provides
     // additional information about what's happening.
     page.setDetailedMessage("This is the detailed message which should explain what's happening.");

     // This is displayed in the processing page title.
     page.setProcessName("<Process Name>");

     // Forward to the processing page.  Note that the OA Framework assumes you are retaining
     // the root application module.  Since we haven't specified a different root AM on
     // the processing page, the OA Framework assumes it uses the same root AM as the
     // launching page.
     pageContext.forwardToProcessingPage(page);
   }

}
```

Personalization Considerations

- See a summary of Processing Page personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key Processing Page issues with suggested workarounds if available.

Related Information

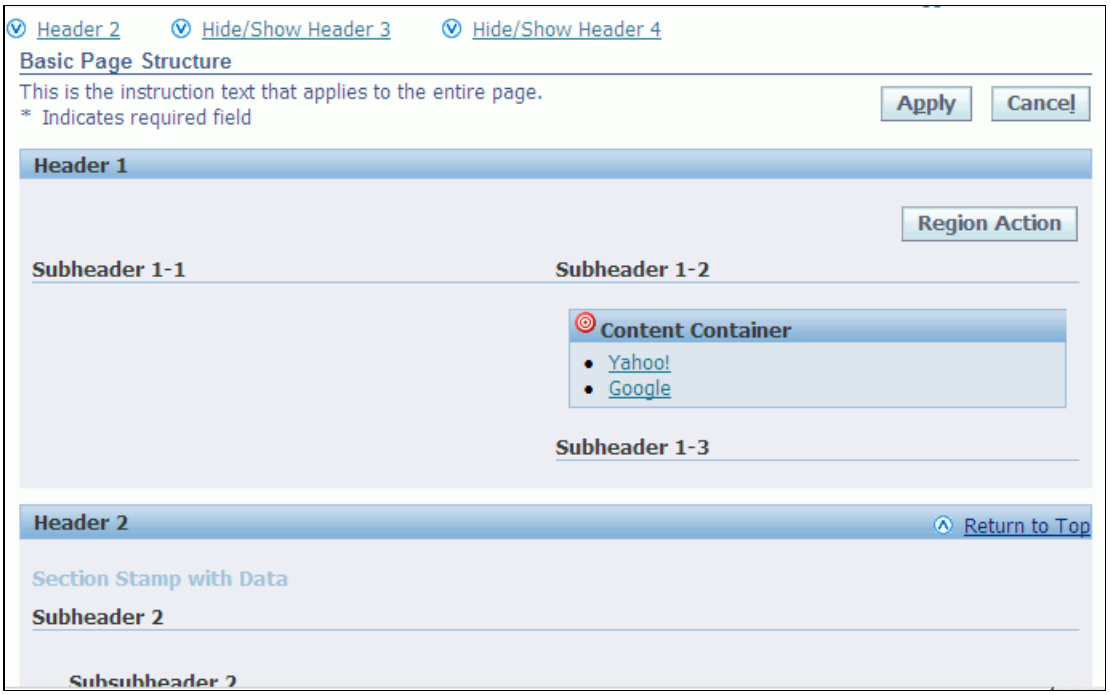
- BLAF UI Guideline(s)
  - Processing Templates
- Javadoc
  - `oracle.apps.fnd.framework.webui.OAProcessingPage`
- OA Framework ToolBox Tutorial / Sample Library
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.ProcessExamplePG.xml`
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.ProcessExamplePageCO.java`
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.ProcessPageCO.java`

Quick Links

Overview

As described in the BLAF UI Guideline: Quick Links specification, Quick Links are a means for users to see and access clearly defined topics within a long page that might otherwise scroll out of sight. Figure 1 illustrates how Quick Links (and the corresponding "Return to Top" links at each target subheader) render in a page.

Figure 1: Example of Quick Links and topic subheaders.

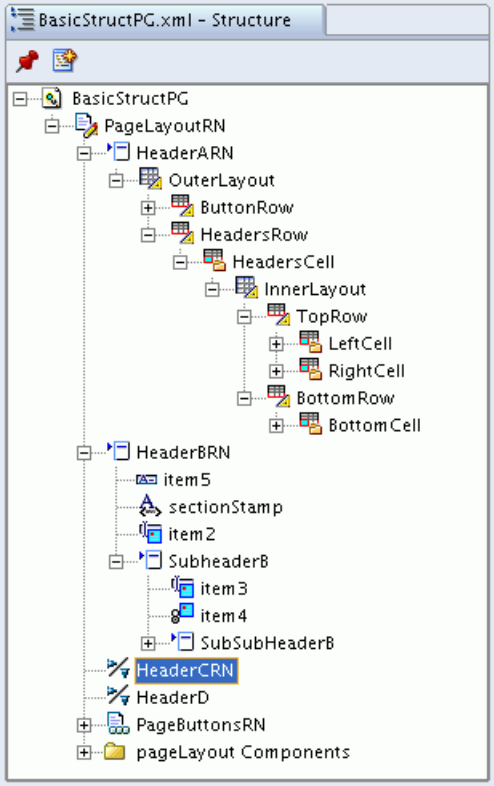


Declarative Implementation

To enable Quick Links, set the **pageLayout** region's `Quick Links Shown` property to **true**. The OA Framework automatically includes all top-level headers (*except for the first which is clearly visible from the top of the page*) in the list of Quick Links. In this case, "top-level headers" are defined as those that you add directly to the **pageLayout** region. For example, consider the **pageLayout** JDeveloper definition for the "Samples" page shown above.

- Only the three headers added to the pageLayout region are eligible to be included in the Quick Links, and the first is always omitted.
- Were any subheaders or subsubheaders to be created beneath these top-level header regions, they would not be included in the Quick Links.

Figure 2: Quick Links example page region relationships to the pageLayout region



Note: The UI Guideline includes recommended minimum and maximum numbers of Quick Links. This is not enforced by the framework; it's up to you to build a compliant design.

Runtime Control

If you want to enable or disable Quick Links programmatically, call `setQuickLinksShown()` on the `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean` as illustrated below.

```
import oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean;
...
processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  ...
   // Assuming the controller is associated with the pageLayout region
   OAPageLayoutBean page = (OAPageLayoutBean)webBean;
   page.setQuickLinksShown(true);
   ...
```

Personalization Considerations

- None

Known Issues

- None

Related Information

- BLAF UI Guidelines:
  - Quick Links
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean`
- OA Framework ToolBox Tutorial / Sample Library
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.BasicStructurePG.xml`

Rating Bar

Overview

Beginning in Release 12.2.2, you can create a new rating bar component that allows users to rate a product, service or entity in an OA Framework-based page. You can define the rating bar component to be updatable or read-only. Users may modify existing rating information or provide new rating information to an updatable rating bar, whereas they may only view existing rating information for a read-only rating bar.

You can also render partial rating images for fractional values. Currently, a half star represents any fractional value for the rating. The partial rating is only applicable for a NUMERIC data type in read-only mode.

You can implement a rating bar in a new page or introduce this component into an existing page.

You can also associate an action with the rating bar, such that when a user selects a rating image, a corresponding action performs.

Figure 1: Example of a read-only partial rating bar and an updatable rating bar within an Employees table.



Note: For the current release, the rating image is represented as a star.

The number of rating images that appear is based on the view object you specify as the Picklist View Object.

For example, suppose your Picklist View Object consists of the following SQL statement and it returns five rows as shown in the table below:

```
SELECT Grade_id, grade_value
FROM   Supplier_Grading_Lkp
```

Value Attribute (Grade_id)	Display Attribute (grade_value)
1	Poor
2	Average
3	Good
4	Very Good
5	Excellent

The resulting rating bar would render five rating images as shown in Figure 1. If the rating bar is updatable, the corresponding "Display Attribute" value appears when you hover over each rating image. If the rating bar is read-only, the "Display Attribute" value of the current rating appears regardless of which rating image position you hover over.

Contents

- Declarative Implementation
- Runtime Control
- Accessibility Support
- Personalization Considerations
- Known Issues
- Related Information

Declarative Implementation

You may use the Oracle JDeveloper OA Extension Property Inspector to add a rating bar component to any region, such as to a messageComponentLayout region or to a table.

You may also associate an action with the rating bar web bean so that when a user selects a rating image, a corresponding action fires.

Step 1: In your pageLayout region, select the region to which you want to add a rating bar component. Select **New** > **Item** from the context menu.

Step 2: Set the `Item Style` property for this new item to **messageRatingBar**.

Step 3: Use the Property Inspector to set the properties on the **messageRatingBar** item, specifically (where * indicates a required property):

- `ID`* - Specify an identifier that uniquely identifies the item in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Attribute Set` - specify an attribute set if you want to maintain a standard look and feel for your item.
- `Read Only` - specify whether the rating bar is read-only. **Note:** Set this property to **Yes** to implement a partial rating bar.
- `Data Type` - specify the data type of the rating bar's value. **Note:** Set this property to **Number** to implement a partial rating bar.
- `Initial Value` - specify a default value if you want the rating bar to render a default selected value.
- `Picklist View Definition`* - specify the fully-qualified view object name for the picklist, cached at the JVM level. The picklist view object must have two attributes, a display attribute and a value attribute. The number of rows queried by the view object determine the number of rating images to render for the rating bar. Be sure to add an instance of this view object in the root application module.
- `Picklist View Instance`* - specify the view instance name for the picklist.
- `Picklist Display Attribute`* - specify the view attribute that serves as the displayed value of choices. When a user hovers the mouse over a rating image, the value of this view attribute displays. This same attribute is also used in accessibility mode.
- `Picklist Value Attribute`* - specify the view attribute that serves as the internal value of choices. This attribute determines the value of the rendered rating image.
- `Prompt` - specify the text label for the rating bar. For example, Rating.
- `View Attribute*` - specify the name of the view attribute.

  Note: The `View Attribute` value for this web bean must correspond to the `Picklist Value Attribute`.

  Note: In Release 12.2.2, the `Add Blank Value` property is not implemented.

Runtime Control

At runtime, OA Framework instantiates an `oracle.apps.fnd.framework.webui.beans.message.OAMessageRatingBarBean`.

You may also programmatically create a rating bar web bean inside any region by including code similar to the example shown below in your controller's `processRequest` method. In this example, the rating bar web bean is added under a message component layout region.

```
//Find the region in the page to which you want to add the rating bar web bean.
OAMessageComponentLayoutBean messageComponentLayoutBean =
   (OAMessageComponentLayoutBean)webBean.findChildRecursive("MainRN");

//Create the instance of rating bar web bean here.
OAMessageRatingBarBean ratingBean = (OAMessageRatingBarBean)
   createWebBean(pageContext, MESSAGE_RATING_BAR_BEAN, null, "RatingID");
ratingBean.setID("RatingID");
//Set the View Object and View Attribute.
ratingBean.setViewUsageName("EmployeeCreateVO1");
ratingBean.setViewAttributeName("PositionCode");
//Set Picklist View Object.
ratingBean.setPickListViewObjectDefinitionName
   ("oracle.apps.fnd.framework.toolbox.poplist.server.PositionsVO");
//Set Picklist Display Attribute and Value Attribute.
ratingBean.setListDisplayAttribute("Meaning");
ratingBean.setListValueAttribute("LookupCode");
//Set rating bar prompt.
ratingBean.setPrompt("Performance Rating");
//Add rating bar web bean to the desired region.
messageComponentLayoutBean.addIndexedChild(0, ratingBean);
```

Accessibility Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still use the rating bar component as follows.

Keyboard Support

Use the following keys to interact with an updatable rating bar:

- **[Left Arrow]** or **[Down Arrow]** - moves the focus to the previous rating image of the rating bar. If the focus is already on the first rating image, pressing either of these keys shifts the focus to the last rating image of the same rating bar.
- **[Right Arrow]** or **[Up Arrow]** - moves the focus to the next rating image of the rating bar. If the focus is already on the last rating image, pressing either of these keys shifts the focus to first rating image.
- **[Tab]** - moves the focus to the next page element. If the current focus is on the rating bar, pressing **[Tab]** shifts the focus to the next element on the page. If the current focus is on an element prior to a rating bar, pressing **[Tab]** moves the focus to the rated image of the rating bar. For example, if a rating bar rates a three out of five, then pressing **[Tab]** from a previous page element shifts the focus to the third image of the rating bar.
- **[Shift+Tab]** - moves the focus to the previous page element from the rating bar. If the previous page element is another rating bar, pressing **[Shift+Tab]** shifts the focus to rated image of the previous rating bar.
- **[Space]** or **[Enter]** - if focus is on a rating image on a rating bar, pressing **[Space]** or **[Enter]** rates the current rating bar with that rating value. For example, if the focus is on the fourth star of a rating bar, pressing either of these keys rates the rating bar with four stars.

Screen Reader Optimized Mode

In Screen Reader Optimize Mode, when you move the focus from one rating image to another, the Screen Reader calls out the corresponding rating image's Alt text. The Alt text for each rating image derives from the Picklist View Instance (display attribute) specified for that rating bar.

Personalization Considerations

You may personalize an OA Framework-based page at the administrative-level to create a new or update an existing rating bar component by using the Create Item page or Update Item page, respectively.

- See a summary of Rating Bar personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key issues with suggested workarounds if available.

Related Information

Related Information

This topic focuses on the basic instructions for creating and configuring a rating bar. See the following documents for additional information related to special behaviors:

- BLAF UI Guideline(s)
  - None
- Developer's Guide
  - To change component properties at runtime (for example, you want to make the rating bar read-only), see the Dynamic User Interface documentation for recommended approaches.
  - To enable partial page rendering (PPR) events when users interact with this component, also see the Dynamic User Interface documentation.
  - To configure some of this item to perform a form submit when selected, see Submitting the Form.
  - To disable "save for changes" warnings for certain components, see the Save Model documentation.
- OA Component Reference
  - For a description of all the possible properties that you can set for the **messageRatingBar** item type, see the OA Component Reference in the JDeveloper online Help. To access, select **Help > Help Topics** from the main menu. In the **Help Navigator** window, expand the **OA Component Reference** and **OA Item Styles** nodes.
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageRatingBarBean`
- OA Framework ToolBox Tutorial / Sample Library
  - None

Record History

Overview

A user often needs to identify who created or updated a record within an OA Framework-based page. For example, a Payables Manager may want to see which Payables Specialist entered a particular invoice, or the manager may need to check when an invoice or invoice line was last updated. Similarly, a Buyer may need to see who created or updated a particular supplier.

Record History is a feature that allows a user to see information about a particular record, such as who created the record and when, and who most recently updated the record and when. This information is generally saved in the standard "WHO columns" of the database tables linked to an OA Framework page through BC4J entity objects (EOs). These standard WHO columns represent the following:

- Name of the user who created the row
- Date of creation
- Name of user who updated the row
- Date when row was last updated

You may implement the Record History feature for a header, classic table or advanced table region. A Record History icon, as shown in Figure 1 and Figure 2, appears when you enable this feature.

Figure 1: Record history for a header region.



Figure 2: Record history for table region.

Empno	Ename	Sal	Job	Dept	Record History
10	Test1	50000	Manager	100	
20	test2	20000	Developer	100	
30	test3	20000	Developer	200	
40	test4	60000	Director	300	
50	test5	20000	Developer	300	
60	test6	50000	Manager	200	

When a user selects the Record History icon, the Record History modal pop-up displays the WHO column values for the current row.

Note: As of Release 12.2.2, the Record History modal pop-up displays the User Name (rather than the User ID) of the "Last Updated By" and "Created By" WHO columns.

Contents

- Usage Restrictions
- Declarative Implementation
- Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

Usage Restrictions

- This feature requires changes to the existing pages and involves uptake by the product developer.
- This feature works if the product teams have already built "WHO columns" into their tables.
- This feature works for header, classic table and advanced tables, with the following caveats:
  - Headers with a nested AM are supported.
  - Classic or advanced tables with a nested AM are not supported.

As of Release 12, to enable this feature after implementing it declaratively, you must set the profile option **FND: Record History Enabled** to **Yes**. The default value for this profile is **No**.

A system administrator may turn on the Record History feature for specific users by setting the profile value to **Yes**.

Declarative Implementation

To implement Record History for a header, table or advanced table region, set the `Record History Enabled` property on the region to **true**.

Setting the `Record History Enabled` property to **true** and the profile option **FND: Record History Enabled** to **Yes** displays the Record History icon as follows:

- Below the header for a header region.
- In a separate column for a classic or advanced table.

Note: If you are using a PL/SQL-based updateable ViewObject on the feature, make sure you populate the "WHO column" on the ViewObject.

Runtime Control

There are no programmatic steps necessary to implement the Record History feature.

The OAHeaderBean, OATableBean and OAAdvancedTableBean expose a method called `setRecordHistoryEnabled(boolean)`.

Personalization Considerations

See a summary of Record History personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key Record History issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline
- Javadoc File
- Lesson
- Sample Code

Related Links / Shortcuts

Overview

Per the Oracle Browser Look-and-Feel (BLAF) UI Guideline "Related" Links / Shortcuts, a related link/shortcut is a link that takes the user to pages whose content relates to the current task. They are strictly a convenience mechanism so the user can access related information or tasks; they are not part of the page's primary focus.

These special links are grouped together and presented in standard locations within a page: within one or more content containers as shown in Figure 1, or within a region subsection as shown in Figure 2 and Figure 3 (see the UI Guideline for region subsection location variations).

Figure 1: Related links in content containers.

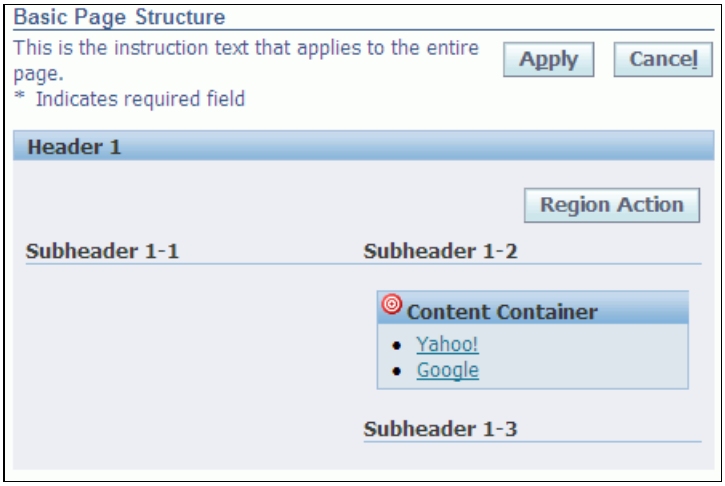


Figure 2: Related links in a region subsection.

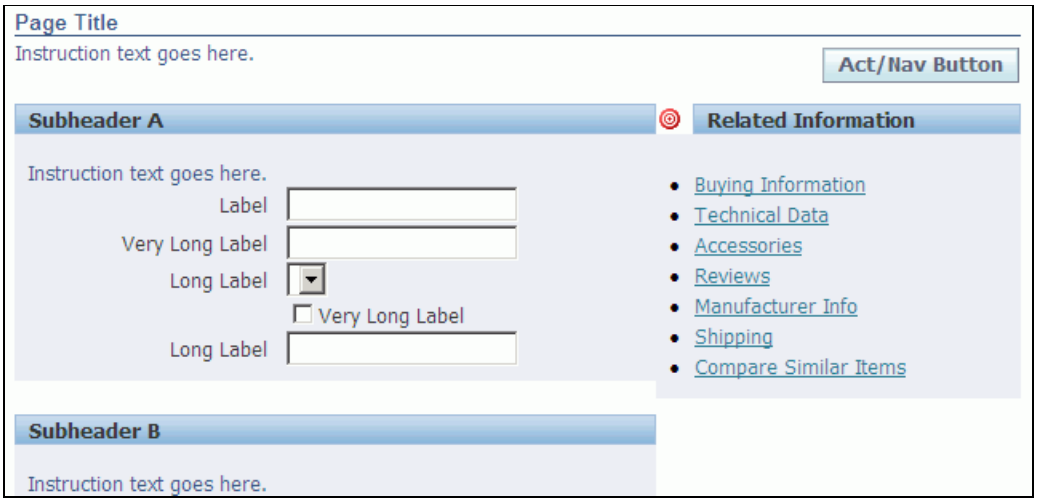
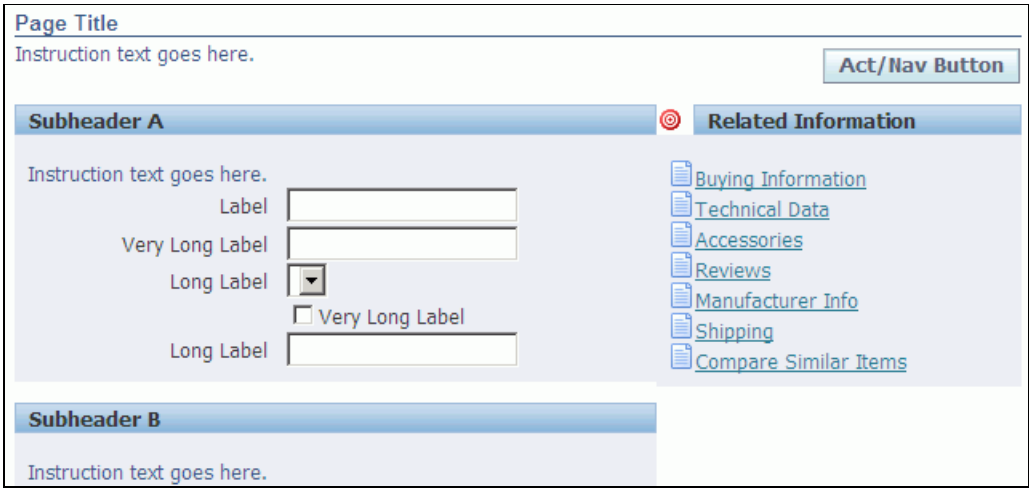


Figure 3: Related Links w/ icons instead of bullets.



Implementation

Regardless of where you actually add your related information links, the following will help you create the individual links:

- If you want to implement your links as a bulleted list, see the Bulleted List documentation.
- If you want to implement your links with images a shown in Figure 3, see Page Layout (How to Place Content)
- For information on creating a link (either in a bulleted list, or in a flow layout with an image), see Buttons (Links).

Related Links in a Content Container

In this context, you are adding your related links to a content container.

For information on how to add a content container to your page, see Content Containers in a Page.

See the ToolBox Tutorial Home Page Lab for an example implementation of this.

Related Links in a Region Subsection

In this context, you are adding your related links to a header.

For information on how to create adjacent subheaders, see Headers.

If you want to add your related links in-line with text items in a region, see Page Layout (How to Place Content).

Personalization Considerations

- None

Known Issues

- None

Related Information

- BLAF UI Guidelines
  - "Related" Links / Shortcuts
- OA Framework Developer's Guide
  - Content Containers in a Page
  - Bulleted Lists
  - Headers
- OA Framework ToolBox Tutorial / Sample Library
  - Home Page Lab

RESTful Service Interface

Overview

REST, (**RE**presentational **S**tate **T**ransfer), is an architectural style for distributed hypermedia systems. REST is similar to SOAP-based (**S**imple **O**bject **A**ccess **P**rotocol) web services, but with much less payload per message. REST is architecturally tailored for frequent and lightweight handshakes between the browser and the middle tier server. Some usability enhancements in Oracle E-Business Suite, such as the rich UI clients (both desktop- and browser-based), need REST interfaces on top of Oracle E-Business Suite to exchange business data and reuse existing business logic. The term 'RESTful' refers to conforming to REST constraints.

Note: REST Interfaces are supported by Oracle E-Business Suite Release 12.1.2 and higher, and by a subset of the browsers certified with Release 12. For Microsoft Internet Explorer (IE), this feature is only supported in version 7.0 and higher.

Contents

This document contains the following topics:

- Overview
  - REST principles
  - Service Types
- Architectural Overview
- REST Request and Response
- Using RESTful Services
- Coding Standards for Rich Clients
- Known Issues
- Related Information

REST principles

The following principles guide the design of the OA Framework RESTful service interface:

- **Resource** - Any method eligible for exposure as a RESTful service in Oracle E-Business Suite is considered a resource.
- **URL** – Each service in the system is accessible via a uniquely addressable URL.
- **Protocol** – REST architecture is tightly associated with the HTTP protocol, and assumes HTTP as the default protocol.
- **Stateless communication** – The communication between the server and client is, by default, stateless in a REST-based approach and is enforced through the HTTP protocol. However, in a business application, the application state needs to be maintained to support various functionalities and is achieved through the use of cookies.
- **Data-centric approach** – A REST service takes a data-centric, rather than a document-based or procedure-based approach.

Service Types

At a high level, the RESTful service interface supports services classified into two categories, as described in the table below. Type I services are application business services. This document's scope is limited to Type I services (Model and Search). Type I services can be used by developers creating RESTful services. Type II services are internal to OA Framework and are used only by new UI components (such as Inline LOV) introduced in Release 12.1.2. Discussion of Type II services in this document are for informational purpose only.

Classification	Service Type	Description
Type I	Model	Public methods in Oracle E-Business Suite products' application modules are exposed as services.
Type I	Search	OA Framework's `OAQueryBean` is exposed as a service.
Type II	LOV	LOV results based on a search string.
Type II	Attachments	Attachment view, update and create operations.
Type II	MainMenu	Functions under a specific responsibility.
Type II	Custom	Dependent on the implementation.
Type II	Personalization	Dynamic User-level personalization.

Architectural Overview

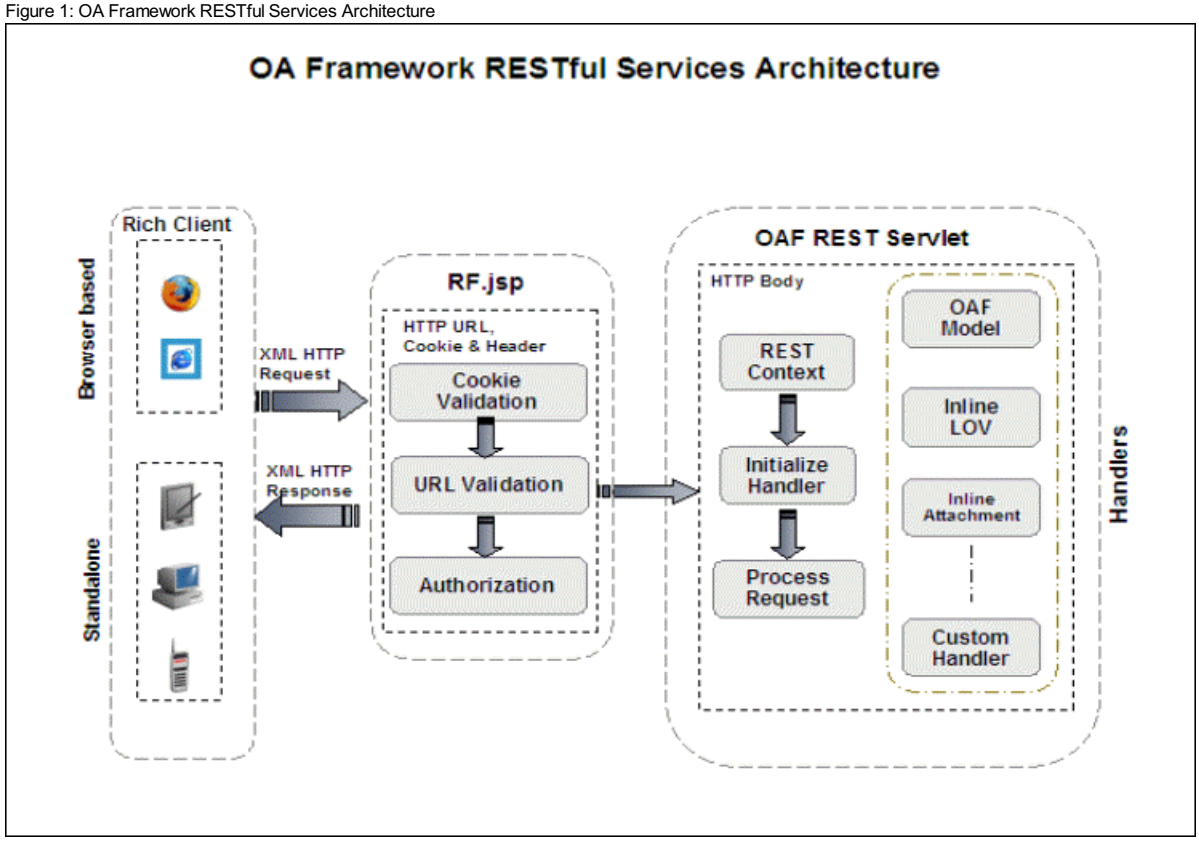
OA Framework RESTful Services are intended for use by rich clients that are standalone widgets on a desktop or are embedded objects in an OA Framework page running within a browser. RESTful services are exposed through the AOL/J `RF` (`RunFunction`) interface. The AOL/J `RF` interface has been enhanced to act as the REST interface or gateway for all Oracle E-Business Suite RESTful services. To expose a Type I RESTful service through `RF.jsp`, the service was first registered as an FND Function. The service can then be invoked from the client code by sending a XML HTTP `POST` request to `RF.jsp` with the Function name (or ID).

All required runtime parameters can be passed in the request body.

Note: There is no support for any HTTP operation other than `POST` on the RESTful service interface.

`RF.jsp` performs security checks including cookie validation and authorization and then forwards the request to the **OAF REST Servlet**. The servlet builds the `RESTContext` object, identifies and initializes the corresponding REST handler and processes the request. The response is then sent back to the client in XML format, as illustrated in Figure 1.

Figure 1: OA Framework RESTful Services Architecture



RF.jsp

As mentioned above, the AOL/J `RF.jsp` interface that currently serves as a generic, function-secured entry point to the Oracle E-Business Suite UI is enhanced to serve as the entry point for all Oracle E-Business Suite RESTful service requests. The `RF` interface performs security checks before delegating the request to an OA Framework REST servlet. Based on the service type, further delegation is possible to one of many service type-specific handler implementations.

URL Validation

The AOL/J layer is enhanced to support a new FND Function type called **REST**. Any Type I service available through the REST interface is registered as a function of this type. The `RF.jsp` URL for the function serves as the unique address of the service. For example:

```
http://<host_name>:<port>/OA_HTML/RF.jsp?function_id=TEST_REST_FUNC
```

For the parameter `function_id`, `RF` expects either a FND function ID or function name. Gadgets use the **function name**, as **function id** is not unique across instances. Only HTTP `POST` requests are supported in the current release.

Cookie Validation (Authentication)

A cookie, set during the login process, accompanies each HTTP request. `RF.jsp` validates the cookie and forwards the request to OAF REST Servlet only if the validation succeeds. The client sending the request must be within an authenticated OA Framework context to be able to send the cookie with each request.

Authorization (Function Security)

`RF.jsp` uses the FND function ID or name sent by the rich client to authorize the request. `RF.jsp` forwards the request to the OAF REST servlet only after verifying that the current user has access to the function. It returns an appropriate HTTP error message if the

authorization check fails.

OAF REST Servlet

The OAF REST servlet is responsible for creating REST Context, initializing the service handlers, processing the request and writing the response back to the client. The servlet supports multiple service types through separate service handlers.

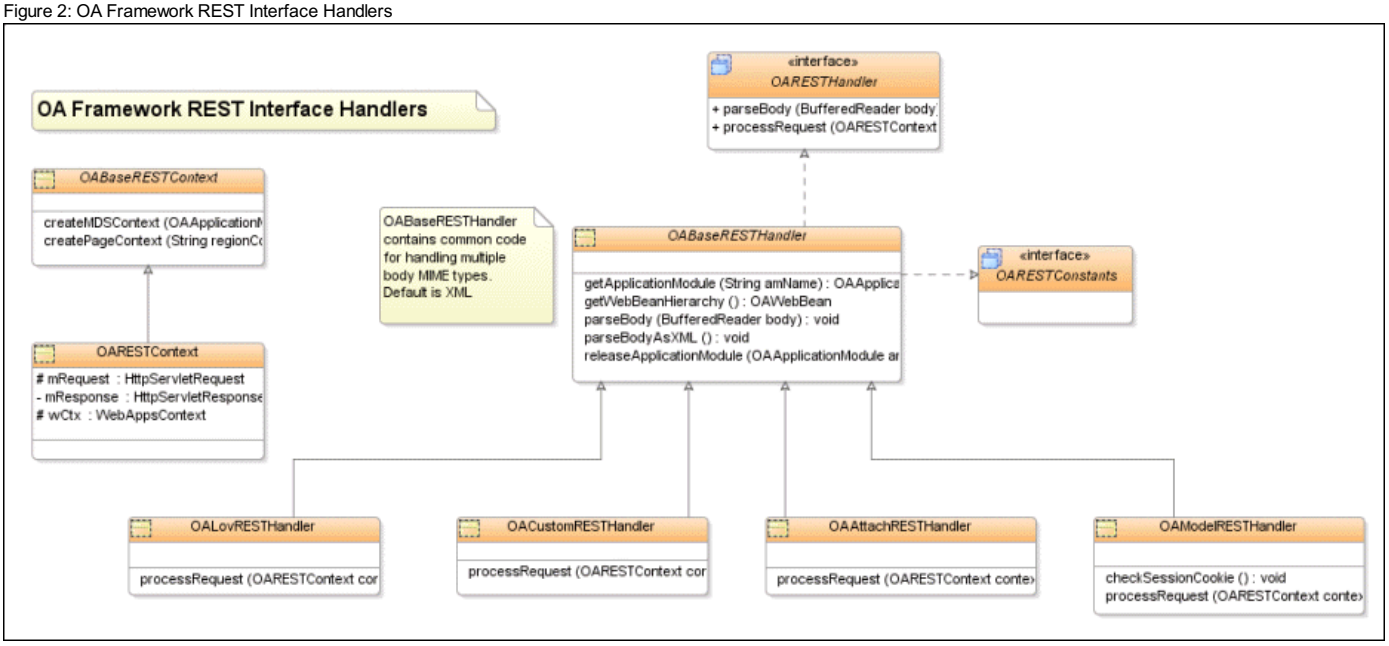
REST Context

The `RESTContext` object represents the context for execution of a RESTful service. It serves as a central point to store all context variables in all stages of the RESTful service's life cycle. Some of the context-specific variables stored are the HTTP session, WebAppsContext, JSP page context, Function, and so on.

REST Handlers

A REST Handler object contains the core processing logic for a specific request type. Each REST Handler supports a different REST service type. All handlers extend a base handler that contains common processing logic such as XML body parsing and Application Module check out. The OAF REST Servlet identifies and initializes the specific handler object required and invokes two predefined methods: `parseBody()` and `processRequest()`. These methods are specified by the interface `OARESTHandler`, which all handlers implement. Figure 2 below shows a class diagram that explains the hierarchy of REST Handlers.

Figure 2: OA Framework REST Interface Handlers



Initialization & Body parsing

Based on the REST request, the `RESTHandlerFactory` identifies the handler to be instantiated and initialized. The handler parses the XML request body and stores the runtime parameters in the `RESTContext` object in canonical form. This is specified by the `parseBody()` method in the REST handler.

Request processing

Each handler has separate code for processing a request, as specified by the `processRequest()` method. The processing logic for each of the handlers is outlined as follows:

- **Model Handler**
  - This handler first identifies the Application Module name and method name from the function parameters.
  - It then checks out the AM instance from the AM pool. If the AM is not already checked out in the current session context, a new instance is obtained.
  - The parameters are retrieved from the `RESTContext` object and the method is invoked through the `invokeMethod` API on the AM instance.
  - The response is stored in the `RESTContext` object.
  - The AM instance is released back if it was a new instance obtained only for the current request.
- **Search Handler**
  - This handler supports the OA Framework query web bean functionality as a RESTful service.
  - The query is sent in XML format (as specified in Using RESTful Services - Search Services).
  - It checks out the required Application Module, executes the query and sends the response back in XML.

    - The AM instance is retained or released based on client input (as discussed in Using RESTful Services - Search Services).
- **LOV Handler**
    - The client sends the LOV criteria through the request.
    - The handler checks out the LOV AM.
    - The LOV criterion is transformed into a query and is executed.
    - The query result is transformed into XML and is stored in the `RESTContext` object.
    - The LOV AM instance is retained as multiple requests on the same AM instance are expected.
- **Attachments Handler**
    - This handler supports CRUD (Create, Read, Update, Delete) operations on OA Framework attachments.
    - The rich client sends the attachment ID in a specific format through the request.
    - The handler checks out the AttachmentAM, performs the requested operation on it and returns the response back.
    - The AM instance is nested under the page's root AM.
- **Main Menu Handler**
    - This handler returns selected functions or responsibilities based on the request parameters.
    - The functions and responsibilities granted to a user are available through BC4J objects.
    - This handler receives requests based on the user actions on the enhanced **Navigator** menu that appears on each OA Framework page.
- **Personalization Handler**
    - This handler first identifies the Application Module name, table name and operation from the function parameters.
    - It then uses the values provided for operation to reorder or resize the table specified by the table name.
    - This handler retains the changes in MDS by creating a personalized view for the page that initially called the Personalization Handler.

Note: Type I services are handled by Search and Model REST handlers and Type II services are handled by all others.

REST Request and Response

HTTP Request

Rich clients construct a valid REST request as follows:

- URL - Each functionality is uniquely identified by the FND function through the `RF.jsp` URL:

  `http://<host>:<port>/OA_HTML/RF.jsp?function_id=<function_id_or_name>`

  The specific details such as the type, method name to be invoked, as so on, are wrapped in the function parameters.

- Body – All runtime parameters specific to the functionality go in the body. The supported format is text/XML.
- Cookie – The Session ID is retrieved from a cookie and is used to validate the current session by `RF`.

HTTP Response

The response received by the rich client is in the following form:

- If the processing is successful, the response status is set to `200` and the REST handler output is set on the body. The handler output is optional and can be empty.
- If the given function name or user session is invalid, a `HTTP 404` or `401` error with no body is returned by `RF.jsp`.
- If the request processing is not successful due to an Exception, the response status is set to `500` and an error message is set on the body. (An error stack is returned only if Diagnostics are enabled).

Using RESTful Services

Methods on OAApplicationModule

The following new methods have been added to the `oracle.apps.fnd.framework.server.OAApplicationModuleImpl` class and are available on every Application Module (AM):

- `getRowData`
- `saveRowData`

Refer to the Javadoc files for additional information.

Note: In Release 12.1.3.2, OA Framework introduced an overloading method for `getRowData`, and includes a new parameter `lastPageIterMode` that can be passed to specify the Last Page Iteration Mode when a range reaches the end of the Row Set. The two supported last page iteration modes, as defined in OARESTConstants, are:

- `OARESTConstants.ITER_MODE_LAST_PAGE_PARTIAL` - indicates partial last page iteration mode. When this mode is

specified, the BC4J layer does not try to pull rows from the previous range to fill the last page to form a full page (such as returned row count equals the RangeSize).
- `OARESTConstants.ITER_MODE_LAST_PAGE_FULL` - indicates full last page iteration mode. When this mode is specified, the BC4J layer would try to pull as many rows as possible from the previous range to fill the last page to form a full page (such as returned row count equals the RangeSize). If a 'null' value is passed for 'lastPageIterMode', then the default BC4J behavior takes place.

The following is a code example for invoking `getRowData` with 'lastPageIterMode' from the Application Module:

```
//specify dept count
deptCount = new Integer(0);

//specify range start
rangeStart =new Integer(0);

//specify range size
rangeSize = new Integer(10);

//specify last page iteration mode
lastPageIterMode = new
Integer(OARESTConstants.ITER_MODE_LAST_PAGE_PARTIAL);

//invoke getRowData to retrieve VO data in XML format
Node output = (Node) getRowData(voName, deptCount, rangeStart, rangeSize,
lastPageIterMode);
```

Model Services

To create and invoke a Model REST Service:

Step 1: Identify the Application Module and the method in it to be available as a RESTful service.

Step 2: Create an FND Function with the following attributes:

- **Web HTML Call** - OAR
- **Type** - REST
- **Function Parameters** - type=model&amName=<*fully_qualifiied_am_name*>&methodName=<*method_name*>&childAMName=<*child_am_name*>

  Note: Providing a ChildAMName is optional when specifying Function parameters. It is only required when there is a nested AM hierarchy and you want to invoke the method on a nested AM.

Step 3: Clients should use `RF.jsp` as the entry point to the REST interface:

- `POST` is the only request type supported for accessing OA Framework Model methods.
- Clients should invoke the API through the following URL to perform a RESTful invocation to the corresponding Application Module method:

  `http://<host>:<port>/OA_HTML/RF.jsp?function_id=<function_name_or_id>`

- As of Release 12.2.5, to take control of AM retention behavior for a REST request, use the `retainAM` <param> tag rather than add code in your method to set `REST_CHECKOUT_FLAG`.
- Make sure the HTTP body contains XML-ized parameters, in order, as in the example code for an Adobe Flex client:

```
<mx:HTTPService id="fetchDetailsService" method="post"
   url= "http://<host>:<port>/OA_HTML/RF.jsp?function_id=TEST_REST_MODEL_FUNC"
result="handleFetchDetailsService(event)"
fault="handleFetchDetailsErrorService(event)"
resultFormat="e4x">
</mx:HTTPService>
...
var xmlreq:XML =<params><param>EmployeeSummaryVO1</param><param>0</param>
  <param>0</param><param>10</param></params>;

fetchDetailsService.contentType = "text/xml";
fetchDetailsService.request = xmlreq;
fetchDetailsService.send()
```

- The Session ID is retrieved from the cookie, which is used to validate the current session. For gadgets included in an OA Framework page, the parameters are passed automatically when a `POST` request is made. The response is XML-ized and the output is wrapped around the `<response>` tag.
- The HTTP response status indicates if the request is successful. Status 200 indicates success, while any other value indicates

error.

Step 4: A sample response when invocation is successful looks like this:

```
<response status="200">1</response>
```

An error sample looks like this:

```
<response status="500"><error>Unexpected Error while processing REST request</error>
  <stack>...</stack></response>
```

Note: Adobe Flex clients can use either the HTTPService or URLRequest objects. Ajax clients can use the XMLHttpRequest object.

Search Services

The prior mechanism to invoke the Search API by using the URL below as a REST service has been deprecated:

`http://<host>:<port>/OA_HTML/RF.jsp?function_id=FWK_SEARCH_REST_SERVICE`

Beginning in Release 12.2.5, to create and invoke a Search REST Service:Step 1: Identify the Application Module and the View Object that needs to be executed as part of this service.Step 2: Create an FND Function with the following attributes:

- **Web HTML Call** - OAR
- **Type** - REST
- **Function Parameters** - type=queryl&rootAM=<*fully_qualifiied_am_name*>&voName=<*view_object_name*>&searchAM=<*search_am_name*>

  The following form parameters can be specified:

  - `rootAM` (required) - specify the root AM of the current page.
  - `voName` (required) - specify the view object name.
  - `searchAM` (optional) - specify the search AM.

  Note: Providing a searchAM is optional when specifying Function parameters. It is not required when the View Object is part of the root AM, only when it is part of the searchAM. The Search REST service ensures that the search is performed only on a View Object present in the current root AM or the searchAM.

Step 3: Clients should use `RF.jsp` as the entry point to the REST interface:

- `POST` is the only request type supported for accessing OA Framework Search Services.
- Clients should invoke the API through the following URL to perform a RESTful invocation to the corresponding View Object:

  `http://<host>:<port>/OA_HTML/RF.jsp?function_id=<function_name_or_id>`

Step 4: Make sure the HTTP body contains XML-ized parameters. The following body parameters can be specified:

- `lastPageMode` (optional) - controls the last page iteration mode. The value passed to this parameter may be "partial" or "full". If not specified, defaults to the "full" mode BC4J behavior.
- `rangeStart` (optional) - depicts the start index of the result. Default value is 0.
- `rangeSize` (optional) - depicts the number of the records returned. Default value is 10.

The `query` parameter may also expect or require the following information:

- `<criterialist>` - This Is an optional tag and can have a maximum of 1 occurrence. It has the following attributes:
  - joinCondition - an optional attribute, with a default value of **AND**
  - isCaseSensitive - an optional attribute with default value of **false**.
- `<criteria>` - This can only be present as a child of the `<criterialist>` tag, with minimum of 1 and maximum of n occurrence. It has following children:
- `<value>` - A mandatory item that specifies the value of the item being queried.
- `<condition>` - An optional item whose value is one of the following: **is**, **starts with**, **ends with**, **contains**, **before**, **after**, **greater than** or **less than**. The default value is **is** for number or date data type or **starts with** for all other data types.
- `<attributeName>` - A mandatory item that specifies the view attribute of the item being queried.
- `<parameters>` - An optional tag with a maximum of 1 occurrence that receives values for the bind parameters defined, if any, in the view object.
- `<parameter>` - There can be as many parameter items as the number of bind parameters defined in the view object. The attributes, index value and dataType are mandatory. A valid value for an index should start with 0. The attribute dataType supports the valid data type defined in the model layer.

Note: The `whereClause` query parameter is no longer supported as part of the request payload.

Example Scenarios

The list below identifies some possible scenarios from which Search services can be invoked, followed by the form parameters and HTTP body Request payload parameters that are expected:**Scenario 1** - The client is a page built using independent custom technology, such as Adobe Flex. Set the `rootAM` parameter to the query AM. Specify the `rootAM`, `voName` in the form parameters and the `query` parameter in the request payload body.**Scenario 2** - The client is an OA Framework page with a query AM present in the page definition itself, but the rootAM and the query AM are different. Specify the `rootAM`, `searchAM`, and `voName` in the form parameters and the `query` parameter in the request payload body.**Scenario 3** - The client is an OA Framework page where the query AM is NOT in the page definition itself and is different from the rootAM. The client will not be able to invoke the View Object which is part of the searchAM. The searchAM needs to be a child of the rootAM.The response is XML-ized and the output is wrapped around the `<response>` tag.The following shows example code from an Adobe Flex client:

```
<mx:HTTPService id="fetchDetailsService" method="post"
  url= "http://<host>:<port>/OA_HTML/RF.jsp?function_id=FWK_SEARCH_REST_SERVICE"
result="handleFetchDetailsService(event)"
fault="handleFetchDetailsErrorService(event)"
resultFormat="e4x">
</mx:HTTPService>
```

The corresponding sample input XML looks like this:

```
<params>
<param name ="query" >
<queryData>
<criterialist joinCondition="OR" isCaseSensitive ="true">
<criteria>
<condition>less than</condition>
<value>9000</value>
<attributeName>Empno</attributeName>
</criteria>
<criteria>
<condition>starts with</condition>
<value>MANAGER</value>
<attributeName>Job</attributeName>
</criteria>
...
...
</criterialist>
<parameters>
<parameter index ="0" value ="1981-04-02" dataType ="oracle.jbo.domain.Date"/>
<parameter index ="1" value ="JONES" dataType ="java.lang.String"/>
</parameters>
<queryData>
</param>
<params>
fetchDetailsService.contentType = "text/xml";
fetchDetailsService.request = xmlreq;
fetchDetailsService.send();
```

The sample response XML looks like this:

```
<response status="200">
<EMPVO>
<EMPVORow>
<Empno>9996</Empno>
<Ename>LovCh1</Ename>
<Deptno>83</Deptno>
</EMPVORow>
</EMPVO>
</response>
```

can have a maximum of 1 occurrence. It has the following attributes:
joinCondition - an optional attribute, with a default value of **AND**. isCaseSensitive - an optional attribute with default value of **false**.
`<criteria>` - This can only be present as a child of the `<criterialist>` tag, with minimum of 1 and maximum of n occurrence. It has following children:
`<value>` - A mandatory item that specifies the value of the item being queried.`<condition>` - An optional item whose value is one of the following: **is**, **starts with**, **ends with**, **contains**, **before**, **after**, **greater than** or **less than**. The default value is **is** for number or date data type or **starts with** for all other data types.`<attributeName>` - A mandatory item that specifies the view attribute of the item being queried.
`<whereClause>` - An optional tag with a maximum of 1 occurrence that specifies any additional whereclause.
`<parameters>` - An optional tag with a maximum of 1 occurrence that receives values for the bind parameters defined, if any, in the view object.`<parameter>` - There can be as many parameter items as the number of bind parameters defined in the view object. The attributes, index value and dataType are mandatory. A valid value for an index should start with 0. The attribute dataType supports the valid data type defined in the model layer.

Example Scenarios

The list below identifies some possible scenarios from which Search services can be invoked, followed by the body parameters that are expected in those scenarios:**Scenario 1** - The client is a page built using independent custom technology, such as Adobe Flex. Set the `rootAM` parameter to the query AM. Specify the `rootAM`, `voName` and `query` parameters in the body.**Scenario 2** - The client is an OA Framework page with a query AM present in the page definition itself, but the rootAM and the query AM are different.Specify the `rootAM`, `searchAM`, `voName` and `query` parameters.
Scenario 3 - The client is an OA Framework page where the query am is NOT in the page definition itself and is different from the rootAM.Specify the `rootAM`, `searchAMDef`, `voName` and `query` parameters.
The response is XML-ized and the output is wrapped around the `<response>` tag.The following shows example code from an Adobe Flex client:

```
<mx:HTTPService id="fetchDetailsService" method="post"
  url= "http://<host>:<port>/OA_HTML/RF.jsp?function_id=FWK_SEARCH_REST_SERVICE"
result="handleFetchDetailsService(event)"
fault="handleFetchDetailsErrorService(event)"
resultFormat="e4x">
</mx:HTTPService>
```

The corresponding sample input XML looks like this:

```
<params>
<param name ="rootAM">oracle.apps.fnd.framework.toolbox.labsolutions.server.EmployeeAM</param>
<param name ="voName" >EMPVO</param>
<param name ="query" >
<queryData>
<criterialist joinCondition="OR" isCaseSensitive ="true">
<criteria>
<condition>less than</condition>
<value>9000</value>
<attributeName>Empno</attributeName>
</criteria>
<criteria>
<condition>starts with</condition>
<value>MANAGER</value>
<attributeName>Job</attributeName>
</criteria>
.....
.....
</criterialist>
<whereClause>SAL != 5000</whereClause>
<parameters>
<parameter index ="0" value ="1981-04-02" dataType ="oracle.jbo.domain.Date"/>
<parameter index ="1" value ="JONES" dataType ="java.lang.String"/>
</parameters>
<queryData>
</param>
<params>
fetchDetailsService.contentType = "text/xml";
fetchDetailsService.request = xmlreq;
fetchDetailsService.send();
```

The sample response XML looks like this:

```
<response status="200">
<EMPVO>
<EMPVORow>
<Empno>9996</Empno>
<Ename>LovCh1</Ename>
<Deptno>83</Deptno>
</EMPVORow>
</EMPVO>
</response>
```

-->

Controlling AM Retention Behavior for REST Requests

Beginning in Release 12.2.5, to take control of AM retention behavior for REST requests, use the `retainAM` <param> tag rather than set the `REST_CHECKOUT_FLAG` in your AM method. Specify the `retainAM` <param> tag as a body parameter for a Model or Search REST service as follows:

`retainAM` (optional) - indicate whether the AM checked out during this RESTful request needs to be released at the end of the request processing. Valid values are:

- Y - AM is to be retained
- N - AM is to be released

Note: If the AM used by REST is associated with any region of an OA Framework-based page in the same transactional flow, you should carefully evaluate your design and functional requirements to determine whether or not `retainAM` should be set since releasing or retaining this AM during the REST request will impact the behavior of the base OA Framework page.

Examples

Sample request body for a Model Service:

```
<params>
   <param>EmployeeSummaryVO1</param>
   <param>0</param>
   <param>0</param>
   <param>10</param>
   <param name ="retainAM" >N</param>
</params>
```

Sample request body for a Search service:

```
<params>
     <param name=="rangeStart" >N</param>
     <param name=="rangeSize" >N</param>
     <param name ="retainAM" >N</param>
<params>
```

Coding Standards for Rich Clients

Use `application/xml` in the Request/Response Content-Type header. Do not use any other MIME-type such as `text/xml` or `application/x-www-form-urlencoded`.
Do not specify character set information in the Content-Type header. Let the XML parser decide the character set by `<?xml>` tag-encoding the information.
Always use UTF-8 for XML. The encoding attribute of the `<?xml>` tag must be specified as `utf-8`. For example `<?xml version='1.0' encoding="utf-8"?>`.

Known Issues

Related Information

BLAF UI Guideline(s) None Javadoc File(s) `java.lang.Integer` `java.lang.Boolean` `java.lang.String` `org.w3c.dom.Node` `oracle.jbo.server.ViewObjectImpl.writeXML` `oracle.jbo.server.ViewObjectImpl.readXML` `oracle.apps.fnd.framework.server.OAApplicationModuleImpl`

Rich Content Container

Overview

A Rich Content Container is a component designed to hold external content within an OA Framework page. Examples of external content include embedded analytics from Oracle Business Intelligence Applications (OBIA), Adobe Flex or third-party Oracle-compatible widgets. The Rich Content Container component can accept either an URL or URI of the rich content. The rich content may be embedded within an OA Framework page directly, or enclosed within an iFrame depending on the rich content source. The Rich Content Container exposes a set of parameters that can be set by the root OA Framework application and passed to the rich content URL, allowing context-sensitive content. The Rich Content Container has no specific user interactions.

Contents

- Overview
- Security
  - Authentication
  - Authorization
  - Cross-Site Scripting
- Declarative Implementation
- Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

Security

The Rich Content Container has security requirements, primarily because embedded third-party content is outside the realm of the Oracle E-Business Suite instance. The security aspects are handled as follows:

Authentication

Authentication is required for both the container component and for the content. The container component is an OA Framework component and is authenticated as long as the containing page is authenticated. Content authentication, however, may be handled in several ways:

- Authentication as a Service - Established coding standards are available for third-party widget developers. These standards describe the services that need to be called and checked to verify validity of the user and the user session before any content code gets executed. It is the responsibility of the content developer to follow these authentication verification guidelines. The authentication credentials are part of the ICX Session Cookie which is passed to the third-party application when the Rich Content Container invokes a HTTP request to fetch the third-party content.
- Java Authentication and Authorization Services (JAAS) – Third-party gadget developers can leverage the AOL JAAS implementation API that can be deployed in the same container where the third-party rich application runs.
- Technology-specific authentication – As an example, Oracle Business Intelligence Suite Enterprise Edition (OBIEE) has come up with its own mechanism to understand Oracle E-Business Suite users through single sign-on (SSO) integration, allowing users to drill down from Oracle E-Business Suite to OBIA and vice versa.

Authorization

Authorized access to the embedded widget and to the content that the widget is going to show, is required.

- To authorize access to the embedded widget, you can define a FND Function for each content that is embedded. Authorized access to this function ensures that OA Framework renders the Rich Content Container Component. If access to the FND Function is not authorized, the Rich Content Container is not rendered, and no request is made to the third-party application. (**Note:** There is no need to define a FND Function for OBIEE-type content. Since OBIEE has tight integration with Oracle E-Business Suite, function and data security is ensured in the OBIEE layer itself.)
- The rich application that generates the content should handle the authorization of the content that the widget will show.

Cross-Site Scripting

Rich content is automatically embedded in an OA Framework page in one of two ways, depending on the source of the rich content:

Case 1 – Embed within page (for Flash content)

```
// OA page source
…
<embed src="<URL of third-party content>" />
```

…

Case 2 – Embed within an iFrame in a page (for OBIEE content)

```
// OA page source
…
<iframe src="<URL of third-party content>" />
…
```

Generally, the source of the rich content URL is from a domain outside Oracle E-Business Suite. The Rich Content Container allows such external content to be drawn in and embedded within the OA Framework page. While the cross-site domain model in browsers prevents any such added content from doing a POST to the Oracle E-Business Suite server, there is always the risk that any script added by the rich content can potentially tamper with the form values of the base OA Framework page. However, given that these widgets are added only by application developers or administrators, these are considered trusted widgets.

Declarative Implementation

To add a Rich Content Container to a region using Oracle JDeveloper OA Extension:

Step 1: Copy your rich content widget to your `jdevhome/myhtml/OA_HTML/<product_code>/webui` folder. For example, if your rich content is from Adobe Flex Builder, move the Flash executable file, `<FlexFileName>.swf`, to the `jdevhome/myhtml/OA_HTML/<product_code>/webui` folder.

Note: This step is not required for OBIEE or other third-party Oracle-compatible content.

Step 2: To embed rich content, define a FND function to secure the widget being added to the Rich Content Container. To create a function:

- Start Oracle E-Business Suite in your development environment and log in as a user with the System Administrator or Application Developer responsibility.
- Select either the "System Administrator" or "Application Developer" responsibility.
- Select **Application** > **Functions** from the main menu.
- To add a new function, set values for the following properties (navigate through the form's content tabs to do this). Note that you can accept all the other default values.
    - **Function** - specify a unique developer key for the function. You can name this whatever you wish, as long as you follow any naming conventions used by your product team. This is the function name that you will later set as the `Destination Function` on the Rich Content Container.
    - **User Function Name** - specify a unique, user-friendly version of the function name that displays.
    - **Description** - provide a short description of the associated widget.
    - **Type** - set this to "Widget".
    - **HTML Call** - provide the required function Web HTML Call:
        - For Adobe Flex Builder content, enter your Flex component URL, specified as `/OA_HTML/<product_code>/webui/<FlexFileName.swf>`.
        - For OBIEE content, enter the Web HTML Call as `GWY.jsp&targetAppType=OBIEE`.
        - For OTHER content, enter the web HTML call as, `GWY.jsp?targetAppType=CUSTOM`.

Step 3: In the OA Extension Structure pane, select the region in which you want to add the rich content and add a **richContainer** item to the region.

Step 4: Set the following properties on the **richContainer** item:

Property Name	Type	Personalizable	Description
`Id`	String	No	Specify a unique identifier for the richContainer component.
`Content Type`	String	Yes	Specify the type of embedded content. Possible values are: • **flash** - for content developed using Adobe Flex builder, outputted as Flash (.swf) executable files. • **obiee** - for OBIEE analytics pages. For prerequisites and additional information on OBIEE integration, refer to the My Oracle Support (formerly Oracle*MetaLink*) Knowledge Document 974422.1, "Embedding Analytics in Oracle E-Business Suite". • **other** - for third-party Oracle-compatible widgets.

Destination Function	String	Yes	Specify the developer name of the FND function used to secure the embedded content or widget. The function should be defined as: • Function Type: WIDGET • Web HTML Call: <*relative path of the widget*>
Width	Number	Yes	Specify the width of the container region in pixels.
Height	Number	Yes	Specify the height of the container region in pixels. **Note:** If you set the content type as `FLASH_CONTENT`, then you must set the height of the container region to the height of the .swf Flash file's output stage.
Parameters	String	Yes	Specify parameters that can be passed to the embedded content as a list of comma-separated, name-value pairs. Dynamic parameters may be specified as `paramName={@viewAttr}`. **Note:** If the `Content Type` property is set to **other**, you must provide the accessorClass name as a parameter. The accessorClass is the class that extends the `oracle.apps.fnd.service.gwy.AbstractExternalAppAccessor` abstract class and implements the `makeAccessURL()`, `makeForwardURL()` and `isAuthorized()` methods. The URL to be embedded should be returned by `makeAccessURL()` and `isAuthorized()` should return **true**.
View Instance	String	Yes	If you pass dynamic parameters to the embedded content, then specify a view object name from which to select the dynamic parameter values.

Note: When you embed content using this component, you should also be sure to handle accessibility settings for the embedded content itself. For example, if the **Self Service Accessibility Features** profile option is set to **Standard Accessibility** for the OA Framework application, then you need to map this setting to a corresponding accessibility mode in the software used to render the embedded content. The Rich Content Container exposes the **Parameters** attribute which you can use to pass in any parameter. Use this attribute to pass in the accessibility setting and handle the accessibility in the embedded content layer. In the OA Framework application, you can get the accessibility mode by calling `pageContext.getProfile("ICX_ACCESSIBILITY_FEATURES")`.

Runtime Control

To add a Rich Content Container to a region programmatically:

```
import oracle.apps.fnd.framework.webui.beans.layout.OARichContainerBean
```

In the controller's `processRequest` method, create the `OARichContainerBean` object. For example, to create a `OARichContainerBean` object for a Rich Content Container to contain Flash content:

```
OARichContainerBean flexContainer =
  (OARichContainerBean)createWebBean(pageContext,RICH_CONTAINER_BEAN,null, "Flash");
```

Set the required properties for the content, using the method `setContentType` to specify the type of embedded content as follows:

- `FLASH_CONTENT` - for content developed using Adobe Flex builder, outputted as Flash (.swf) executable files.

  Note: If you set the content type as `FLASH_CONTENT`, then you must set the height of the container region to the height of the .swf Flash file's output stage.

- `OBIEE_CONTENT` - for OBIEE analytics pages. For prerequisites and additional information on OBIEE integration, refer to the My Oracle Support (formerly Oracle*MetaLink*) Knowledge Document 974422.1, "Embedding Analytics in Oracle E-Business Suite".
- `OTHER_CONTENT` - for third-party Oracle-compatible widgets.

The following example shows the required properties set for Flash content:

```
flexContainer.setFunctionName("FLEX_FUNCTION");
flexContainer.setContentType(FLASH_CONTENT);
flexContainer.setAttributeValue("width",200);
flexContainer.setAttributeValue("height",200);
```

If you need to pass variables, set them as follows in single string. For example, if FlashVars are required on the Flex component:

```
flexContainer.setParameters("param1=value1 & param2=value2");
```

Add the Rich Content Container as an indexed child to the region that should display the rich content. For example:

```
webBean.addIndexedChild(flexContainer);
```

Internationalization and Localization

All features of this component have been considered for internationalization given that they display prompts, data and messages. All prompts are either stored as translatable attributes in the page metadata, or shall come from the FND Message Dictionary. For example, the Title attribute of the Rich Content Container component is a translatable attribute on the Rich Content Container component metadata. All messages used for the new Rich Content Container interactions also come from the FND Message Dictionary.

Personalization Considerations

- None

Known Issues

- Rich containers cannot be used in email facets. Inline frames are often used to insert advertisements, video, audio, or forms in other websites. IFrames often contain scripts, so most email clients block them. Because they are implemented with HTML inline IFrame elements, rich containers will not be displayed correctly in email.

Related Information

- BLAF UI Guidelines
  - None
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OARichContainerBean`
- ToolBox Tutorial / Sample Library
  - None
- OA Framework Component Reference (Oracle JDeveloper 10*g* Online Help)
  - None

Rich Interactions

Overview

Oracle E-Business Suite Release 12.2.3 introduced a set of Rich Interactions that improve the look and feel of classic and advanced tables. In Release 12.2.4, support for some of these features was extended to HGrids.

In Release 12.2.9, Rich Interactions support has been extended to tables in the following categories:

- Table with Descriptive Flexfield rendered as Popup
- Table with Key Flexfield that has combinations UI display
- Table with record history enabled

Also, some previous restrictions on use of Rich Interactions on certain types of table have been removed in Release 12.2.9. Such changes are noted in the text, under the relevant lists of the form 'Tables and Columns That Do Not Support...'

The rich features include:

- **Column Resizing** - You can instantly increase or decrease the width of a table column to improve data visibility. If, upon resizing, the table width extends beyond the width of the browser, then a horizontal scroll bar appears under the table.
- **Column Reordering** - You can instantly reorder the rendering of visible columns in a table.
- **Horizontal Scrolling** - If the contents of a table exceeds the browser width, a horizontal scroll bar displays under the table to let you see the remaining table data.
- **Column Hide/Show** - You can instantly hide or show the columns in a table.
- **Table Detaching** - You can detach a table from the base page to view as a separate component. You can drag the detached table around the browser and work on the table directly as you would from its base page. You can also move quickly up and down, as well as left and right between the table's form fields.
- **Reset to Default** - You can delete all user-personalizations made via these rich features and reset the table back to its original definition.
- **Table Refresh** - You can refresh the table by re querying for the latest table records in case modifications have been simultaneously made elsewhere.
- **Vertical Scrolling** - You can scroll through the rows of a table using a vertical scroll bar instead of selecting the **Next** and **Previous** links.
- **Column Freeze** - You can keep a set of columns in view while scrolling a table horizontally.

Important: The rich features are supported only for certain browsers. For more information, refer to My Oracle Support Knowledge Document 389422.1, *Recommended Browsers for Oracle E-Business Suite Release 12*.

Note: For Release 12.2.9, the table toolbar has been significantly reorganized. If desired, the user can revert to the previous behavior by setting a profile option.

Contents

- Enabling Rich Interactions
- Rich Features in Detail
  - Column Resizing
  - Column Reordering
  - Horizontal Scrolling
  - Table Detaching
  - Column Hide/Show
  - Reset to Default
  - Table Refresh
  - Vertical Scrolling
  - Column Freeze
- Disabling All Rich Table Interactions
- Support for Touch Devices
- Debugging Rich Interaction Issues
- Personalization Considerations
- Known Issues
- Support Matrix
- Related Information

Enabling Rich Interactions

You can enable or disable all rich features by setting the profile option **FND: Enable Rich Table Interactions** to **True** (enable) or **False** (disable), respectively at the Site, Application and/or User level. Setting the profile value to null enables rich interactions. This profile is set to **True** by default.

Note: As a prerequisite for export to Excel or PDF, the **FND: Enable Rich Table Interactions** profile option must be enabled.

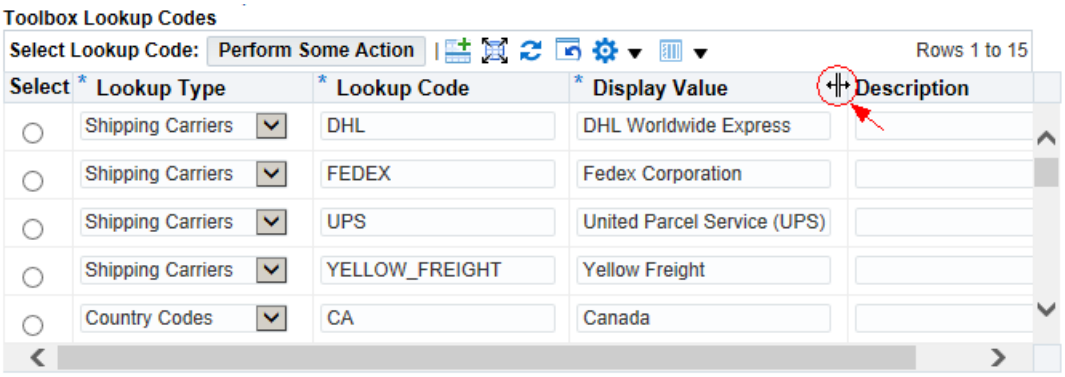
Rich Features in Detail

This section lists each rich feature and discusses the implementation and support of each feature in detail.

Column Resizing

The column resize cursor appears when you hover your mouse over the right edge of a column heading.

Note: The column resize cursor appears over the right edge of a column heading only if the user's language preference is set to English.



Drag the resize cursor left or right to increase or decrease a column to the desired width, respectively. The input fields under the column heading resize in accordance with the new column heading width. For an advanced table, you can resize a column group which simply resizes just the last column of the column group or you can resize any of the individual columns of the column group.

Column Resizing Implementation and Support

- Declarative Implementation
- Runtime Control
- Touch Devices Support
- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Column Resize
- Usage Restrictions
- Behavior in Query Bean With Result Tables

Declarative Implementation

Column resizing is enabled by default in OA Framework-based tables. To turn off this feature for a table, use the Oracle JDeveloper OA Extension Property Inspector to set the `Enable Column Resize` property on the table or advanced table region to **False**.

Note: Turning off column resize in this way disables the feature for all users.

Important: A value of **False** for the profile option **FND: Enable Rich Table Interactions** overrides the `Enable Column Resize` property value. For example, if the profile **FND: Enable Rich Table Interactions** is set to **False** at the Site, Application or User level, then column resizing is disabled even if the `Enable Column Resize` property for a table is set to **True**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. You cannot personalize, reuse, or easily extend programmatically created web beans. For example, if you add any column programmatically to a (classic or advanced) table web bean or create the table programmatically, the column resize functionality will be unavailable for the whole table as there is no personalize view support for programmatic tables or for columns added programmatically to a metadata-based table.

To enable or disable column resizing, you can use the following APIs on `OATableBean`, `OAAdvancedTableBean`, or `OAHGridBean`:

```
public void setColumnResizeEnabled(boolean colResizeEnabled)
public boolean isColumnResizeEnabled()
```

For example, to programmatically turn off column resizing you can call the API as follows:

```
table.setColumnResizeEnabled(false);
```

Note: By default, you can only resize a column's width to a minimum of its header text length to prevent hiding its header text. If you need to override this behavior to make the column width less than the width of its header text, you can set the following attributes on the tableBean:

```
tableBean.setAttributeValue(COL_RESIZE_MODE,MIN_LENGTH_MODE);
tableBean.setAttributeValue(RESIZE_MIN_LENGTH,/*Integer value*/);
```

Express the second argument of the second attribute as the minimum number of pixels to which a column can be resized.

Note: Resizing a column is considered a user personalization that is saved as a view on the database. Once you stop resizing a column, a REST call fires to update the back end view with information related to the width of the column. The view is created for each user. This information is then used to render the column in the next PPR cycle or full page mode cycle.

Touch Devices Support

As of Release 12.2.4, OA Framework provides gestures support for rich table interactions. For touch devices, you may horizontally Pan on a column header to resize a column.

Accessible Modes Support

Column resize is not supported if you set the **Self Service Accessibility Features** profile to **Standard Accessibility** or **Screen Reader Optimized**.

HGrids Support

While support for the column resize feature is extended to HGrids as of Release 12.2.4, the hierarchy column may not be resized.

Personalization Support

When a user resizes columns to quickly personalize a table, the personalization is saved as a user-level view of the table in the MDS repository. You may examine or delete the personalization metadata in the MDS repository using the `JDR_UTILS` package.

See a summary of Rich Table personalization considerations in the Oracle Application Framework Personalization Guide.

Tables and Columns That Do Not Support Column Resize

- Tables created programmatically
- Tables in which one or more columns are added programmatically
- Special table columns such as Details, Select or row header stamp columns
- Special columns such as selection, focus and object hierarchy columns in an HGrid
- All tables and HGrids when the profile **Disable Self Service Personal** is set to **Yes**
- LOV tables
- Inner tables
- Printable page
- Email facet
- Tables with record history enabled
- Attachment tables
- Tables that contain a key or descriptive flexfield
- Tables inside an embedded or parameterized pop-up
- OA Framework-based tables embedded in a JTT skin
- All tables and HGrids, when column resize is turned off through its meta data property or controller code
- Tables rendered in portlets
- OA Framework pages containing tables that may be accessed via a Guest user login, such as the Supplier Registration page

Note: From Release 12.2.9, all these tables support Rich Interactions.

Usage Restrictions

- If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.
- Assign each item/column in a page a unique ID (which also adheres to the HTML standards) to avoid impacting the rich table interactions on the page.
- To ensure that tables function properly when you enable rich table interactions, follow the classic or advanced table implementation and coding standards carefully. For example, when you implement an advanced table declaratively, you can not add an item as a direct child of the advanced table, you may only add an item to a column or column group of the advanced table. Similarly, you should follow this standard when you add items programmatically to an advanced table.
- If a Query Bean configuration contains multiple tables/advanced tables as its children, column resize is supported only in the first table. As a result we recommend you disable column resize altogether in this scenario.

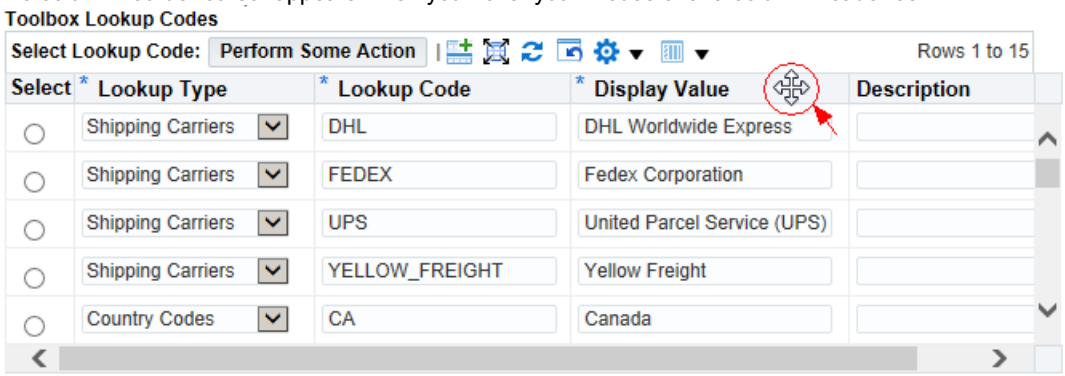
Behavior in Query Bean With Result Tables

The column resize feature behaves as follows in the result table that appears in each of the query bean panels.

- **Simple Search Panel**
  - Column resize is allowed.
  - Resize changes go to an OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - If there is no default view associated with the query region then <*TableName*>**_UserView becomes the default view.
  - Whenever a user returns back to the page with the Simple Search panel, the changes made in <*TableName*>**_UserView is applied.
  - The view <*TableName*>**_UserView is shared between Simple Search and Advanced Search panels.
- **Advanced Search Panel**
  - Column resize is allowed.
  - Resize changes go to an OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - If there is no default view associated with the query region then <*TableName*>**_UserView becomes the default view.
  - Whenever a user returns back to the page with the Advanced Search panel, the changes made in <*TableName*>**_UserView is applied.
  - The view <*TableName*>**_UserView is shared between Simple Search and Advanced Search panels.
- **Views Panel**
  - This panel will list <*TableName*>**_UserView in the Views drop down list if the user previously personalized the table using any rich interaction in the Simple Search and Advanced Search panels.
  - In the case of a seeded default view, the view is non-editable, so column resize is disabled.
  - Column resize is enabled in all views except seeded views.
  - If <*TableName*>**_UserView is selected from the Views drop down list and a user performs further changes to this view, then the changes are also reflected in the Simple and Advanced Search panels since this view is used for rendering tables in the Simple and Advanced Search panels.

Column Reordering

The column reorder cursor appears when you hover your mouse over a column header cell.



Select this column reorder cursor and then drag the column header to drop off the column to a new desirable location within the table. You may also reorder columns using the **Table Settings** icon that appears in the table's control bar.

Column Reordering Implementation and Support

- Declarative Implementation
- Runtime Control

- Touch Devices Support
- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Column Reorder
- Usage Restrictions
- Behavior in Query Bean With Result Tables

Declarative Implementation

Column reordering is enabled by default in OA Framework-based tables. To turn off this feature for a table, use the Oracle JDeveloper OA Extension Property Inspector to set the `Enable Column Reorder` property on the table or advanced table region to **False**.

Note: Turning off column reorder in this way disables the feature for all users.

Important: A value of **False** for the profile option **FND: Enable Rich Table Interactions** overrides the `Enable Column Reorder` property value. For example, if the profile **FND: Enable Rich Table Interactions** is set to **False** at the Site, Application or User level, then column reordering is disabled even if the `Enable Column Reorder` property for a table is set to **True**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. You cannot personalize, reuse, or easily extend programmatically created web beans. For example, if you add any column programmatically to a (classic or advanced) table web bean or create the table programmatically, the column resize functionality will be unavailable for the whole table as there is no personalize view support for programmatic tables or for columns added programmatically to a metadata-based table.

To enable or disable column reordering, you can use the following APIs on `OATableBean`, `OAAdvancedTableBean`, or `OAHGridBean`:

```
public void setColumnReorderEnabled(boolean colReorderEnabled)
public boolean isColumnReorderEnabled()
```

For example, to programmatically turn off column reordering you can call the API as follows:

```
table.setColumnReorderEnabled(false);
```

Note: Reordering a column is considered a user personalization that is saved as a view on the database. Once you stop reordering a column, a REST call fires to update the back end view with information related to the order of the column. The view is created for each user. This information is then used to render the column in the next PPR cycle or full page mode cycle.

Touch Devices Support

As of Release 12.2.4, OA Framework provides gestures support for rich table interactions. For touch devices, you may horizontally Pan on a column header to reorder a column.

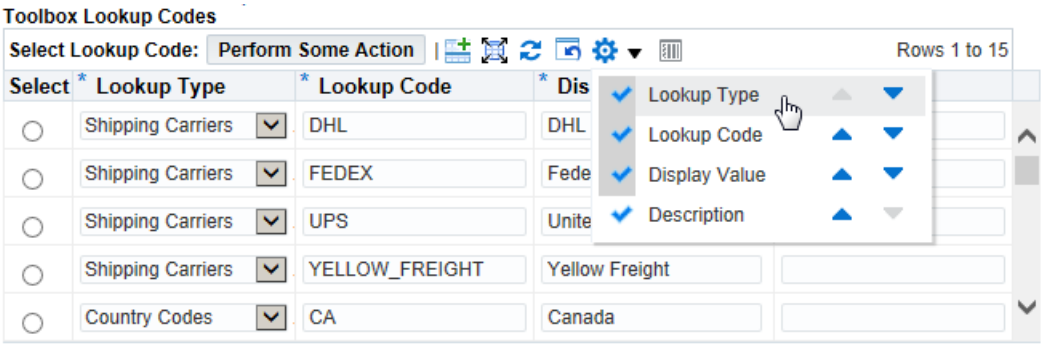
Accessible Modes Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still take advantage of column reordering through a simpler user interface that does not require a mouse.

Use the [Tab] key or [Shift][Tab] key combination to move the cursor focus to the **Table Settings** icon in the control bar.

1. Press **[Enter]** to display a menu that lists the columns of the table.
2. Press **[Tab]** to move the focus to the column names in the menu.
3. Use the **[Up Arrow]** or **[Down Arrow]** keys to navigate between column names, "wrapping" at the top and bottom of the menu.
4. With focus on a column name, press **[Ctrl][Up]** to move the column up the menu list, keeping focus on the moved column.
5. With focus on a menu item, press **[Ctrl][Down]** to move the column down the menu list, keeping focus on the moved column.
6. With the focus on a column name, you can apply the change by pressing any of the following:
    - **[Esc]** which also closes the menu and returns focus to the menu-enabled item.
    - **[Tab]** which also closes the menu and navigates to the next item on the page.
    - **[Tab]** or **[Shift][Tab]** which also closes the menu and moves focus back to the menu-enabled item.

Beginning in Release 12.2.4, if you select the **Table Settings** icon that appears in the table's control bar, a menu displays, listing the columns of the table. Alongside each column is an **[Up Arrow]** and a **[Down Arrow]** reorder icon that you can select to rearrange the order of the column. Select any area outside the menu to close the menu and apply the changes to the table.



For more information about user interactions of menu-enabled items such as the **Table Settings** icon, refer to the User Interactions section of the Menu Component topic.

HGrids Support

While support for the column reorder feature is extended to HGrids as of Release 12.2.4, the hierarchy column may not be reordered.

Personalization Support

When a user reorders columns to quickly personalize a table, the personalization is saved as a user-level view of the table in the MDS repository. You may examine or delete the personalization metadata in the MDS repository using the `JDR_UTILS` package.

See a summary of Rich Table personalization considerations in the Oracle Application Framework Personalization Guide.

Tables and Columns That Do Not Support Column Reorder

- Tables created programmatically
- Tables in which one or more columns are added programmatically
- Special table columns such as Details, Select or row header stamp columns
- Special columns such as selection, focus and object hierarchy columns in an HGrid
- All tables and HGrids when the profile **Disable Self Service Personal** is set to **Yes**
- LOV tables
- Inner tables
- Printable page - size of columns, arranged by the user in the previous page from which the printable page is launched, is preserved. No resizing is allowed in the UI in the printable page.
- Email facet - size of columns, arranged by the user in the previous page from which the Email facet page is launched, is preserved. No resizing is allowed in the UI in the Email facet.
- Tables with record history enabled
- Attachment tables
- Tables that contain a key or descriptive flexfield
- Tables inside an embedded or parameterized pop-up
- OA Framework-based tables embedded in a JTT skin
- All tables and HGrids, when column reorder is turned off through its meta data property or controller code
- Tables rendered in portlets
- OA Framework pages containing tables that may be accessed via a Guest user login, such as the Supplier Registration page

Note: From Release 12.2.9, all these tables support Rich Interactions.

Usage Restrictions

- If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.
- Assign each item/column in a page a unique ID (which also adheres to the HTML standards) to avoid impacting the rich table interactions on the page.
- To ensure that tables function properly when you enable rich table interactions, follow the classic or advanced table implementation and coding standards carefully. For example, when you implement an advanced table declaratively, you can not add an item as a direct child of the advanced table, you may only add an item to a column or column group of the advanced table. Similarly, you should follow this standard when you add items programmatically to an advanced table.

Behavior in Query Bean With Result Tables

The column reorder feature behaves as follows in the result table that appears in each of the query bean panels.

- **Simple Search Panel**
  - Column reorder is allowed.
  - Reorder changes go to an OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - If there is no default view associated with the query region then <*TableName*>**_UserView becomes the default view.
  - Whenever a user returns back to the page with the Simple Search panel, the changes made in <*TableName*>**_UserView is applied.
  - The view <*TableName*>**_UserView is shared between Simple Search and Advanced Search panels.
- **Advanced Search Panel**
  - Column reorder is allowed.
  - Reorder changes go to an OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - If there is no default view associated with the query region then <*TableName*>**_UserView becomes the default view.
  - Whenever a user returns back to the page with the Advanced Search panel, the changes made in <*TableName*>**_UserView is applied.
  - The view <*TableName*>**_UserView is shared between Simple Search and Advanced Search panels.
- **Views Panel**
  - This panel will list <*TableName*>**_UserView in the Views drop down list if the user previously personalized the table using any rich interaction in the Simple Search and Advanced Search panels.
  - In the case of a seeded default view, the view is non-editable, so column reorder is disabled.
  - Column reorder is enabled in all views except seeded views.
  - If <*TableName*>**_UserView is selected from the Views drop down list and a user performs further changes to this view, then the changes are also reflected in the Simple and Advanced Search panels since this view is used for rendering tables in the Simple and Advanced Search panels.

Note: If a Query Bean configuration contains multiple tables/advanced tables as its children, column reorder is supported only in the first table. As a result we recommend you disable column reorder altogether in this scenario.

Horizontal Scrolling

If a table's width exceeds the browser's width, a horizontal scroll bar appears beneath the table so you can scroll through the table to view its complete content. A horizontal scroll bar also appears if column resizing is enabled and you resize the table's column(s) such that the table's width now exceeds the browser's width.

Horizontal Scrolling Implementation and Support

- Declarative Implementation
- Runtime Control
- Touch Devices Support
- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Horizontal Scroll
- Usage Restrictions

Declarative Implementation

Horizontal scrolling is enabled by default in OA Framework-based tables. To turn off this feature for a table, use the Oracle JDeveloper OA Extension Property Inspector to set the `Enable Horizontal Scroll` property on the table or advanced table region to **False**.

Note: Turning off horizontal scrolling in this way disables the feature for all users.

Important: A value of **False** for the profile option **FND: Enable Rich Table Interactions** overrides the `Enable Horizontal Scroll` property value. For example, if the profile **FND: Enable Rich Table Interactions** is set to **False** at the Site, Application or User level, then horizontal scrolling is disabled even if the `Enable Horizontal Scroll` property for a table is set to **True**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. You cannot personalize, reuse, or easily extend programmatically created web beans. For example, if you add any column programmatically to a (classic or advanced) table web bean or create the table programmatically, the column resize functionality will be unavailable for the whole table as there is no personalize view support for programmatic tables or for columns added programmatically to a metadata-based table.

To enable or disable horizontal scrolling, you can use the following APIs on `OATableBean`, `OAAdvancedTableBean`, or `OAHGridBean`:

```
public void setHorizontalScrollEnabled(boolean horizontalScrollEnabled)
public boolean isHorizontalScrollEnabled()
```

For example, to programmatically turn off horizontal scrolling, you can call the API as follows:

```
table.setHorizontalScrollEnabled(false);
```

Bound Container For Scroll Bar

By default, OA Framework treats the browser as a container for the tables. If you need to render multiple large tables (that have more columns than can render in the width of the browser), you can use the attribute `BOUND_PARENT_CONTAINER_ID` to specify the web bean from which to derive the width for rendering the table.

For example, suppose you have a row layout and 2 cell formats where each cell format renders a table. If both tables are extremely large with many columns, then by default, OA Framework renders the first table fully within the browser, and pushes the second table beyond the width of the browser.

To avoid this outcome, set the `BOUND_PARENT_CONTAINER_ID` attribute to the cell format web bean and set the width of each of the cell format web beans to 50% or less so that the two tables render side by side with scroll bars for each, as shown below:

```
rowlayout
  --cellFormat1 width = 50%
      --table1
  --spacer width =10%
  --cellFormat2 width = 40%
      --table2

//for table1, we need to set
table1.setAttributeValue(BOUND_PARENT_CONTAINER_ID,"cellFormat1");
//for table2, we need to set
table2.setAttributeValue(BOUND_PARENT_CONTAINER_ID,"cellFormat2");
```

Touch Devices Support

In touch devices, use the method that is native to the specific device to horizontally scroll through the table.

Accessible Modes Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still take advantage of horizontal scrolling through a simpler user interface that does not require a mouse.

Use the **[Left Arrow]** or **[Right Arrow]** keys to scroll a table horizontally.

HGrids Support

Support for the horizontal scrolling feature is extended to HGrids as of Release 12.2.4.

Personalization Support

User personalization support is not applicable for this feature.

To personalize the enabling or disabling of this feature as an admin-level personalization, see a summary of Rich Table personalization considerations in the Oracle Application Framework Personalization Guide.

Tables and Columns That Do Not Support Horizontal Scroll

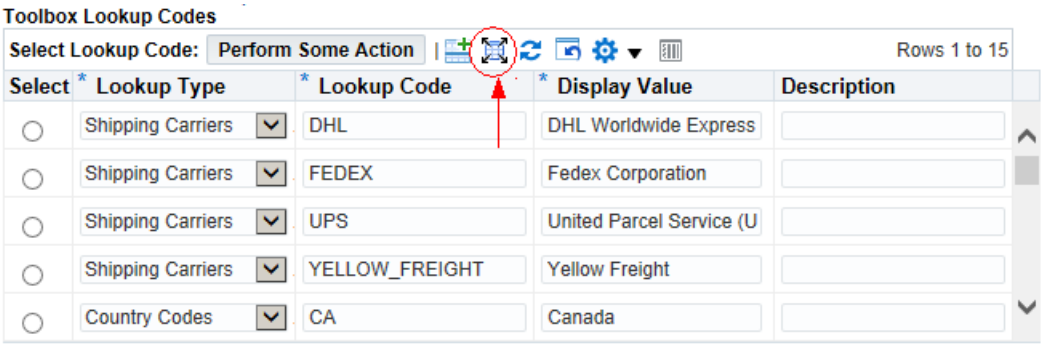
- LOV tables
- Inner tables
- Printable page
- Email facet
- Attachment tables
- Tables inside an embedded or parameterized pop-up
- OA Framework-based tables embedded in a JTT skin
- All tables and HGrids, when horizontal scrolling is turned off through its meta data property or controller code
- Tables rendered in portlets

Usage Restrictions

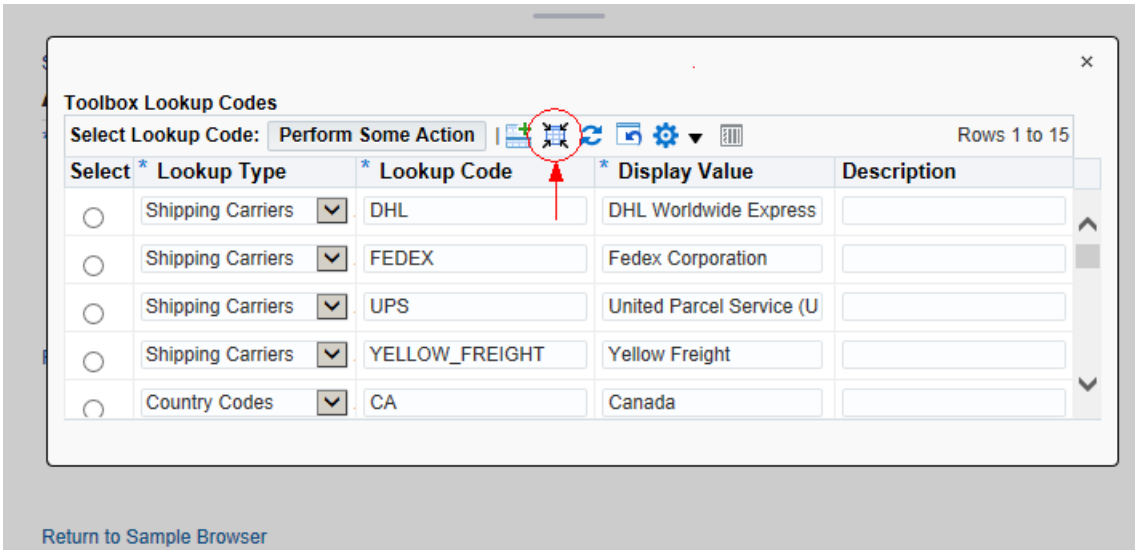
- If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.

Table Detaching

The table control bar now displays a **Detach Table** icon.



When you select the **Detach Table** icon, the table itself opens in a separate window that displays more records. The detached table displays at most, twice the number of rows as in the base page table as per the screen size of the display.



To move the cursor up or down between form fields of the same column, use the **[Alt]** + **[Up]** or **[Alt]** + **[Down]** key combinations, respectively. To move the cursor forwards or backwards between form fields, use the **[Tab]** or **[Shift]** + **[Tab]** key combination, respectively.

You can drag the detached table in and around the browser to move it to a new location. To reattach the table to its base page, select the **Attach Table** icon or the **Close** icon.

Table Detaching Implementation and Support

- Declarative Implementation
- Runtime Control
- Touch Devices Support
- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Table Detach
- Usage Restrictions
- Behavior in Query Bean With Result Tables

Declarative Implementation

Table detaching is enabled by default in OA Framework-based tables. To turn off this feature for a table, use the Oracle JDeveloper OA Extension Property Inspector to set the `Enable Detach` property on the table or advanced table region to **False**.

Note: Turning off table detach in this way disables the feature for all users.

Important: A value of **False** for the profile option **FND: Enable Rich Table Interactions** overrides the `Enable Detach` property value. For example, if the profile **FND: Enable Rich Table Interactions** is set to **False** at the Site, Application or User level, then table detaching is disabled even if the `Enable Detach` property for a table is set to **True**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. You cannot personalize, reuse, or easily extend programmatically created web beans. For example, if you add any column programmatically to a (classic or advanced) table web bean or create the table programmatically, the column resize functionality will be unavailable for the whole table as there is no personalize view support for programmatic tables or for columns added programmatically to a metadata-based table.

To enable or disable table detaching, you can use the following APIs on `OATableBean` or `OAAdvancedTableBean`:

```
public void setDetachEnabled(boolean detachEnabled)
public boolean isDetachEnabled()
```

For example, to programmatically turn off the ability to detach a table, you can call the API as follows:

```
table.setDetachEnabled(false);
```

Touch Devices Support

As of Release 12.2.4, OA Framework provides gestures support for rich table interactions. For touch devices, you may Single Tap on the **Detach Table**/**Attach Table** icons to detach or attach a table.

Accessible Modes Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still take advantage of table detaching through a simpler user interface that does not require a mouse.

1. Use the **[Tab]** key or **[Shift][Tab]** key combination to move the cursor focus to the **Detach Table** icon in the control bar.
2. Once you detach the table, you can interact with the table fields and records.
3. Use the **[Tab]** key or **[Shift][Tab]** key combination to move the cursor focus to the **Attach Table** or **Close** icon and press **[Enter]** to reattach the table to its base page.

HGrids Support

Table detaching is not supported in HGrids.

Personalization Support

User personalization support is not applicable for this feature.

To personalize the enabling or disabling of this feature as an admin-level personalization, see a summary of Rich Table personalization considerations in the Oracle Application Framework Personalization Guide.

Tables and Columns That Do Not Support Table Detach

- LOV tables
- Inner tables
- Printable page
- Email facet
- Attachment tables
- Tables inside an embedded or parameterized pop-up
- OA Framework-based tables embedded in a JTT skin
- All tables and HGrids, when column reorder is turned off through its meta data property or controller code
- Tables rendered in portlets

Usage Restrictions

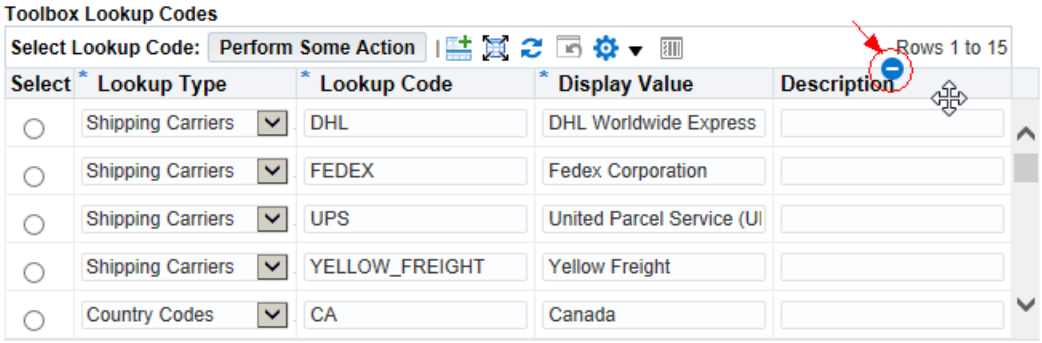
- If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.

Behavior in Query Bean With Result Tables

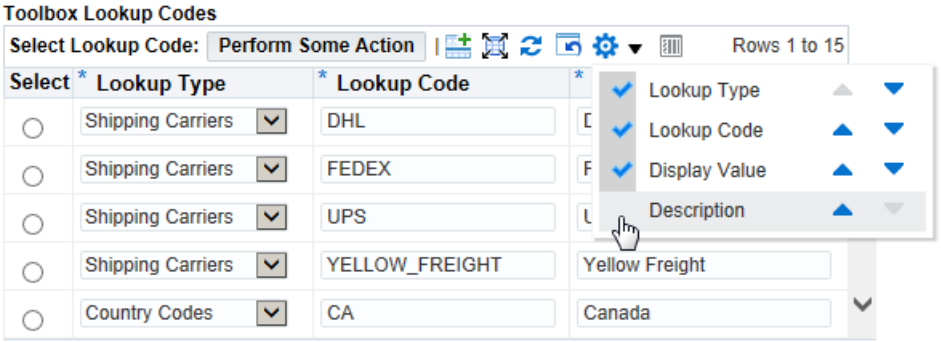
Since Detach Table does not save any user personalization data, the **Detach Table** icon and functionality is available in the Simple Search, Advanced Search, and Views panels.

Column Hide/Show

Beginning in Release 12.2.4, a column **Hide** icon appears when you hover your mouse over the center of a column or column group header cell. Select the **Hide** icon to hide the column in the table.



To show the hidden column, select the **Table Settings** icon that appears in the table's control bar. A menu displays, listing the columns of the table. Select the hidden column to toggle the display of the Check icon next to the column name. Select any area outside the menu to close the menu and apply the changes to the table. Note that the **Table Settings** menu serves to provide support for accessible Modes as well.



Column Hide/Show Implementation and Support

- Declarative Implementation
- Runtime Control
- Touch Devices Support
- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Column Hide/Show
- Usage Restrictions
- Behavior in Query Bean With Result Tables

Declarative Implementation

Column Hide/Show is enabled by default in OA Framework-based tables. To turn off this feature for a table, use the Oracle JDeveloper OA Extension Property Inspector to set the `Enable Column Hide Show` property on the table or advanced table region to **False**.

Note: Turning off column Hide/Show in this way disables the feature for all users.

Important: A value of **False** for the profile option **FND: Enable Rich Table Interactions** overrides the `Enable Column Hide Show` property value. For example, if the profile **FND: Enable Rich Table Interactions** is set to **False** at the Site, Application or User level, then column hide/show is disabled even if the `Enable Column Hide Show` property for a table is set to **True**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. You cannot personalize,

reuse, or easily extend programmatically created web beans. For example, if you add any column programmatically to a (classic or advanced) table web bean or create the table programmatically, the column resize functionality will be unavailable for the whole table as there is no personalize view support for programmatic tables or for columns added programmatically to a metadata-based table.

To enable or disable column Hide/Show, you can use the following APIs on `OATableBean`, `OAAdvancedTableBean`, or `OAHGridBean`:

```
public void setColumnHideShowEnabled(boolean colHideShowEnabled)
public boolean isColumnHideShowEnabled()
```

For example, to programmatically turn off column Hide/Show you can call the API as follows:

```
table.setColumnHideShowEnabled(false);
```

Touch Devices Support

As of Release 12.2.4, OA Framework provides gestures support for rich table interactions. For touch devices, you may **Hold** on a column header to reveal the hide/show icon for a column.

Accessible Modes Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still take advantage of column Hide/Show through a simpler user interface that does not require a mouse.

Use the **[Tab]** key or **[Shift][Tab]** key combination to move the cursor focus to the **Table Settings** icon in the control bar.

1. Press **[Enter]** to display a menu that lists the columns of the table.
2. Press **[Tab]** to move the focus to the column names in the menu.
3. Use the **[Up Arrow]** or **[Down Arrow]** keys to navigate between column names, "wrapping" at the top and bottom of the menu.
4. With the focus on a column name, press **[Space]** to check or uncheck the focused column name. A check shows the column.
5. With the focus on a column name, you can apply the change by pressing any of the following:
    - **[Esc]** which also closes the menu and returns focus to the menu-enabled item.
    - **[Tab]** which also closes the menu and navigates to the next item on the page.
    - **[Shift][Tab]** which also closes the menu and moves focus back to the menu-enabled item.

For more information about user interactions of menu-enabled items such as the **Table Settings** icon, refer to the User Interactions section of the Menu Component topic.

HGrids Support

Support for the column Hide/Show feature is extended to HGrids as of Release 12.2.4.

Personalization Support

When a user hide or show columns to quickly personalize a table, the personalization is saved as a user-level view of the table in the MDS repository. You may examine or delete the personalization metadata in the MDS repository using the `JDR_UTILS` package.

See a summary of Rich Table personalization considerations in the Oracle Application Framework Personalization Guide.

Tables and Columns That Do Not Support Column Hide/Show

- Tables created programmatically
- Tables in which one or more columns are added programmatically
- Special table columns such as Details, Select or row header stamp columns
- Special columns such as selection, focus and object hierarchy columns in an HGrid
- All tables and HGrids when the profile **Disable Self Service Personal** is set to **Yes**
- LOV tables
- Inner tables
- Printable page
- Email facet
- Tables with record history enabled
- Attachment tables
- Tables that contain a key or descriptive flexfield
- Tables inside an embedded or parameterized pop-up
- OA Framework-based tables embedded in a JTT skin
- All tables and HGrids, when column reorder is turned off through its meta data property or controller code
- Tables rendered in portlets
- OA Framework pages containing tables that may be accessed via a Guest user login, such as the Supplier Registration page

Note: From Release 12.2.9, all these tables support Rich Interactions.

Usage Restrictions

- If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.
- Assign each item/column in a page a unique ID (which also adheres to the HTML standards) to avoid impacting the rich table interactions on the page.
- To ensure that tables function properly when you enable rich table interactions, follow the classic or advanced table implementation and coding standards carefully. For example, when you implement an advanced table declaratively, you can not add an item as a direct child of the advanced table, you may only add an item to a column or column group of the advanced table. Similarly, you should follow this standard when you add items programmatically to an advanced table.

Behavior in Query Bean With Result Tables

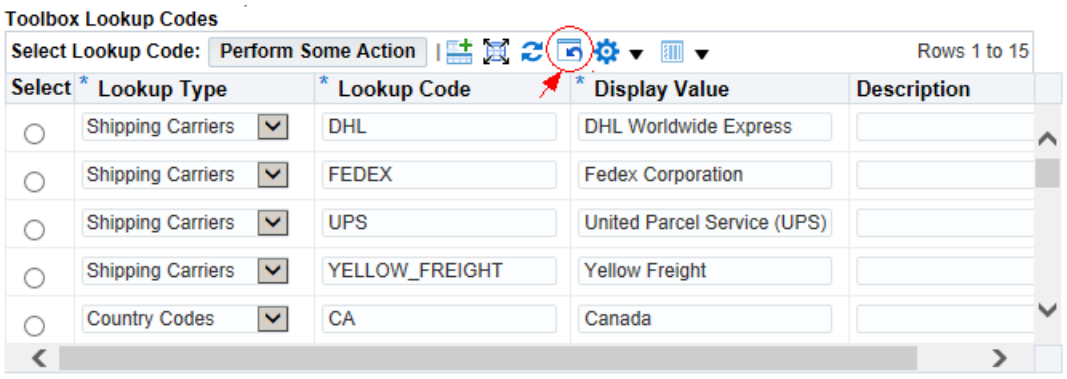
The column hide/show feature behaves as follows in the result table that appears in each of the query bean panels.

- **Simple Search Panel**
  - Column hide/show is allowed.
  - Hide/show changes go to an OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - If there is no default view associated with the query region then <*TableName*>**_UserView becomes the default view.
  - Whenever a user returns back to the page with the Simple Search panel, the changes made in <*TableName*>**_UserView is applied.
  - The view <*TableName*>**_UserView is shared between Simple Search and Advanced Search panels.
- **Advanced Search Panel**
  - Column hide/show is allowed.
  - Hide/show changes go to an OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - If there is no default view associated with the query region then <*TableName*>**_UserView becomes the default view.
  - Whenever a user returns back to the page with the Advanced Search panel, the changes made in <*TableName*>**_UserView is applied.
  - The view <*TableName*>**_UserView is shared between Simple Search and Advanced Search panels.
- **Views Panel**
  - This panel will list <*TableName*>**_UserView in the Views drop down list if the user previously personalized the table using any rich interaction in the Simple Search and Advanced Search panels.
  - In the case of a seeded default view, the view is non-editable, so column resize is disabled.
  - Column resize is enabled in all views except seeded views.
  - If <*TableName*>**_UserView is selected from the Views drop down list and a user performs further changes to this view, then the changes are also reflected in the Simple and Advanced Search panels since this view is used for rendering tables in the Simple and Advanced Search panels.

Note: If a Query Bean configuration contains multiple tables/advanced tables as its children, column hide/show is supported only in the first table. As a result we recommend you disable column hide/show altogether in this scenario.

Reset to Default

As of Release 12.2.4, the control bar displays a new **Reset to Default** icon. A user may select this icon to revert the table back to its original definition by deleting all user-personalized views of the table, such as column resizing, column reordering and/or column hide/show. The **Reset to Default** icon is disabled if the table has no user changes to reset.



Note: In the case of a query region that displays a Simple Search, Advanced Search, or (user-personalizable) Views panel, if a user navigates to the Views panel, the Reset to Default icon does not appear.

Reset to Default Implementation and Support

- Declarative Implementation
- Runtime Control
- Touch Devices Support
- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Reset to Default
- Usage Restrictions
- Behavior in Query Bean With Result Tables

Declarative Implementation

None

Runtime Control

None

Touch Devices Support

As of Release 12.2.4, OA Framework provides gestures support for rich table interactions. For touch devices, you may tap on the **Reset to Default** icon to revert the table back to its original definition.

Accessible Modes Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still take advantage of reset to default through a simpler user interface that does not require a mouse.

Use the **[Tab]** key or **[Shift][Tab]** key combination to move the cursor focus to the **Reset to Default** icon in the control bar.

HGrids Support

Support for the Reset to Default feature is extended to HGrids as of Release 12.2.4.

Personalization Support

User and admin-level personalization support are not applicable for this feature.

Tables and Columns That Do Not Support Reset to Default

If all four of the rich interaction features, resize column, reorder column, column Hide/Show, and column freeze, are disabled for a table, then the **Reset to Default** icon does not appear. If any one of these features is enabled on a table, then the **Reset to Default** icon displays.

Usage Restrictions

If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.

Behavior in Query Bean With Result Tables

The reset to default feature behaves as follows in the result table that appears in each of the query bean panels.

- **Simple Search Panel**
  - Reset to default is allowed.
  - Reset to default deletes the OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - As the view <*TableName*>**_UserView is shared between the Simple and Advanced Search panels, resetting the view to default in one panel also resets the view to default in the other panel.
  - In addition, the Views drop down list in the Views Panel will no longer display the view <*TableName*>**_UserView.
- **Advanced Search Panel**

    - Reset to default is allowed.
    - Reset to default deletes the OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
    - As the view <*TableName*>**_UserView is shared between the Simple and Advanced Search panels, resetting the view to default in one panel also resets the view to default in the other panel.
    - In addition, the Views drop down list in the Views Panel will no longer display the view <*TableName*>**_UserView.
- **Views Panel**
    - The **Reset to Default** icon is not be seen in the User Interface.
    - Since there is a Personalize Views page and the view <*TableName*>**_UserView can be deleted from that page, there is no **Reset to Default** icon available in the Views panel page.
    - To delete the user personalization data, a user needs to navigate to the Personalize Views page and select the view <*TableName*>**_UserView and choose **Delete**.

Note: If a Query Bean configuration contains multiple tables/advanced tables as its children, reset to default is supported only in the first table. As a result we recommend you disable reset to default altogether in this scenario.

Table Refresh

As of Release 12.2.4, the table control bar displays a **Refresh** Icon. Select **Refresh** to requery the table for any modifications made simultaneously to the table elsewhere by another user. Refer to the Runtime Control section if you wish to disable this feature.

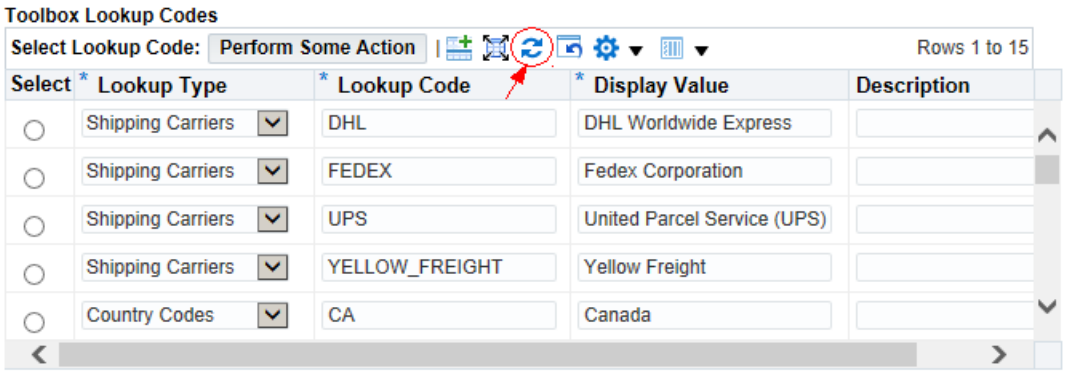


Table Refresh Implementation and Support

- Declarative Implementation
- Runtime Control
- Touch Devices Support
- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Table Refresh
- Usage Restrictions
- Behavior in Query Bean With Result Tables

Declarative Implementation

None

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. You cannot personalize, reuse, or easily extend programmatically created web beans. For example, if you add any column programmatically to a (classic or advanced) table web bean or create the table programmatically, the column resize functionality will be unavailable for the whole table as there is no personalize view support for programmatic tables or for columns added programmatically to a metadata-based table.

To enable or disable table refresh, you can use the following APIs on `OATableBean` or `OAAdvancedTableBean`:

```
public void setRefreshEnabled(boolean tableRefreshEnabled)
public boolean isRefreshEnabled()
```

For example, to programmatically turn off the ability to refresh a table, you can call the API as follows:

```
table.setRefreshEnabled(false);
```

Touch Devices Support

As of Release 12.2.4, OA Framework provides gestures support for rich table interactions. For touch devices, you may tap on the **Refresh** icon to equerry the table for any modifications made simultaneously to the table elsewhere by another user.

Accessible Modes Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still take advantage of table refresh through a simpler user interface that does not require a mouse.

Use the **[Tab]** key or **[Shift][Tab]** key combination to move the cursor focus to the **Refresh** icon in the control bar.

HGrids Support

Table Refresh is not supported in HGrids.

Personalization Support

User and admin-level personalization support are not applicable for this feature.

Tables and Columns That Do Not Support Table Refresh

- LOV tables
- Inner tables
- Printable page
- Email facet
- Attachment tables
- Tables inside an embedded or parameterized pop-up
- OA Framework-based tables embedded in a JTT skin
- All tables and HGrids, when column reorder is turned off through its meta data property or controller code
- Tables rendered in portlets

Usage Restrictions

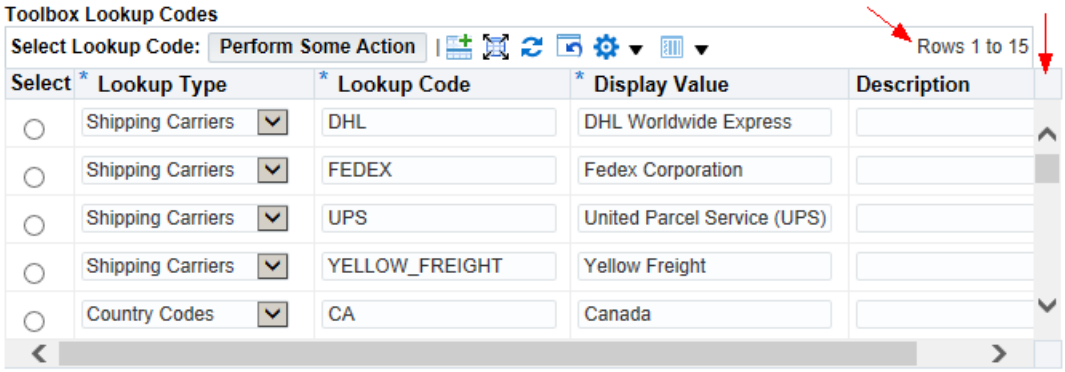
- If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.
- Since the Table Refresh feature is implemented on the basis of the table Sorting functionality, the same usage restrictions that apply for table Sorting apply here.

Behavior in Query Bean With Result Tables

Since Table Refresh does not save any user personalization data, the **Refresh** icon and functionality is available in the Simple Search, Advanced Search, and Views panels.

Vertical Scrolling

Vertical scrolling is introduced in Release 12.2.5 to enhance the usability of row navigation. It replaces the pagination mode where previously, users select the "Next" and "Previous" links to display next or previous sets of rows. These links are now replaced by simple text that identifies the range of rows currently displayed.



The following table identifies how users interact with vertical scrolling on different devices:

User Interactions For Vertical Scrolling By Device

Device	Input Control	Interaction Method
Desktop	Mouse + scroll bar	Same as for other scrollable regions.
Desktop	Mouse wheel	Same as for other scrollable regions.
Desktop	Keyboard	**[Up Arrow]** for moving up, **[Down Arrow]** for moving down.
Touch Devices	Finger	Same as for other scrollable regions.

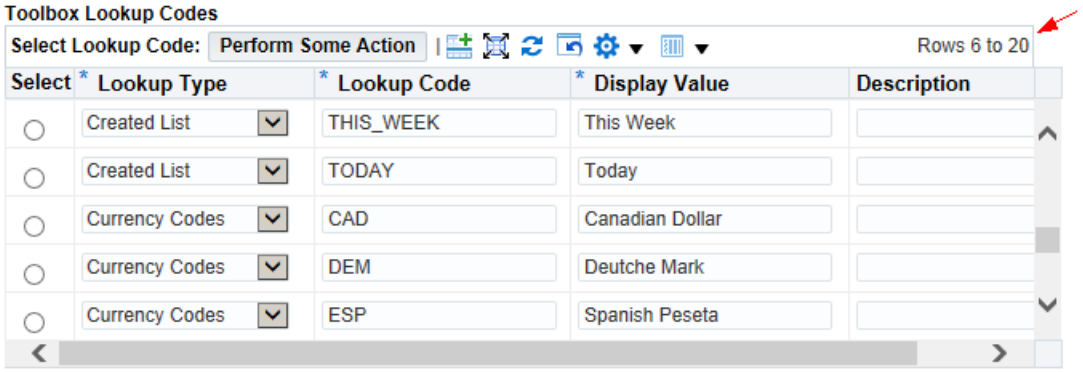
Forward Fetch - Next Event

The new vertical scrolling feature fetches more records into the UI than what is displayed for a smooth scrolling experience. The feature initially fetches 3 times the number of rows displayed. If the default number of rows displayed is five, then it initially fetches fifteen rows and then continues to forward fetch five rows at a time. Every response for the next forward event caches the older two thirds (ten, in this example) set of records. The event name is GOTO.

In the figure above, the Toolbox Lookup Codes table displays 5 rows at a time (the `Number of Rows Displayed` property is set to 5). The table initially fetches rows 1 to 15 but displays rows 1 to 5. When you vertically scroll forward, the table does the following:

- Caches rows 6 - 15.
- Fetches the next 5 rows.
- Displays the row range as 6 - 20, as shown in the figure below.
- Focuses on rows 16 - 20, as rows 6 - 15 are already displayed when you vertically scroll forward.

In this example, the forward fetch cycle will be rows 1 - 15, 6 - 20, 11 - 25, 16 - 30, and so on.



Backward Fetch - Previous Event

When you scroll backwards and release the vertical scroll, a hint displays to indicate the row range you select. For backwards scrolling, the table can fetch from a previously displayed range plus two times that number of records displayed, going forward. The event name is GOTO.

In the example above, where the `Number of Rows Displayed` property is set to 5, rows 1 - 15 are initially fetched and the focus is on rows 1 - 5. If you forward scroll twice, rows 11 - 25 are fetched and the focus is on rows 21- 25. The following happens when you scroll backwards:

- Hints for two range options appear, 6 - 10 and 1 - 5.
- If you scroll backwards to the range 6 - 10, the number of rows that the table fetches is equal to the selected range (6 - 10) plus two times the forward rows (11 - 20, which are cached since they were already fetched earlier).
- The table focuses on range 6 -10.

Impact on Existing Table Features

The vertical scrolling feature impacts the **Add Another Row** and **Total** features as follows:

- **Add Another Row** button - Upon selecting the **Add Another Row** button, a new row is added at the end of the fetched rows. The cursor focus is on the first editable field of the newly added row.
- **Total** button - Upon selecting the **Total** button, totaling includes all the records fetched into the UI. It includes the records that are visible as well as the records that are under the scroll. The new enhanced **Total** button text shows the record range totaled.

Vertical Scrolling Implementation and Support

- Declarative Implementation
- Runtime Control
- Touch Devices Support

- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Vertical Scroll
- Usage Restrictions

Declarative Implementation

Vertical scrolling is enabled by default in OA Framework-based tables. To turn off this feature for a table, use the Oracle JDeveloper OA Extension Property Inspector to set the `Row Navigation Policy` property on the table or advanced table region to **Page** to revert back to pagination mode. To turn on the feature, set the property to **scroll**.

Note: Turning off vertical scrolling in this way disables the feature for all users.

Important: A value of **False** for the profile option **FND: Enable Rich Table Interactions** overrides the `Row Navigation Policy` property value. For example, if the profile **FND: Enable Rich Table Interactions** is set to **False** at the Site, Application or User level, then vertical scrolling is disabled even if the `Row Navigation Policy` property for a table is set to **scroll**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. You cannot personalize, reuse, or easily extend programmatically created web beans. For example, if you add any column programmatically to a (classic or advanced) table web bean or create the table programmatically, the column resize functionality will be unavailable for the whole table as there is no personalize view support for programmatic tables or for columns added programmatically to a metadata-based table.

To enable or disable vertical scrolling, you can use the following APIs on `OATableBean`, or `OAAdvancedTableBean`:

```
public void setRowNavigationPolicy(String navigationPolicy)
public String getRowNavigationPolicy()
```

Supported values are: `UIConstants.SCROLL` and `UIConstants.PAGINATION`

Touch Devices Support

In touch devices, use the method that is native to the specific device to vertically scroll through the table.

Accessible Modes Support

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still take advantage of vertical scrolling through a simpler user interface that does not require a mouse.

In **Standard Accessibility** mode, use **[Up Arrow]** and **[Down Arrow]** to vertically scroll the table.

In **Screen Reader Optimized** mode, tables run in page mode as in earlier releases with, with "Next" and "Previous" links.

HGrids Support

Vertical scrolling is not supported in HGrids.

Personalization Support

User personalization support is not applicable for this feature.

See a summary of admin-level Rich Table personalization considerations in the Oracle Application Framework Personalization Guide.

Tables and Columns That Do Not Support Vertical Scroll

The following tables do not support vertical scrolling and run in page mode with "Next" and "Previous" links.

- LOV tables
- Inner tables
- Printable page
- Email facet
- Attachment tables
- Tables inside an embedded or parameterized pop-up
- OA Framework-based tables embedded in a JTT skin
- All tables and HGrids, when column resize is turned off through its meta data property or controller code

- Tables rendered in portlets

Usage Restrictions

- If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.
- Any developer code that is dependent on the `Number of Rows Displayed` property may fail as the vertical scrolling feature fetches 3*[*number of rows displayed*] into the UI. For example, if you set the range size of the table view object equal to the `Number of Rows Displayed` property, then either UI issues or server-side code failures may occur, depending on the number of actual records returned. To avoid this situation, you can use code similar to this:

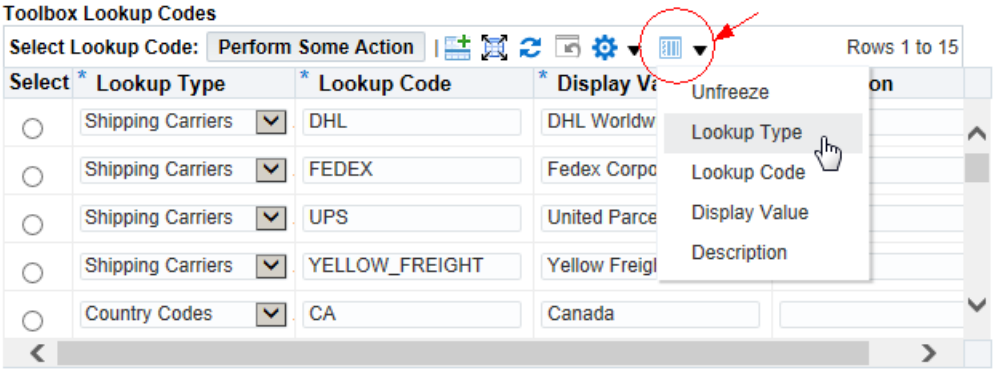
```
boolean isScrollModeTable =
   OATableUtils.isScrollModeTable(tableBean,pageContext);
if(isScrollModeTable)
{
   int rowsInUI =
      OATableUtils.getNumberOfRecordsToFetch(tableBean.getNumberOfRowsDisplayed());
   tableVO.setRangeSize(rowsInUI);
}
else
{
   tableVO.setRangeSize(tableBean.getNumberOfRowsDisplayed());
}
```

- If the cellFormat bean enclosing the table bean is center- or right-aligned using the `Horizontal Alignment` property, then in Mozilla Firefox, the vertical scroll bar will be hidden. To prevent this from happening, you should set the `Horizontal Alignment` property to **left** or leave this property null, which by default, indicates left alignment.
- If your controller code iterates all the records of the VO associated with the table using the `hasNext()` API, then internally, BC4J changes the range start of the VO. This impacts vertical scrolling by not showing the correct records or by producing Javascript errors in the page. To overcome this, update your code to save the range start and then set it back after iterating all the rows in the VO using the `next()` or `hasNext()` call, as in the example code below:

```
int savedRangeStart = vo.getRangeStart();
while(vo.next())
{
// Developer logic - existing
...
}
// Finally set the saved range start
vo.setRangeStart(savedRangeStart);
```

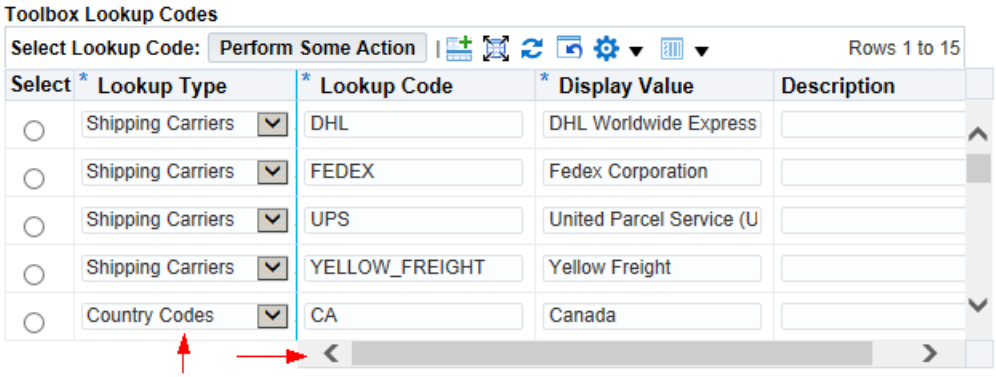
Column Freeze

From Release 12.2.5, you can select the **Column Freeze** icon in the table control bar to select one or more columns to keep in view while scrolling the table. Selecting the icon displays a menu that displays the columns as menu items you can choose to freeze. Note that the **Column Freeze** menu is enabled only when the table displays a horizontal scrollbar.



When you select a column from the column freeze menu, all columns above and up to the selected column in the menu are frozen in place in the table. The horizontally scrollable region readjusts to enclose only the columns that are not frozen. You can not freeze the last column of a table.

Note also that the first item on the **Column Freeze** menu is **Unfreeze**. Upon selecting **Unfreeze**, the table restores to its original state.



Column Freeze Implementation and Support

- Declarative Implementation
- Runtime Control
- Touch Devices Support
- Accessible Modes Support
- HGrids Support
- Personalization Support
- Tables and Columns That Do Not Support Column Freeze
- Usage Restrictions
- Behavior in Query Bean With Result Tables

Declarative Implementation

Column freezing is enabled by default in OA Framework-based tables. To turn off this feature for a table, use the Oracle JDeveloper OA Extension Property Inspector to set the `Enable Column Freeze` property on the table or advanced table region to **False**.

Note: Turning off column freeze in this way disables the feature for all users.

Important: A value of **False** for the profile option **FND: Enable Rich Table Interactions** overrides the `Enable Column Freeze` property value. For example, if the profile **FND: Enable Rich Table Interactions** is set to **False** at the Site, Application or User level, then column freeze is disabled even if the `Enable Column Freeze` property for a table is set to **True**.

Runtime Control

Warning: You should create web beans programmatically only if you cannot create them declaratively. You cannot personalize, reuse, or easily extend programmatically created web beans. For example, if you add any column programmatically to a (classic or advanced) table web bean or create the table programmatically, the column resize functionality will be unavailable for the whole table as there is no personalize view support for programmatic tables or for columns added programmatically to a metadata-based table.

To enable or disable column freezing, you can use the following APIs on `OATableBean`, or `OAAdvancedTableBean`:

```
public void setColumnFreezeEnabled(boolean colFreezeEnabled)
public boolean isColumnFreezeEnabled()
```

For example, to programmatically turn off column freezing you can call the API as follows:

```
table.setColumnFreezeEnabled(false);
```

Touch Devices Support

From Release 12.2.4, OA Framework provides gestures support for rich table interactions. For touch devices, you may Single Tap on the **Column Freeze** icon to display the menu list of columns to freeze or unfreeze.

Accessible Modes Support

If you have the **Self Service Accessibility Features** profile set to **Screen Reader Optimized**, the column freeze feature is disabled.

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility**, you may still take advantage of column freeze through a simpler user interface that does not require a mouse.

Use the **[Tab]** key or **[Shift][Tab]** key combination to move the cursor focus to the **Column Freeze** icon in the control bar.

1. Press **[Enter]** to display a menu that lists the columns of the table.
2. Press **[Tab]** to move the focus to the column names in the menu.
3. Use the **[Up Arrow]** or **[Down Arrow]** keys to navigate between column names, "wrapping" at the top and bottom of the menu.
4. With the focus on a column name, press **[Space]** to select the column to freeze.
5. With the focus on a column name, you can apply the change by pressing any of the following:
    - **[Esc]** which also closes the menu and returns focus to the menu-enabled item.
    - **[Tab]** which also closes the menu and navigates to the next item on the page.
    - **[Shift][Tab]** which also closes the menu and moves focus back to the menu-enabled item.

Use the **[Left Arrow]** or **[Right Arrow]** keys to scroll the table horizontally.

For more information about user interactions of menu-enabled items such as the **Column Freeze** icon, refer to the User Interactions section of the Menu Component topic.

HGrids Support

Column freeze is not supported in HGrids.

Personalization Support

When a user freezes columns to quickly personalize a table, the personalization is saved as a user-level view of the table in the MDS repository. You may examine or delete the personalization metadata in the MDS repository using the `JDR_UTILS` package.

See a summary of Rich Table personalization considerations in the Oracle Application Framework Personalization Guide.

Tables and Columns That Do Not Support Column Freeze

- Tables created programmatically
- Tables in which one or more columns are added programmatically
- Special table columns such as Details, Select or row header stamp columns
- All tables when the profile **Disable Self Service Personal** is set to **Yes**
- LOV tables
- Inner tables
- Printable page
- Email facet
- Tables with record history enabled
- Attachment tables
- Tables that contain a key or descriptive flexfield
- Tables inside an embedded or parameterized pop-up
- OA Framework-based tables embedded in a JTT skin
- All tables, when column freeze is turned off through its meta data property or controller code
- Tables that cannot be scrolled horizontally
- Tables rendered in portlets
- OA Framework pages containing tables that may be accessed via a Guest user login, such as the Supplier Registration page

Note: From Release 12.2.9, all these tables support Rich Interactions.

Usage Restrictions

- If a page contains multiple tables, you must assign a unique ID to each table to avoid impacting the rich table interactions on the page.
- Assign each item/column in a page a unique ID (which also adheres to the HTML standards) to avoid impacting the rich table interactions on the page.
- To ensure that tables function properly when you enable rich table interactions, follow the classic or advanced table implementation and coding standards carefully. For example, when you implement an advanced table declaratively, you can not add an item as a direct child of the advanced table, you may only add an item to a column or column group of the advanced table. Similarly, you should follow this standard when you add items programmatically to an advanced table.
- A user may still reorder or hide/show a frozen column using the **Table Settings** menu.
- For an advanced table that contains column spans, only the top level (or parent) columnGroups may be frozen.

Behavior in Query Bean With Result Tables

The column freeze feature behaves as follows in the result table that appears in each of the query bean panels.

- **Simple Search Panel**
  - Column freeze is allowed.
  - Frozen column changes go to an OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - If there is no default view associated with the query region then <*TableName*>**_UserView becomes the default view.
  - Whenever a user returns back to the page with the Simple Search panel, the changes made in <*TableName*>**_UserView is applied.
  - The view <*TableName*>**_UserView is shared between Simple Search and Advanced Search panels.
- **Advanced Search Panel**
  - Column freeze is allowed.
  - Frozen column changes go to an OA Framework-created view named <*TableName*>**_UserView where <*TableName*> is the name of the table bean.
  - If there is no default view associated with the query region then <*TableName*>**_UserView becomes the default view.
  - Whenever a user returns back to the page with the Advanced Search panel, the changes made in <*TableName*>**_UserView is applied.
  - The view <*TableName*>**_UserView is shared between Simple Search and Advanced Search panels.
- **Views Panel**
  - This panel will list <*TableName*>**_UserView in the Views drop down list if the user previously personalized the table using any rich interaction in the Simple Search and Advanced Search panels.
  - In the case of a seeded default view, the view is non-editable, so column resize is disabled.
  - Column freeze is enabled in all views except seeded views.
  - If <*TableName*>**_UserView is selected from the Views drop down list and a user performs further changes to this view, then the changes are also reflected in the Simple and Advanced Search panels since this view is used for rendering tables in the Simple and Advanced Search panels.

Note: If a Query Bean configuration contains multiple tables/advanced tables as its children, column freeze is supported only in the first table. As a result we recommend you disable column freeze altogether in this scenario.

Disabling All Rich Table Interactions

You may disable all rich table interactions in a table through a single API call using the following method on `OAHGridBean`, `OATableBean` or `OAAdvancedTableBean`:

```
hgridBean.disableRichInteractions();
tableBean.disableRichInteractions();
advancedTableBean.disableRichInteractions();
```

Support for Touch Devices

From Release 12.2.4, OA Framework provides gestures support for rich table interactions. For touch devices, you may horizontally **Pan** on a column header to resize or reorder a column, **Hold** on a column header to reveal the hide/show icon for a column, vertically Pan the first set of records in a table to refresh the table or **Swipe-Up / Swipe-Down** to navigate to the next or previous set of rows.

Debugging Rich Interaction Issues

To enable screen-level logging of rich interaction information, you must first enable Logging.

1. Select the **Diagnostics** global button to display the Oracle Diagnostics screen.
2. For Diagnostic, select **Show Log on Screen** and set the Log Level to **Procedure (2)**.

Step 3. Navigate to the page you wish to debug. A separate log table for each rich classic table, rich advanced table or rich HGrid in the page appears. Each log table displays debug information related to column order, width of columns, hide/show column status, and so on.

Personalization Considerations

- See a summary of Rich Table personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key issues with suggested workarounds if available.

Support Matrix

This table provides a quick reference to the support status of some specialized table settings and Rich Table Interactions.

Table Feature	RTI Enabled on Table?	RTI Available for Column?	Comments
Table with DFF rendered as Popup	Yes	Yes	Bug 24288268
Table with DFF not rendered as Popup	No	No	From Release 12.2.5, it is recommended for a DFF in a table to be rendered as a popup.
Table with KFF combinations LOV display	Yes	Yes	Bug 28402932
Table with KFF non-combinations LOV display	No	No	From Release 11.5.10, KFF should be displayed as combinations LOV as per BLAF UI guidelines.
Table with record history enabled	Yes	No	Bug 28402955
Table with programmatic columns added in controller	No	No	Attribute 'IS_SPECIAL_COLUMN_ATTR' needs to be set to True on the column.

Related Information

- BLAF UI Guideline(s)
- Javadoc File(s)
  - `oracle.apps.fnd.framework.webui.beans.table.OATableBean`
  - `oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean`
- Lesson(s)

Sample Code

Save Model ('Warn About Changes')

Overview

The Oracle Browser Look-and-Feel (BLAF) UI Guideline: Save Model describes how data saving (and caching) should behave in the context of different designs. For the most part, compliance with this guideline is a function of proper page design coupled with Cancel, Apply and Submit button implementations as needed. One of the Save Model requirements does require additional development effort. Given that users should be warned when leaving a page (or a flow of related pages) with pending changes, this document describes how to enable this check.

Note: To fully understand the save model, you must also understand state management in the OA Framework, including how data is read from and written to the model layer from the view layer. If you are new to the OA Framework, please read Chapters 2 and 3 in the Developer's Guide. Also, see the ToolBox Tutorial for a series of hands-on labs that will help you understand these broader concepts.

Implementation

In general terms, you must ensure that any user navigation that might result in the loss of application module transaction state be flagged to raise a warning. Specifically, if you follow the Page-Level Control and Item-Level Control instructions below, the OA Framework displays a Javascript Alert message asking the user if she wishes to proceed whenever she performs one of the following actions with pending data changes:

- Selecting a tab, horizontal navigation or side navigation menu entry
- Selecting a global button (implies retainAM=N)
- Selecting a breadcrumb link.
- Selecting a link with URL parameter retainAM=Y and the `Warn About Changes` property set to **True** (note that this applies only to declaratively defined links; this does not apply to menu links)
- Selecting an image which does not post your changes. For example, selecting an LOV icon or a long tip window will not trigger the save model warning. However, if the user selects on an image that navigates to Yahoo, for example, the warning is displayed.
- Selecting a link that performs a form submit and has the `Warn About Changes` property set to **True**.

Note: Changing anything in a Shuttle region triggers the save model warning. However, there is currently no declarative or programmatic support to explicitly turn the save model off for the Shuttle web bean.

Page-Level Control

For any single page, or first page in a navigation flow where the `retainAM` URL parameter value is set to **Y** (and the pages share the same root UI application module) as you navigate between each page, set the `Warn About Changes` property to **True** on the **pageLayout** region. Alternatively, call `pageContext.activateWarnAboutChanges()` in a `processRequest()` method associated with the same page where you might otherwise set this property declaratively.

It is important that you implement this check on the *first* page of a flow. If, for example, you don't implement it until the third page in a six-page flow, the first two pages won't be covered (unless the user returns to these pages after accessing the third page).

Note: The programmatic implementation overrides the declarative one. So, if you enable the check programmatically, but the `Warn About Changes` property is **False**, the check is still performed.

For pages coded that use the old `pageContext.activateBlockOnNavigation()` method to enable this check, this functionality has been extended to include all the pages in a flow where the `retainAM` URL parameter is set to **Y** as you navigate between pages. Previously, this method applied only to the single page where it was called.

Item-Level Control

Note: Item-level settings apply only if this check is enabled in the containing page.

The item level controls are available on the following types:

- OAMessageStyledTextBean
- OAStaticStyledTextBean
- OAImageBean
- OALinkBean
- OASubmitButtonBean
- OAButtonBean
- OAMessageCheckBoxBean
- OAMessageChoiceBean
- OAMessageDateFieldBean
- OAMessageListBean

- OAMessageLovChoiceBean
- OAMessageLovInputBean
- OAMessageRadioButtonBean
- OAMessageRadioGroupBean
- OAMessageTextInputBean
- OATreeDefinitionBean (hierarchy column) for an HGrid or a Gantt chart (see the respective documentation for these components for special implementation instructions)
- OASideNavBarBean (see the special implementation instructions below).

By default, the `Warn About Changes` property is set to **True** for each of the items in this list *except for* the `OASubmitButtonBean` whose default value is **False** (a ccording to the UI guidelines, the "Warn About Changes" check should be performed for all submit button instances *except for* Cancel, Apply and Submit buttons). If you want to explicitly enable the check for a submit button, set this property to **True**. Note that can also set it programmatically by calling `OASubmitButtonBean.setWarnAboutChanges(pageContext, Boolean.TRUE)` in `processRequest()`.

Tip: The `Warn About Changes` property is set to **False** in the standard Cancel button attribute set `oracle/apps/fnd/attributesets/Buttons/Cancel`. If you use this attribute set for your Cancel button, it will automatically comply with the UI guidelines.

Side Navigation

By default, selecting the links in a side navigation warns the user if the "Warn About Changes" feature is enabled for the current page and there are pending changes in the transaction.

Since the side navigation component is not created declaratively, you must programmatically disable the "Warn About Changes" feature if you are not satisfied with the default behavior. So, for example, if you want users to be able to navigate within this component with pending changes, implement the following `processRequest()` logic (note that, as a rule, you should also retain your root UI application module as the user navigates back and forth because this design implies that the pages within the side navigation are part of the same task).

```
OASideNavBean sideNav = (OASideNavBean)pageContext.getPageLayoutBean().getSideNav();

if (sideNav != null)
{
  sideNav.setWarnAboutChanges(false);
}
```

Alternatively, you could get a handle to the side navigation as shown and selecting disable the save model warning for individual links, although for Oracle's internal E-Business Suite developers, this particular UI must be approved by the corporate UI team.

Items Configured to Perform a Form Submit

For items from the list above that do not ordinarily submit the form when selected (like an `OALinkBean` or an `OAImageBean`), you must configure the item to submit the form using the **fireAction** event if you want the "Warn About Changes" check to be enabled (see the Declarative Submit Form documentation for `fireAction` implementation instructions). These items cannot submit the form using a Javascript URL.

Note: If you configure any of these beans to perform a form submit with the **fireAction** event, you must also disable client and server side validation if you want the save model test to be performed (set the `Disable Client Side Validation` and `Disable Server Side Validation` properties to **True**).

Items Without a Data Source

Any changes to items that do not have associated view object attribute bindings (search fields, for example) should not cause the "Warn About Changes" message to display when the user navigates out of the page. To prevent this from happening, you should set the `Warn About Changes` property to **False** on any of the following item types used for this purpose:

- OAMessageCheckBoxBean
- OAMessageChoiceBean
- OAMessageDateFieldBean
- OAMessageListBean
- OAMessageLovChoiceBean
- OAMessageLovInputBean
- OAMessageRadioButtonBean
- OAMessageRadioGroupBean
- OAMessageTextInputBean

Data Defaulting and Warn About Changes

The "Warn About Changes" feature checks transaction state (whether view object rows are "dirty") to determine whether to show the warning message. If your application sets default data in a new page, and you don't want the "Warn About Changes" warning to display if the user leaves the page without making any changes, you must explicitly reset the row state as described in the OA Framework Model Coding Standard M69.

Personalization Considerations

- See a summary of Save Model personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key Save Model issues with suggested workarounds if available.

Related Information

- BLAF UI Guidelines
  - Save Model
- OA Framework Developer's Guide
  - Declarative Submit Form
  - HGrid
  - Gantt
- Javadoc
  - `oracle.apps.fnd.framework.webui.OAPageContext`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageStyledTextBean`
  - `oracle.apps.fnd.framework.webui.beans.OAStaticStyledTextBean`
  - `oracle.apps.fnd.framework.webui.beans.OAImageBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OALinkBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OAButtonBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OASideNavBarBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageCheckBoxBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageChoiceBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageDateFieldBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageListBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageLovChoiceBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageLovInputBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageRadioButtonBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageRadioGroupBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageTextInputBean`

Search

Contents

- Overview
  - Primary Search Regions
  - Supplemental Search Regions
  - Using Descriptive Flexfields
- Search Implementation Considerations
  - Using Query Regions
  - Using Nesting Regions
  - Using Search Persistence
- Simple Search
  - Declarative Implementation: Results Based Search
  - Declarative Implementation: Auto Customization Criteria
  - Manual Search
- Advanced Search
  - Declarative Implementation: Results Based Search
  - Declarative Implementation: Auto Customization Criteria
  - Manual Search
  - Runtime Control
    - Custom Validations
    - Handling processFormRequest
    - Setting the Default Criteria Count
- (User Personalizable) Saved Searches
  - Implementation
  - Non-Personalizable Saved Searches
- OAQueryBean Behavior in Detail
  - Criteria Object Overview
  - Default WHERE Clause Generation
  - Overriding the WHERE Clause Generation
- Multiple Query Regions
- Quick Search
- Side Navigation Search
- Personalization Considerations
- Searching Using Descriptive Flexfields
- Known Issues
- Related Information

Overview

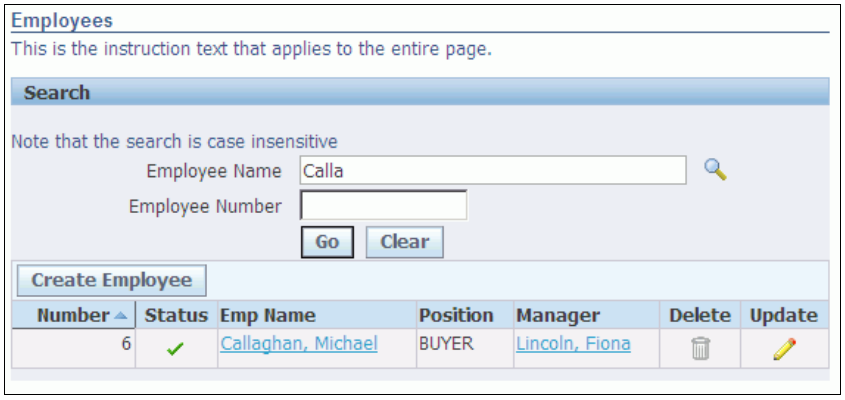
Primary Search Regions

The Oracle Browser Look-and-Feel (BLAF) UI Guideline: Search and Query Templates describes the following core design options for searching:

- Simple Search - presents users with a limited number of basic search criteria (usually 1 - 4 fields) with an optional hide/show control to view additional search criteria.
- Advanced Search - presents users with extensive search criteria and power search capabilities including the ability to declaratively specify "AND" or "OR" searches and datatype-specific operators on individual fields. For example, a "Salary" value must be greater than 100000.
- (User Personalizable) Saved Searches - presents users with a personalizable list of saved, named searches.
- Runtime Control - OAQueryBean - describes special instructions for manipulating the query region at runtime.
- Multiple Query Regions - describes special instructions for displaying multiple query regions on a single page.

In all cases, these regions are presented on the same page with the associated results, as shown in the following illustration of a simple search:

Figure 1: Example of a simple search and results table on the same page



The simple search, advanced search and user-personalizable saved searches can be displayed individually or in combination to provide extremely flexible searching on a list of objects. Figure 2 shows how the user toggles between these search panels if all three are enabled in a single page.

Figure 2: toggle navigation between a Simple Search, Advanced Search and user-personalizable Saved Searches region on the same page

Supplemental Search Regions

The UI Guidelines also describe two designs for presenting special search "shortcuts" to users:

- Quick Search - as shown in Figure 3, this renders immediately below the menu and presents the user with a single search field and optional poplist of search object types
- Side Navigation Search - as shown in Figure 4, this also presents the user with a single search field and optional poplist of searchable object types, but the search region renders in the Side Navigation instead.

Figure 3: Valid Quick Search configurations per the BLAF UI Guidelines

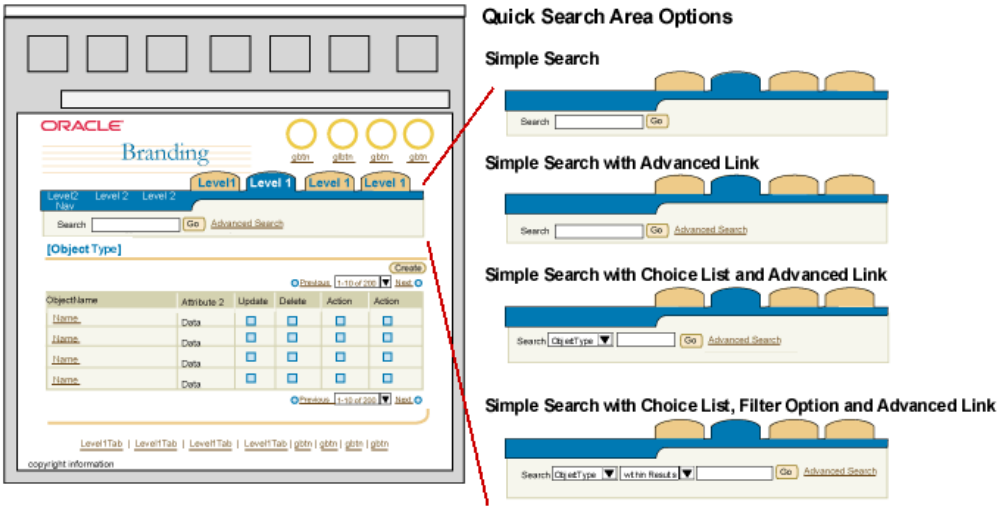
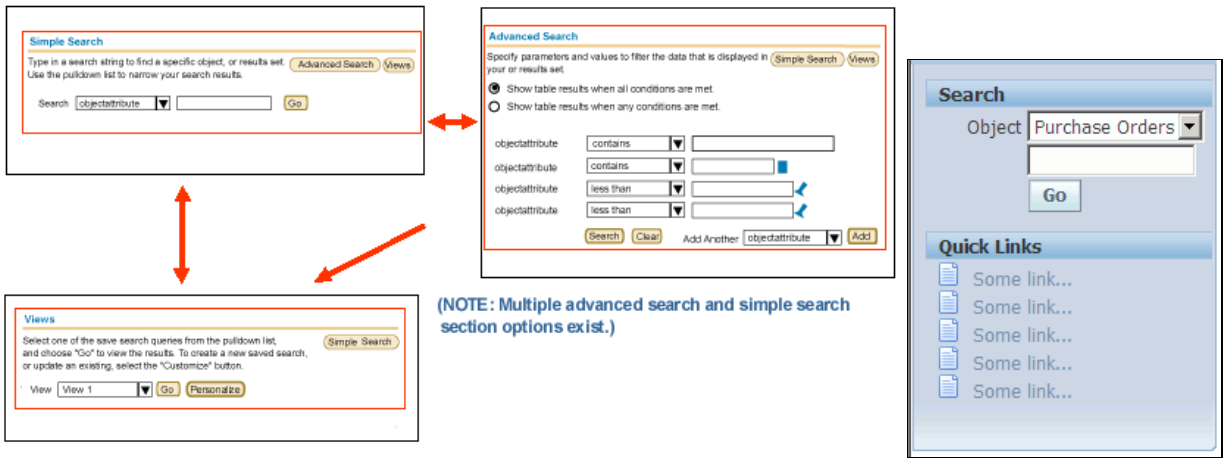


Figure 4: Example side navigation search.



Using Descriptive Flexfields

From Oracle E-Business Suite Release 12.2.9, you can also perform searches using DFF (descriptive flexfields) as a criterion under a Search (Query) region.

This new feature is available for query with DFF items in both Classic and List Search mode. It is supported for query bean with both Results Based Search (RBS) and Auto Customization Criteria (ACC) construction mode.

- Supported Flows in Classic Search Mode:
  - Simple Search with DFF criteria
  - Advanced Search with DFF criteria
  - Save or Update Saved Search with DFF criteria
- Supported Flows in List Search Mode:
  - Search with DFF criteria in filter panel
  - Save or Update Saved Search with DFF criteria

Searching using DFFs requires specifically uptaking by products teams, and is discussed in its own section later in this chapter.

Search Implementation Considerations

This section discusses using query regions, nested regions, and search persistence when implementing Searches. See the Simple Search and Advanced Search implementation sections below for guidance.

Using Query Regions

This section provides an overview of query regions, and discusses query region constraints and search criteria recommendations.

You can build the simple and advanced searches as if they were regular regions and then implement the associated button press to execute the query. However, if possible, use the special query region.

Note: You *must* use the query region if you want to implement the user-personalizable saved searches. Furthermore, it is *strongly* recommended that you use the query region to implement searching on a HGrid as the alternative requires significant effort on your part.

Finally, the Quick Search and Side Navigation Search UIs cannot be implemented using the query region. Their manual implementations are described below.

Understanding Query Regions

When you add a query region to a pageLayout region, OA Framework automatically generates an `oracle.apps.fnd.framework.webui.beans.layout.OAQueryBean` which, depending on its configuration, works in concert with a child table, advanced table or HGrid to implement any combination of simple search, advanced search and saved search panels. OA Framework automatically generates buttons as appropriate for toggling between the applicable regions.

The simple search and advanced search panels can be constructed using three different modes, which indicate the level of region and search construction automation.

Construction Mode	Region Construction Impact	Search Execution Impact
resultsBasedSearch	OA Framework automatically renders both the Simple and Advanced search regions based on the designated queryable items in the associated table or HGrid. **Note:** The search regions automatically include both a **Go** and a **Clear** button.	OA Framework automatically executes the underlying search when the user selects the **Go** button. If the underlying view object is "dirty" (it has pending changes), OA Framework displays an error message instead of executing the query.
autoCustomizationCriteria	OA Framework automatically renders both the Simple and Advanced search regions based on the corresponding Simple search and Advanced search regions that you define and specify as named children of the query region. **Note:** The search regions automatically include a **Go** button. In addition, the Advanced search region includes a **Clear** button.	OA Framework automatically executes the underlying search when the user selects the **Go** button. However, developers *must* explicitly define mappings between items in the Search panel and items in the table/HGrid region. As in the resultsBasedSearch case, OA Framework displays an error message instead of executing the query if the underlying view object has pending changes.

none	The Search regions are rendered based on the Simple Search and Advanced Search regions that you define and specify as named children of the query region. **Note:** You *must* implement your own **Go** button in this mode.	The underlying search must be executed by the developer.

Detailed instructions for implementing the simple search, advanced search and saved searches options are provided below.

Query Region Constraints

- LOV Choice components are *not* supported. You should *not* mark a **messageLovChoice** result table column as being "queryable" when using a query region.

Search Criteria Recommendations

- Text Field LOV components (messageLovInput items) used as search criteria within a query region should have their `Disable Validation` property set to **True**. As a result, users will lose the validation behavior that occurs when they tab out of the text field LOV (autocompletion). However, this loss is minor compared to the awkward behavior that results when the `Disable Validation` property is set to **False** for the messageLovInput item in the context of a Query web bean.

  When the `Disable Validation` property is set to its default value of **False**, the following behavior occurs:

  - In simple search - an "equals" search is done. Because there is no way to specify the ID formValue for the messageLovInput in a simple search, validate-on-submit does *not* work even though `Disable Validation` is **False**. In addition, an "equals" search is performed for the messageLovInput item because `Disable Validation` is **False**. This can lead to confusion for the user since there is no indication that the value for the messageLovInput item has to be exact.
  - In advanced search autoCustomizationCriteria (ACC) mode - awkward errors occur. In the advanced search ACC mode, it is possible to specify the ID formValue for the messageLovInput. However, OA Framework awkwardly prompts the user to "Select a valid value" when the user enters a partial value in the Text Field LOV and presses the **Go** button. Instead, it should display the search results in the Results table without displaying any LOV errors.
  - Typing in a value in the Text Field LOV and then selecting **Clear** brings up the LOV modal window. Although this is expected behavior from the LOV perspective, it is confusing when the messageLovInput is search criteria in the context of a query region. By setting the `Disable Validation` property to **True**, this behavior does not occur.

Using Nested Regions

To create a layout in the simple search panel using a queryBean in ACC mode, such as:

```
Text1 ----------------              Text2 -----------------
Currency ------------    * ---------  Text3 -----------------
```

* represents a textInput without a prompt.

define the following layout:

```
messageComponentLayout
                        |
                        messageLayout
                                    |
                                    rowLayout
                                            |
                                            textInput1 (prompt - currency)
                                            textInput2 (no prompt, required)
```

Note: The **textinput** items are *not* direct children of the **messageComponentLayout** or **messageLayout**, they are inside a nested container **rowLayout**. (Nested criteria regions are now supported).

OA Framework *always* renders the advancedSearch region as the filter region in Create Saved Search Page, based on the metadata specified in the query page. If an advancedSearch panel is *not* defined, OA Framework uses the simple search metadata to create the advanced search filter region.

Therefore, if you need to define a layout with nested criteria regions you *must* define an advanced search region in the query region and create proper mappings for the criteria items, which are nested in the simple search panel. This ensures that OA Framework *always* uses the metadata from the defined advancedSearch panel and there are no layout problems in Create Saved Search Page.

Note: If you do *not* want the advanced region to be displayed in the page you *must* still define it and set the `Include Advanced Panel` property of the query bean to **false**.

Using Search Persistence

In the past, if a user performed a search and then drilled down to a details page, search criteria was lost when the user navigated back to the Search page, unless the developer retained the AM or created a custom implementation to persist the search criteria. As of Release 12, developers no longer need a custom implementation to persist search criteria, as the **query** region now has an automatic search persistence mechanism in place.

Generally speaking, you should still retain the AM in favor of using the automatic search persistence mechanism for performance optimization reasons. However, if retaining the AM presents a scalability issue because the underlying view object (VO) had hundreds of attributes, then you should you turn off retainAM and allow the **query** region to automatically handle all the search criteria binding and use the automatic search persistence mechanism.

Note: If you retain the AM state, the automatic search persistence mechanism will not be active. The search persistence mechanism is automatic only if the AM state is not retained.

When a user selects the **Go** button to initiate a search on the **query** region, the OAQueryBean automatically caches the search criteria on the user session. After the user drills down to the detail page of the Search results and then returns to the Search page, the OAQueryBean restores the search criteria from the cache and re-executes the query using that saved critieria.

Note: Since the query is re-executed, any sort, navigation or transient UI state of the results table will not be preserved. Saving the entire table state presents high overhead which is what this particular feature is trying to avoid.

Runtime Control

If you override the default WHERE clause generation of the **query** region by using `getCriteria` or `getNonViewAttrCriteria` in your controller, you must make some changes to your code in order to persist the search criteria on the session. The changes you have to make can be categorized by three different cases.

Case 1 - Your controller handles a subset of the search fields by not defining the search mapping for them.

For the subset of search fields handled by your controller, you need to get the restored search criteria values from the web beans that correspond to the search fields and bind them to the VO in the `processRequest` method of your controller. Upon returning from a detail page, the search criteria entered in the search fields are restored in the `processRequest` cycle of the `OAQueryBean`. The following is an example of the code change you need to make in this case:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
  OAQueryBean queryBean = (OAQueryBean) webBean;

  String value = (String) ((OAMessageTextInputBean)
    webBean.findChildRecursive("job")).getValue(pageContext);
  String currentPanel = queryBean.getCurrentSearchPanel();
  if (SEARCH.equals (currentPanel) && value != null)
  {
    // Build the appropriate where clause for job
    // and set it on your VO using the initQuery method.
    OAApplicationModule am = pageContext.getRootApplicationModule();
    Serializable[] parameters = {value};
    am.invokeMethod("initQuery", parameters);
   }
}
```

Case 2 - Your controller uses `getNonViewAttrCriteria`.

Your controller calls the `getNonViewAttrCriteria` API in `processFormRequest` to return a criteria dictionary for the subset of the fields you want to handle. In this case, after returning from a detail page, you can get access to the same criteria dictionary in the `processRequest` of your controller by calling the following API on `OAQueryBean` (if the current search panel is a Simple or Advanced panel) :

```
Dictionary[] getNonViewAttrCriteriaFromSearchCache(OAPageContext)
```

Use `getNonViewAttrCriteria(OAPageContext)` if the current panel is a Saved Searches panel.

You can then user this criteria dictionary to handle the criteria yourself in `processRequest`. You should extract this logic to a custom method, such as "`handleCriteria`" and call it from `processFormRequest` and `processRequest`. The following is an example of the code change you would have to make in this case:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
  OAQueryBean queryBean = (OAQueryBean) webBean

        Dictionary[] nonViewAttrCriteria = null;
  String panel = queryBean.getCurrentSearchPanel();
  if(SEARCH.equals(panel) || ADVANCED_SEARCH.equals(panel))
    nonViewAttrCriteria = queryBean.getNonViewAttrCriteriaFromSearchCache(pageContext);
  else if(CUSTOMIZE.equals(panel))
    nonViewAttrCriteria  = queryBean.getNonViewAttrCriteria(pageContext);
```

```
  if (nonViewAttrCriteria!=null)
    handleCriteria(pageContext, nonViewAttrCriteria);
}
```

Note: You should use the Auto Customization Criteria mode of the query bean.

Note: Do not use `getNonViewAttrCriteriaFromSearchCache` if you are using `getCriteria` in `processRequest` for the purpose of suppressing WHERE clause generation and query execution in the Saved Search page.

Case 3 - Your controller uses `getCriteria`.

You controller calls the `getCriteria` API in `processFormRequest` to return the entire criteria dictionary and trigger OA Framework to bypass the WHERE clause generation and query execution. In this case, after returning from a detail page, you can get access to the same criteria dictionary in the `processRequest` of your controller by calling the following API on the `OAQueryBean` (if the current search panel is a Simple or Advanced panel) :

```
Dictionary[] getCriteriaFromSearchCache(OAPageContext)
```

Use `getCriteria(OAPageContext)` if the current panel is a Saved Searches panel.

You can then implement your own custom method such as "`handleCriteria`" to use this criteria dictionary to handle the criteria and execute the query in `processRequest`. The following is an example of the code change you would have to make in this case:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
  OAQueryBean queryBean = (OAQueryBean) webBean;

  Dictionary[] searchCriteria = null;
  String panel = queryBean.getCurrentSearchPanel();
  if(SEARCH.equals(panel) || ADVANCED_SEARCH.equals(panel))
    searchCriteria = queryBean.getCriteriaFromSearchCache(pageContext);
  else if(CUSTOMIZE.equals(panel))
    searchCriteria = queryBean.getCriteria(pageContext);

  if (searchCriteria!=null)
    handleCriteria(pageContext, searchCriteria);
}
```

Note: You should use the Auto Customization Criteria mode of the query bean.

Note: Do not use `getCriteriaFromSearchCache` if you are using `getCriteria` in `processRequest` for the purpose of suppressing WHERE clause generation and query execution in the Saved Search page.

Search Persistence Context

Some Search Pages are used in the context of a master record. For example, the Oracle Sourcing product tracks negotiations and negotiation lines. The search performed within a negotiation for negotiation lines is specific to that negotiation.

To enable search persistence for a specific context, you can call the `setSearchPersistenceContext` API on `OAQueryBean`:

```
setSearchPersistenceContext(String context)
```

Note: You should call this API from the `processRequest` method of a controller defined above the **query** region.

Normally, the complete document reference of the **query** region is used as the cache key for the search criteria on a user session. By calling this API, the context value you pass in is appended to the existing cache key for this particular **query** region so that the search criteria are always cached based on that context.

For the negotiations example described earlier, since the Negotiation Lines Search region is always in the context of a particular negotiation line, the search persistence context should be set to the Negotiation Line Id.

Simple Search

As described above, a simple search displays a small number of search criteria items in a constrained format as shown in Figures 5 and 6 below.

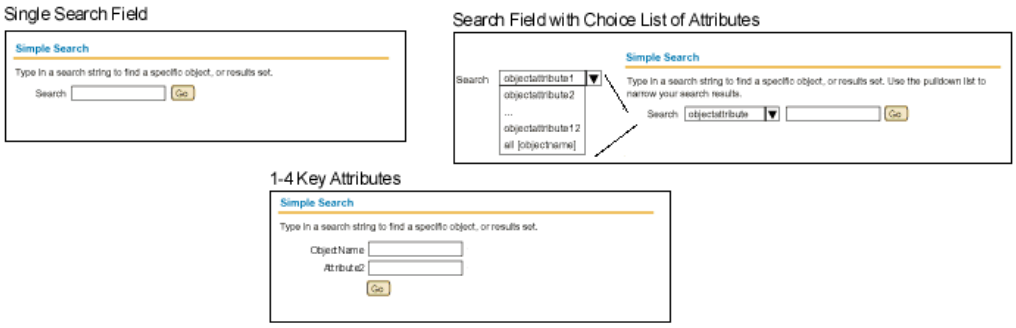
- To implement a simple list of items that map directly to the associated table columns, leverage the Query bean to quickly configure a fully automated search. See the Declarative Implementation: Results Based Search for instructions.
- To implement a simple search with custom items, you need to manipulate the search itself. For example, the user selects a poplist value that *must* be translated to a particular date range. Use the Query bean with your custom query handling. See the Declarative Implementation: Auto Customization Criteria.

  Note: The custom Simple Search panel that you create *must* be a messageComponentLayout region.
- To implement the search field with a poplist of search attributes, you cannot use the Query bean. See the Declarative Implementation:

Manual Search for instructions.

- The None mode is provided for backward compatibility. For any new query regions that you create, the *Results Based Search* and *Auto Customization Criteria* options should satisfy all of your requirements.

Figure 5: Example of valid Simple Search region designs per the BLAF UI Guidelines.



The **Go** button is normally rendered below the search items. However, as shown in the examples above, if a simple search has only one **messageTextInput**, or one **messageTextInput** and one **messageChoice** (where **messageChoice** acts as a conditional poplist), then the **Go** button is rendered next to the **messageTextInput**.

Note: The **messageChoice** acts as a conditional poplist when the prompt is set only for the **messageChoice** and *not* set for the **messageTextInput**.

For a search region that contains a hide/show control, the **Go** button renders below the hide/show control in both the expanded and the collapsed state.

Figure 6: Example of Simple Search region with collapsed "Show More Search Options" hide/show control.

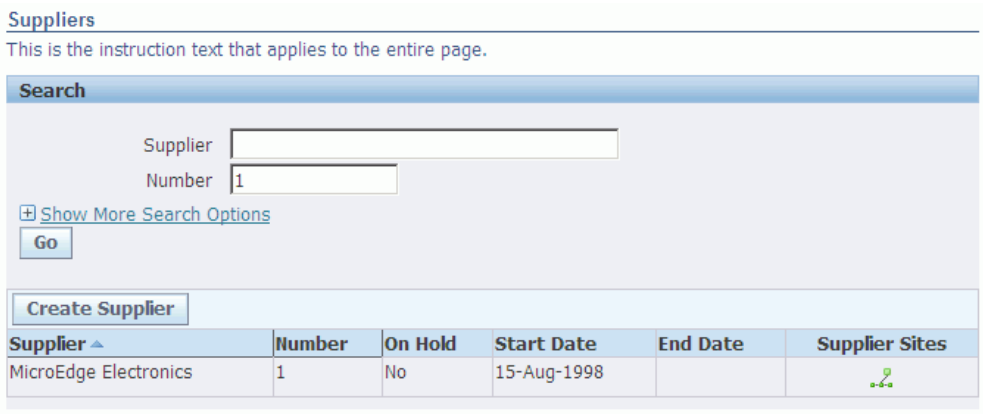


Figure 7: Example of Simple Search region with expanded "Show More Search Options" hide/show control.



Declarative Implementation: Results Based Search

To create a Simple Search panel in the results based search mode:

Step 1: Create the view object that you will use for searching. In this mode, there is no need to create an `initQuery()` method on the view

object to set the WHERE clause, bind the search criteria, and execute the query. OA Framework handles this on your behalf.

Tip: If your view object's definition requires bind values, or you *must* manually append an unrelated WHERE clause yourself every time the query is executed, you can proceed in one of two ways. In either scenario, OA Framework appends the user's search criteria to your WHERE clause immediately before executing the query, so any WHERE clause characteristics that you define are preserved.

1. Override the `executeQuery()` method in your `*VOImpl` to modify your WHERE clause and then call `super.executeQuery()`. This is recommended if you exercise the same logic every time the view object is queried.
2. Handle the generated **Go** button's press and call an `initQuery()` method on your view object where you can modify the WHERE clause and/or set bind values.

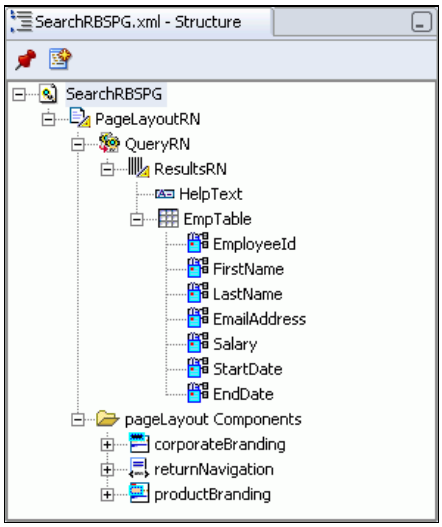
Step 2: Select your pageLayout region in the JDeveloper Structure pane, right-click and select **New > Region**. Set the region's `Style` to **query**, specify an `ID` that complies with the OA Framework File Standards, (all regions and items that you create must comply with these standards), and set the following properties:

- `Construction Mode` - set to **resultsBasedSearch**.
- `Include Simple Panel` - set to **True**.
- `Initial Panel` - if you include the Advanced Search and/or Saved Searches option, specify the initial display panel.
- `Simple Search Instructions` - instruction text for the simple search region.
- `Simple Button Label` - if you include the Advanced Search and/or Saved Searches option, specify the label of the button that OA Framework renders for toggling to the simple search. The default label is "Simple Search".

Note: In the Results Based Search mode, OA Framework automatically generates a "Simple Search" header for you at runtime to contain the query region. There is no need to manually add this to the component hierarchy.

Step 3: Select your query region in the structure panel, right-click and select **New > Region Using Wizard** to quickly create your table or HGrid search results regions. Remember to give the region a standards-compliant `ID` and specify appropriate attribute sets for the associated items. See the Tables and HGrid documentation for additional information about creating these components. (This is particularly important for the HGrid as there are additional properties that you must set on that component to properly enable searching). Figure 8 illustrates a typical Results Based Search structure.

Figure 8: Example of a Results Based Search structure in the ToolBox Sample Library.



Step 4: Select the table/HGrid region that you created in Step 3 and set the `Search Allowed` property to **True** for any items that should be used as search criteria. These items are automatically included in the simple search region according to the rules described below, and OA Framework automatically executes the corresponding query when the user selects the **Go** button.

- If the searchable table item is a poplist, the corresponding search region item is the same poplist.
- If the searchable table item is a text input field, the corresponding search region item is a text input field or a date field as appropriate based on the column's datatype.
- If the searchable table item is an LOV text input field, the corresponding search region item is the same (with the same LOV definition).
- If the searchable table item is a display-only value (like a messageStyledText, a raw text, a formatted text and so on), OA Framework generates a text input field or date input filed as appropriate based on the column's datatype.

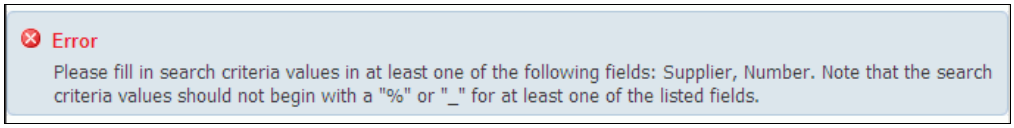
Step 5: Set the `Selective Search Criteria` property to **True** for any search criteria items that ensure a performant query. See the OA Framework View Coding Standards for additional information.

Note: OA Framework treats these values as being *individually* required: if the user enters any one of the required items, the query can proceed.

You currently cannot require that the user enter search criteria in more than one of these required items.

As shown in Figure 9 below, if the user fails to specify at least one of the required search values, an error message displays with the list of candidate items so the user can enter a value and proceed with the search.

Figure 9: Example of an error displayed when the user fails to enter a value for at least one of the "Selective Search Criteria" items in the simple search (note that the actual error message may differ slightly).



Step 6: Save your work.

Note: If you create a results based search, do not add custom search regions that are required for the "auto customization criteria" mode described next.

Declarative Implementation: Auto Customization Criteria

Warning: When implementing both a simple and advanced search using auto customization criteria, the simple search *must* be a subset of the advanced search. This is required because in the Personalization module, if you try to save a search, OA Framework always displays the advanced search.

To create a Simple Search panel in the Auto Customization Criteria mode:

Step 1: Create the view object that you will use for searching as described in the Results Based Search section above.

Step 2: Select your pageLayout region in the JDeveloper Structure pane, right-click and select **New > Region**. Set the region's `Style` to **query**, specify an `ID` that complies with the OA Framework File Standards, (all regions and items that you create must comply with these standards), and set the following properties:

- `Construction Mode` - set to **autoCustomizationCriteria**.
- `Include Simple Panel` - set to **True**.
- `Initial Panel` - if you include the Advanced Search and/or Saved Searches option, specify the initial display panel.
- `Simple Search Instructions` - instruction text for the simple search region.
- `Simple Button Label` - if you include the Advanced Search and/or Saved Searches option, specify the label of the button that OA Framework renders for toggling to the simple search. The default label is "Simple Search"

Step 3: Select your query region in the Structure pane, right-click and select **New > Region Using Wizard** to quickly create your table or HGrid search results regions. Remember to give the region a standards-compliant `ID` and specify appropriate attribute sets for the associated items. See the Tables and HGrid documentation for additional information about creating these components. (This is particularly important for the HGrid as there are additional properties that you must set on that component to properly enable searching).

Step 4: Create your custom simple search region.

- Step 4.1 Select your query region in the Structure panel, right-click and select **New > simpleSearchPanel**.
- Step 4.2 JDeveloper automatically creates two regions for you: a header with the `Text` property set to Simple Search, and beneath this, a messageComponentLayout. Give both of these regions a standards-compliant `ID`. If you are including only a simple search in your query region, change the header's `Text` property to Search, then, configure the messageComponentLayout to include your search criteria fields.
- Step 4.3 Set the `Selective Search Criteria` property to **True** for any search criteria items that ensure a performant query. See the OA Framework View Coding Standards for additional information.

  Note: OA Framework treats these values as being *individually* required: if the user enters any one of the required items, the query can proceed. You currently cannot require that the user enter search criteria in more than one of these required items.

  As shown in Figure 9 above, if the user fails to specify at least one of the required search values, an error message displays with the list of candidate items so the user can enter a value and proceed with the search.

Step 5: Create the query mappings between your custom Simple search region and the table columns so OA Framework can automatically execute the search when the user selects the **Go** button.

Note: If you need to perform the mapping programmatically (for example, you derive a date range WHERE clause based on a poplist value of "ANY_TIME," "LAST_2_WEEKS," "LAST_MONTH" and so on) skip this step and implement the search execution as described in Runtime Control: OAQueryBean below.

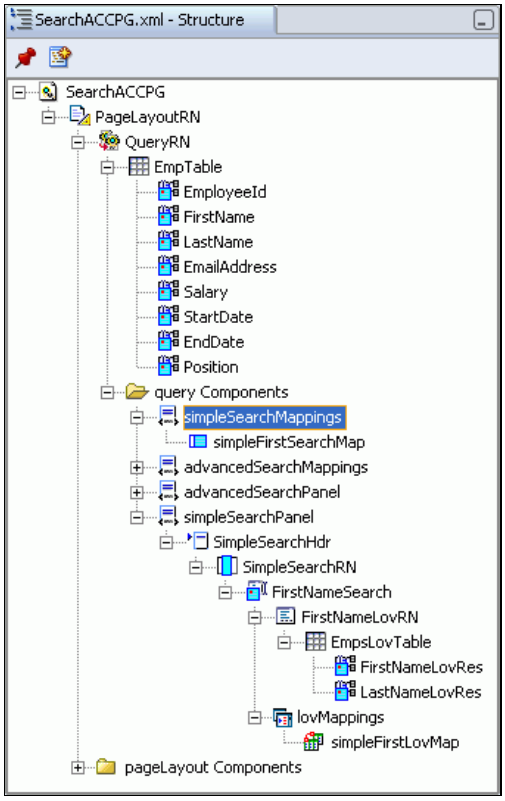
- Step 5.1 Select your query region in the Structure pane, right-click and select **New > simpleSearchMappings**.
- Step 5.2 Configure the default mapping that JDeveloper creates for you. This mapping identifies the item in the Search region whose value is used to query the corresponding item (table column) in the Results region.

Important: You are mapping search criteria values in this step, not display values. For example, the Search region includes an LOV that returns a unique key to a hidden form field and a display value to the LOV input field. You *must* map BOTH the hidden form field *and* the display value to its corresponding table columns.

- Step 5.3 (optional) If you have more than one mapping, select the simpleSearchMapping node in the Structure pane, right-click and select **New > queryCriteriaMap**. Configure the node as you did in Step 7.2. Repeat as needed until all your search criteria items are mapped to results table/HGrid items.

Figure 10 illustrates a Simple Search page structure created in Auto Customization Criteria mode.

Figure 10: Example of an Auto Customization Criteria Simple Search structure in the OA Framework ToolBox Sample Library.



Step 6: Save your work.

Manual Search

To build a search region that cannot be implemented with the Query bean (you should always use the Query bean if possible), follow these steps:

Step 1: Create the view object that you will use for searching and add it to your page's application module. Bind the result items to this view object.

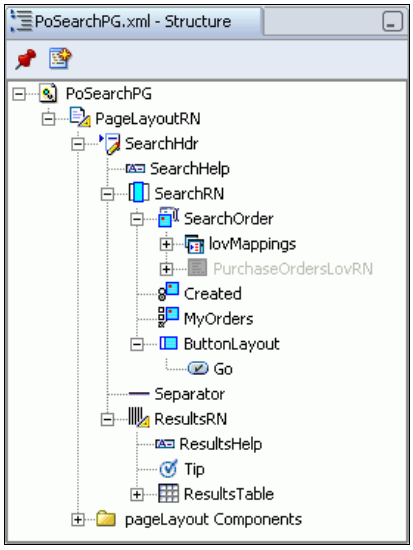
Step 2: Create the layout for your search region. Your region should have a header with the title "Search". For help with this task, see Page Layout (How to Place Content). See Hide/Show if you want to add a "Show More Options" to your search. The **Go** button should be a `submitButton` so that the form is submitted when the user selects it.

Step 3: Set the `Selective Search Criteria` property to **True** for any search criteria items that ensure a performant query. See the OA Framework View Coding Standards for additional information.

Note: OA Framework treats these values as being *individually* required: if the user enters any of the required items, the query can proceed. You currently cannot require that the user enter search criteria in more than one of these required items.

Step 4: Between your search region and your results region, add a "Separator" item. Figure 11 shows a typical Search page structure.

Figure 11: Example of manually configured Search region from the OA Framework ToolBox Tutorial



Step 5: Add an `initQuery()` method to your view object to handle binding the incoming search criteria to the view object's WHERE clause.

Note: Although you can manipulate the WHERE clause in your `initQuery()` method as well as binding search parameters, for performance reasons, you should ideally include a static WHERE clause in your view object definition. Make dynamic changes to the WHERE clause *only* if absolutely necessary due to the complexity and variety of the search criteria.

The following example from the `PoSimpleSummaryVOImpl` in the OA Framework ToolBox shows an appropriate use case for WHERE clause manipulation. It demonstrates the correct way to bind a dynamic number of parameters using Oracle-style binding. It also demonstrates the use of a `Boolean executeQuery` parameter to control whether this view object should explicitly query itself or defer to OA Framework's table handling. See View Objects in Detail -> Initialization Guidelines for additional information:

```
import java.util.Vector;
import oracle.bali.share.util.IntegerUtils;
import oracle.jbo.domain.Number;
import oracle.apps.fnd.framework.OANLSServices;
import oracle.apps.fnd.framework.server.OADBTransaction;
import oracle.apps.fnd.framework.server.OAViewObjectImpl;
...
public void initQuery(String orderNumber, String created, String showMyOrders, Boolean executeQuery)
{
  StringBuffer whereClause = new StringBuffer(100);
  Vector parameters = new Vector(3);
  int clauseCount = 0;
  int bindCount = 0;

  setWhereClauseParams(null);
  // Always reset
  // Note that the use of Oracle-style bindings, while requiring a bit
  // more effort than the ANSI-style binding, is REQUIRED of framework
  // code.

  if ((orderNumber != null) && (!("".equals(orderNumber.trim()))))
  {
    Number orderNum = null;

    try
    {
      orderNum = new Number(orderNumber);
    }
    catch(Exception e) {}

    whereClause.append(" ORDER_NUMBER = :");
    whereClause.append(++bindCount);
    parameters.addElement(orderNum);
    clauseCount++;
  }

  if ((created != null) && (!("".equals(created.trim()))) &&
        (!("ANY".equals(created))))
  {
    if (clauseCount > 0)
    {
      whereClause.append(" AND ");
    }

    whereClause.append(" CREATION_DATE >= :");
```

```
    whereClause.append(++bindCount);
    whereClause.append(" - :");
    whereClause.append(++bindCount);
    parameters.addElement(getClientSysdate());
    parameters.addElement(getDaysToSubtract(created));
    clauseCount++;
  }
  if ((showMyOrders != null) && (!("".equals(showMyOrders.trim()))))
  {
    // Ordinarily, you would set this value based on the current user.
    // Since the tutorial has its own data model for users, we'll
    // set this to the seeded buyer's ID.

    if (clauseCount > 0)
    {
      whereClause.append(" AND ");
    }

    whereClause.append(" BUYER_ID = :");
    whereClause.append(++bindCount);
    parameters.addElement(IntegerUtils.getInteger(6)); // 6 is the seeded buyer employee
    clauseCount++;
  }
  setWhereClause(whereClause.toString());

  if (bindCount > 0)
  {
    Object[] params = new Object[bindCount];

    // The copyInto() is 1.1.8 compliant which, as of 07/2004, is required by ARU

    parameters.copyInto(params);
    setWhereClauseParams(params);
  }

  if ((executeQuery != null) && (executeQuery.booleanValue()))
  {
    executeQuery();
  }
}

// end initQuery()
protected java.sql.Date getClientSysdate()
{

  OADBTransaction txn =
    (OADBTransaction)getApplicationModule().getTransaction();
  OANLSServices nls = txn.getOANLSServices();

  oracle.jbo.domain.Date serverDate = txn.getCurrentDBDate();

  java.util.Date javaClientDate = nls.getUserDate(serverDate);
  long longDate = javaClientDate.getTime();

  return new java.sql.Date(longDate);

} // end getclientSysdate()

protected Integer getDaysToSubtract(String created)

{
  int days = 0;
  if ("TODAY".equals(created))
  {
    days = 0;
  }
    else if ("THIS_WEEK".equals(created))
  {
    days = 6;
  }
    else if ("LAST_30_DAYS".equals(created))
  {
    days = 29;
  }
    else if ("LAST_60_DAYS".equals(created))
  {
    days = 59;
  }

  return IntegerUtils.getInteger(days);
```

```
}  // end getDaysToSubtract()
```

Step 6: Add an `initQuery()` method to your application module that delegates to your view object's `initQuery()` method. Give this method an explicit name if your application module has multiple query initialization methods. For example, `initEmployeeSummary()`, `initEmployeeDetails()`, `initDeparementSummary()`, `initDepartmentDetails()` and so on. The application module `initQuery()` method that delegates to the `initQuery()` method shown in the previous steps looks like:

```
public void initSummary(String orderNumber,
                        String created,
                        String showMyOrders,
                        Boolean executeQuery)
{
  PoSimpleSummaryVOImpl vo = getPoSimpleSummaryVO1();

  if (vo == null)
  {
    MessageToken[] tokens = { new MessageToken("OBJECT_NAME","PoSimpleSummaryVO1")};
    throw new OAException("AK", "FWK_TBX_OBJECT_NOT_FOUND",tokens);
  }

  vo.initQuery(orderNumber, created, showMyOrders, executeQuery);

} // end initSummary()
```

Step 7: Add controller `processFormRequest()` logic to handle the search region's **Go** button press.

The following is an example of how to call the `initSummary()` method, which is shown in the previous step. Since our query results are displayed in a table, note the call to the `queryData()` method at the bottom of the controller. (The reasons for this are fully described in View Objects in Detail -> Initialization Guidelines).

```
import oracle.apps.fnd.framework.OAApplicationModule;

import oracle.apps.fnd.framework.webui.OAQueryUtils;
...

public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{

  super.processFormRequest(pageContext, webBean);

  // If the "Go" parameter value is not null, the user selected the "Go"  submitButton.

  if (pageContext.getParameter("Go") != null)
  {
    // To complete the Selective Search Criteria implementation that you started above,
    // you must also perform the following test. Note that the webBean parameter
    // that you pass to the  checkSelectiveSearchCriteria()
    // method MUST be the direct parent region containing the items for which you have
    // set the  Selective Search Criteria  property to True. If this test fails, the
    // OA Framework displays the exception message described above and your query
    // is not executed.
    OAQueryUtils.checkSelectiveSearchCriteria(pageContext, webBean);

    // Get the user's search criteria from the request.
    String orderNumber = pageContext.getParameter("SearchOrder");
    String created = pageContext.getParameter("Created");
    String showMyOrders = pageContext.getParameter("MyOrders");

    OAApplicationModule am = getApplicationModule(webBean);

    // Note the following is required for view object initialization standards
    // around tables.
    Boolean executeQuery = BooleanUtils.getBoolean(false);
    Serializable[] parameters =  { orderNumber, created, showMyOrders, executeQuery };

    // Since the parameters that we're passing include a non-String type, you must
    // call the version of invokeMethod() that lets you specify the individual
    // parameter types as shown.
    Class[] paramTypes = { String.class, String.class, String.class, Boolean.class };
    am.invokeMethod("initSummary", parameters, paramTypes);

    OAAdvancedTableBean table =
      (OAAdvancedTableBean)webBean.findChildRecursive("ResultsTable");

    // When handling a user initiated search, we always need to execute
    // the query so we pass "false" to queryData().
    table.queryData(pageContext, false);

  }
```

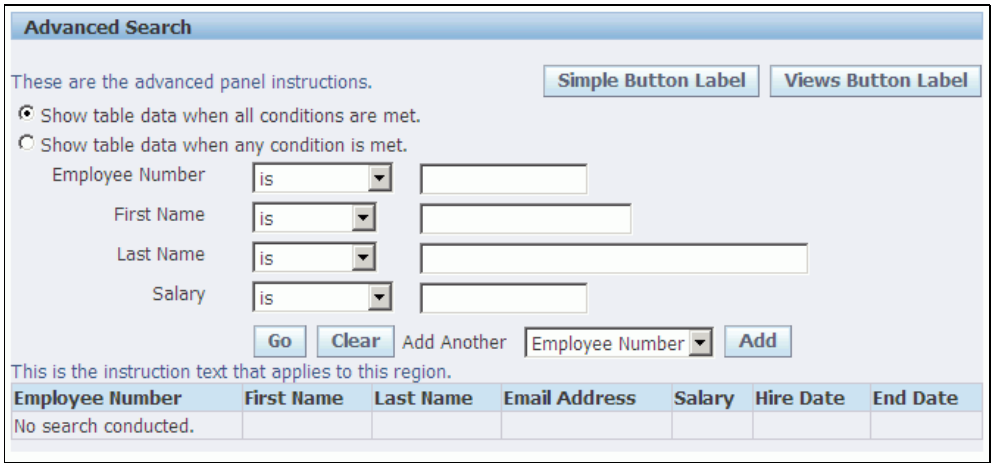
```
}
```

Advanced Search

As described above, an advanced search displays extensive search criteria and power search capabilities, including the ability to declaratively specify "AND" or "OR" searches and datatype-specific operators, such as "greater than", "is", "is not" or "less than", on individual fields. For example, a "Salary" value must be greater than 100000. As of Release 12.2.4, advanced search now allows you to specify the "is not (include blanks)" operator to also return null values from a search.

Note: OA Framework implements this search region using the `oracle.apps.fnd.framework.webui.beans.layout.OAAdvancedSearchBean`. The resulting UI appears as shown in Figure 12.

Figure 12: Example of an Advanced Search panel from the OA Framework ToolBox Sample Library



- To implement searches on items that map directly to the associated table columns, you can leverage the Query bean to quickly configure a fully automated search. See the Declarative Implementation: Results Based Search for instructions.
- To implement a standard advanced search with the ability to manipulate the search itself, such as an intermedia search, use the Query bean with your custom query handling. See the Declarative Implementation: Auto Customization Criteria.
- To implement an Advanced Search layout that differs from the OA Framework `OAAdvancedSearchBean` layout, you *must* implement it manually.

Note for Oracle's internal E-Business Suite developers: If you need to implement an Oracle interMedia search, contact the Oracle E-Business Suite Performance Tuning team.

Declarative Implementation: Results Based Search

To create an advanced search panel in the results based search mode:

Step 1: Create the view object that you will use for searching. In this mode, there is no need to create an `initQuery()` method on the view object to set the WHERE clause, bind the search criteria, and execute the query. OA Framework handles this on your behalf.

Tip: If your view object's definition requires bind values, or you *must* manually append an unrelated WHERE clause every time the query is executed, you can proceed in one of two ways. In either scenario, OA Framework appends the user's search criteria to the WHERE clause immediately before executing the query, so any WHERE clause characteristics that you define are preserved.

1. Override the `executeQuery()` method in your `*VOImpl` to modify your WHERE clause and then call `super.executeQuery()`. This is recommended if you exercise the same logic every time the view object is queried.
2. Handle the generated **Go** button's press and call an `initQuery()` method on your view object where you can modify the WHERE clause and/or set bind values.

Step 2: Select your pageLayout region in the JDeveloper Structure pane, right-click and select **New > Region**. Set the `Style` to **query**, specify an `ID` that complies with the OA Framework File Standards, (all regions and items that you create must comply with these standards), and set the following properties:

- `Construction Mode` - set to **resultsBasedSearch**.
- `Include Advanced Panel` - set to **True**.
- `Initial Panel` - if you include the Simple Search and/or Saved Searches option, specify the initial display panel.
- `Advanced Panel Instructions` - specify the instruction text for the advanced search region.
- `Advanced Button Label` - if you include the Simple Search and/or Saved Searches option, specify the label of the button that OA Framework renders for toggling to the simple search. The default label is "Advanced Search".

- `Advanced Search Criteria Count` - as of Release 12.2.6, specify the number of criteria rows to display in the Advanced Search panel. The default count is 4. This property is personalizable.

Step 3: Select your query region in the structure pane, right-click and select **New > Region Using Wizard** to quickly create your table or HGrid search results regions. Give the region a standards-compliant `ID` and specify appropriate attribute sets for the associated items. See the Tables and HGrid documentation for additional information about creating these components. (This is particularly important for the HGrid as there are additional properties that you must set on that component to properly enable searching).

Note: In the Results Based Search mode, OA Framework automatically generates an "Advanced Search" header at runtime to contain the query region. There is no need to manually add this to the component hierarchy.

Step 4: Select the table/HGrid region that you created in Step 3 and set the `Search Allowed` property to **True** for any items that are used as search criteria. These items are automatically included in the advanced search region's search criteria poplists, and OA Framework automatically executes the corresponding query when the user selects the **Go** button.

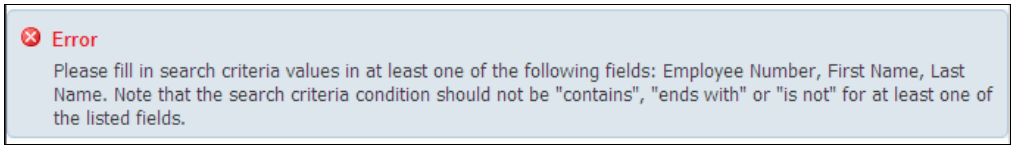
Step 5: Set the `Selective Search Criteria` property to **True** for any search criteria items that ensure a performant query. See the OA Framework View Coding Standards for additional information.

Note: OA Framework treats these values as being *individually* required: if the user enters any one of the required items, the query can proceed. You currently cannot require that the user enter search criteria in more than one of these required items.

As shown in Figure 13 below, if the user fails to specify at least one of the required search values, an error message displays with the list of candidate items so the user can enter a value and proceed with the search.

Note: The error message for the advanced search differs from the simple search.

Figure 13: Advanced search selective search criteria error message (note that the actual error message may differ slightly).



Step 6: Save your work.

Note: If you create a results based search, do not add custom search regions as required for the "auto customization criteria" mode described next.

Declarative Implementation: Auto Customization Criteria

To create an Advanced Search panel in the auto customization criteria mode:

Step 1: Create the view object that you will use for searching as described in the Results Based Search section above.

Step 2: Select your pageLayout region in the JDeveloper Structure pane, right-click and select **New > Region**. Set the region's `Style` to **query**, specify an `ID` that complies with the OA Framework File Standards, (all regions and items that you create must comply with these standards), and set the following properties:

- `Construction Mode` - set to **autoCustomizationCriteria**.
- `Include Advanced Panel` - set to **True**.
- `Initial Panel` - if you include the Simple Search and/or Saved Searches option, specify the initial display panel.
- `Advanced Panel Instructions` - specify the instruction text for the advanced search region.
- `Advanced Button Label` - if you include the Simple Search and/or Saved Searches option, specify the label of the button that the OA Framework renders for toggling to the simple search. The default label is "Advanced Search".
- `Advanced Search Criteria Count` - as of Release 12.2.6, specify the number of criteria rows to display in the Advanced Search panel. The default count is 4. This property is personalizable.

Step 3: Select your query region in the Structure pane, right-click and select **New > Region Using Wizard** to quickly create your table or HGrid search results regions. Assign the region a standards-compliant `ID` and specify appropriate attribute sets for the associated items. See the Tables and HGrid documentation for additional information about creating these components. (This is particularly important for the HGrid as there are additional properties that you must set on that component to properly enable searching).

Step 4: Create your custom advanced search region.

- Step 4.1 Select your query region in the Structure pane, right-click and select **New > advancedSearchPanel**. JDeveloper automatically creates a header region with the `Text` property set to Advanced Search. Give this region a standards-compliant `ID`.
- Step 4.2 Select the header region in the Structure pane, right-click and select **New > advancedSearch**. Give the resulting advancedSearch region a standards-complaint `ID`. JDeveloper creates a Criteria node and a default criteriaRow node with two items beneath it.
- Step 4.3 Configure the default criteriaRow for the first search criteria value you want to display:

- Give the criteriaRow node a meaningful `ID` as shown in Figure 14 below.
- In the first item that JDeveloper creates for you, specify the search criteria `Prompt`.
- Configure the second **messageTextInput** item for the criteria value. Specify a standards-compliant `ID` and apply an appropriate `Attribute Set`. Make sure you select the correct `Data Type` value as this drives the "Conditions" poplist that OA Framework creates for you. Also, set the `Selective Search Criteria` property to **True** for any search criteria items that ensure a performant query. See the OA Framework View Coding Standards for additional information.

  Note: OA Framework treats these values as being *individually* required: if the user enters any one of the required items, the query can proceed. You currently cannot require that the user enter search criteria in more than one of these required items.

  Finally, set the `Addtional Text` value to the same value that you set for the search criteria `Prompt`. This is required per the Accessibility guidelines).

  As shown in Figure 13 above, if the user fails to specify at least one of the required search values, an error message displays with the list of candidate items so the user can enter a value and proceed with the search.

- Step 4.3 If you have additional search criteria, select the Criteria node in the Structure pane, right-click and select **New > criteriaRow** for each search criteria item that you want to add. Repeat step 4.3 to configure each criteria row.

Step 5: Create the query mappings between your custom advanced search region and the table columns, so that OA Framework can automatically execute the search when the user selects the **Go** button.

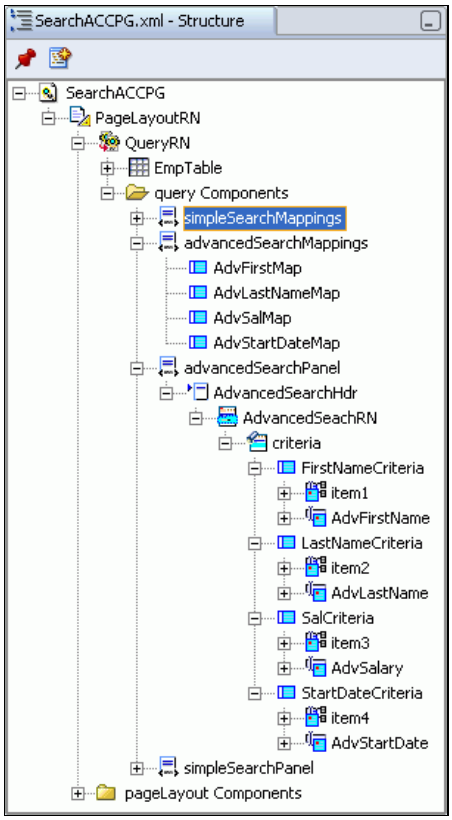
- Step 5.1 Select your query region in the Structure pane, right-click and select **New > advancedSearch Mappings.**
- Step 5.2 Configure the default mapping that JDeveloper creates for you. This mapping identifies the item in the search region whose value is used to query the corresponding item (table column) in the results region.

  Important: You are mapping search criteria values in this step, not display values. For example, the search region includes an LOV that returns a unique key to a hidden form field and a display value to the LOV input field. You must map BOTH the hidden form field and the display value to its corresponding table columns.

- Step 5.3 (optional) If you have more than one mapping, select the advancedSearchMapping node in the Structure pane, right-click and select **New > queryCriteriaMap**. Configure the node as you did in Step 5.2. Repeat until all your search criteria items are mapped to results table/HGrid items.

Figure 14 illustrates an Advanced Search page structure created in Auto Customization Criteria mode.

Figure 14: example of an Auto Customization Criteria Advanced Search structure in the OA Framework ToolBox Sample Library



Step 6: Save your work.

Manual Search

To build an advanced search region that cannot be implemented with the Query bean, (you should always use the query bean if possible), follow the steps outlined in the Simple Manual Search section above.

Runtime Control

This section discusses:

- Custom Validations
- Handling processFormRequest
- Setting the Default Criteria Count

Custom Validations

Two new constants `CUST_VIEW_UPDATE_APPLY` and `CUST_VIEW_APPLY_VIEW_RESULTS` have been added in `OAWebBeanConstants` for holding the ID of the **Apply** and **Apply and View Results** buttons. To see if the buttons have been clicked in the Create /Update Saved Search page, check for these attributes in the controller attached to `AdvancedSearchRegion` using `pageContext.getParameter(CUST_VIEW_UPDATE_APPLY)` or `pageContext.getParameter(CUST_VIEW_APPLY_VIEW_RESULTS)`.

Custom validations *must* be performed in the `processFormData` method of the controller attached to the AdvancedSearchRegion. This is because by the time the `processFormRequest` method is called, the search will be saved and any exceptions thrown from the `processFormRequest` method will not prevent the search from getting saved.

The following sample code can be used to throw an exception if a value is not entered in the first search item in the `AdvancedSearchRegion` before clicking the **Apply** or **Apply and View Results** button:

```
public void processFormData(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormData(pageContext, webBean);
  if( pageContext.getParameter(CUST_VIEW_UPDATE_APPLY) != null ||

  pageContext.getParameter(CUST_VIEW_APPLY_VIEW_RESULTS) != null )
  {
    OAMessageTextInputBean mtiBean = (OAMessageTextInputBean)
    webBean.findChildRecursive("Value_0");
    if( mtiBean.getValue(pageContext) == null)
    {
      throw new OAException("Value should be entered for Item1",
      OAException.ERROR);
    }
  }
}
```

Handling processFormRequest

Generally, you need to handle the form submits of any buttons seeded in your region items. The only button you do not need to handle is the **Add** button, which the advanced search web bean handles.

Since duplicate items can be added to the list of displayed items and the list changes dynamically, use the following naming convention to identify the web beans in the advanced search region:

- The item name of the AND/OR radio group is `OAWebBeanConstants.ADVANCED_SEARCH_RADIO_GROUP`. It returns a value of `AND` or `OR` based on what is chosen.
- All conditions (conditions poplist) displayed on the page have `UINodeNames` set as `Condition_0` to `Condition_n`.
- All actual web beans, that are items in your region, have their `UINodeNames` set to `Value_0` to `Value_n`.
- Use the `getOriginalUINodeName` methods on the advanced search bean to retrieve the original item name of each web bean.
- You can use the method `getDisplayedCriteriaCount` on the advanced search bean to get the number of conditions/criteria displayed on the page at any point in time.
- The conditions poplists derive their values from the FND lookup `ICX_CONDITIONS`, where `LookupCode` is the value attribute and `Meaning` is the display attribute. You can use the method `getCondition` on `oracle.apps.fnd.framework.OAFwkUtils` to get the SQL equivalent of the condition passed. For example, if your conditions poplist returns a value attribute of `CCONTAINS` (and a display attribute of `Contains`), then `OAFwkUtils.getCondition("CCONTAINS")` will return the string "like" which you can use directly in your WHERE clause.

To handle region-specific button submits, the `processFormRequest` method in your advanced search web bean controller should look similar to the example code below:

```
public void processFormRequest (OAPageContext pageContext, OAWebBean webBean)
{
  // get the displayed criteria count first.
```

```
  int displayedCriteriaCount = webBean.getDisplayedCriteriaCount();

  // The following gives you the AND/OR radio group value

  String radioGroupValue = pageContext.getParameter
    (OAWebBeanConstants.ADVANCED_SEARCH_RADIO_GROUP);
  for (int i=0; i < displayedCriteriaCount; i++)
  {
    String newItemName = OAWebBeanConstants.VALUE +
    Integer.toString(i); //returns a string like "Value_0"
    String conditionName = OAWebBeanConstants.CONDITION +
    Integer.toString(i); //returns a string like "Condition_0"

    //first check if some value and condition was entered
    //by the user for this item.

    if ((pageContext.getParameter(newItemName) != null) &&
      (pageContext.getParameter(conditionName)!= null))
    {
      //this gives you the original item name that you chose in AK.
      String originalChildName = webBean.getOriginalUINodeName(newChildName);

      //In order to get the "SQL condition" used you can use
      //the following method
      // this returns a string like: "=", or " like " etc.
      String condition = OAFwkUtils.getCondition(conditionName);

      //This gives you the value chosen
      String value = pageContext.getParameter (newItemName);

      //In order to get the "SQL" value, you can use the following method
      //So, if you had a condition of "CCONTAINS" (like) and a value of
      //"XYZ" this method would return "%XYZ%";
      // This can be used to directly bind to your whereClause.
      value = OAFwkUtils.getValue(conditionName, value);

      //Now that you have the condition, value and you know
      //the child, handle your product specific logic here.
    }
  }
}
```

Concerning pages with multiple query regions, (which includes more than one query region defined under the same subtab layout or under different subtabs in the same page). The second and subsequent query beans have their queryBean's region key prepended to the `Value_n`. If the call `nonFirstQueryBean.findChildRecursive("Value_ ")` is made, it will return **null**. Product teams are required to handle the second and subsequent multiple query region `Value_n` being null by using the following logic:

```
OAQueryBean nonFirstQueryBean = (OAQueryBean)webBean.findChildRecursive(<nonFirstQueryRN>);
OAAdvancedSearchBean advBean =
(OAAdvancedSearchBean)nonFirstQueryBean.findChildRecursive("advancedSearchRN");
nonFirstQueryBean.findChildRecursive(advBean.getRegionKey(advBean,...)+"Value_'<n>'");
```

Setting the Default Criteria Count

As of Release 12.2.6, you may programmatically modify the default number of criteria to display in the Advanced Search panel (for both results based search and auto customization criteria mode) by calling a new API on the query bean as shown in the following code example:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   super.processRequest(pageContext, webBean);
     OAQueryBean query1 =
       (OAQueryBean)webBean.findIndexedChildRecursive("region2");
     if(query1!=null)
        {
        query1.setDisplayedCriteriaCount(3);
        }
}
```

(User Personalizable) Saved Searches

To support user-personalizable searches, which are surfaced in a Saved Searches panel, you must leverage the query region as described below.

Note: The first time the user accesses a saved searches-enabled page (no saved searches to run) OA Framework does the following:

- If there is only a Saved Searches panel, this displays with an empty Saved Searches poplist. The user is expected to select the Personalize button to create one or more saved searches.
- If there is a Saved Searches panel and a simple search, the simple search renders. The user is expected to execute a query and select the **Save Search** button.

- If there is a Saved Searches panel and an advanced search, the advanced search renders. The user is expected to execute a query and select the **Save Search** button.

Implementation

Step 1: Create the view object that you will use for searching. In this mode, there is no need for you to create an `initQuery()` method on the view object to set the WHERE clause, bind the search criteria, and execute the query. OA Framework handles this on your behalf.

Tip: If your view object's definition requires bind values, or you *must* manually append an unrelated WHERE clause every time the query is executed, you can proceed in one of two ways. In either scenario, OA Framework appends the user's search criteria to the WHERE clause immediately before executing the query, so any WHERE clause characteristics that you define are preserved.

1. Override the `executeQuery()` method in your `*VOImpl` to modify your WHERE clause and then call `super.executeQuery()`. This is recommended if you exercise the same logic every time the view object is queried.
2. Handle the generated **Go** button's press and call an `initQuery()` method on your view object where you can modify the WHERE clause and/or set bind values.

Step 2: Select your pageLayout region in the JDeveloper Structure pane, right-click and select **New > Region**. Set the `Style` to **query**, specify an `ID` that complies with the OA Framework File Standards, (all regions and items that you create must comply with these standards), and set the following properties:

- `Construction Mode` - This has no effect in the Saved Searches panel and applies to the Simple and the Advanced search panels only.

  Note: When `Construction Mode` value is **None**, the Criteria section is hidden in the Create Saved Searches page.
- `Include Saved Searches Panel` - set to **True**.
- `Initial Panel` - if you include the Simple Search or Advanced Search options, specify the initial display panel.
- `Saved Searches Button Label` -if you include the advanced search or simple search, specify the label of the button that OA Framework renders for toggling to the saved searches region
- `Saved Searches Panel Instructions` - specify the instruction text for the saved searches region
- `Saved Searches Panel Title` - specify an alternate title if required. The default title is "Saved Searches".
- `Save Search Button Text` - specify the text to display on the **Save Search** button. The default label is "Save Search".

Step 3: Select your query region in the structure pane, right-click and select **New > Region Using Wizard** to quickly create your table or HGrid search results regions. Assign the region a standards-compliant `ID` and specify appropriate attribute sets for the associated items. See the Tables and HGrid documentation for additional information about creating these components. (This is particularly important for the HGrid as there are additional properties that you must set on that component to properly enable searching).

Step 4: Select the table/HGrid region that you created in Step 3 and set its `User Personalization` property to **True**.

Step 5: Select the table/HGrid region that you created above and set each item's `User Personalization` property to **True** if the item should be personalizable. For example, the user can configure its display properties in the underlying table/HGrid and specify search criteria for the associated column value. Set its `Search Allowed` property to **True** if you want the user to be able to filter the search on the item's value.

Note: If the item is administrator personalizable, the user can change the `Search Allowed` property to **True** even if you set it to **False**.

Step 6: Save your work.

Step 7 (optional): If you need to do special search processing, such as binding additional query criteria, handle this in `processFormRequest()` as shown in the Runtime Control section below. If, however, the user creates a new saved search and chooses to immediately use it by selecting the **Apply and View Results** button in the Personalization module, you also need to add the following `processRequest()` logic that checks the `CUST_ANVR_VIEWID` attribute value:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
  ...
  OAQueryBean queryBean =
      (OAQueryBean)pageContext.findIndexedChildRecursive("Query");
  String viewId = queryBean.getAttributeValue(OAWebBeanConstants.CUST_ANVR_VIEWID);

  // If viewId is not null, then the user selected the "Apply and View Results"
  // button and the new search she just created is now the current saved search. Note that
  // this is the same viewId value that is returned when you call
  // queryBean.getCurrentCustomization() in processFormRequest().  If the
  // viewId value is null, this is the normal page flow.
  if (viewId != null)
  {
    ...
  }
}
```

Step 8: Make sure the Disable Self-service Personal profile option is set to **No** at the site or application level as appropriate for your page. If this is set to **Yes**, the Saved Searches option will not be accessible to the user even if it is properly enabled. (If the search region does not include either Simple or Advanced Search options, the Saved Searches option will render but the Saved Searches poplist will be empty).

Note: When a user selects a personalized saved search to display, OA Personalization does not retain the current saved search selection for the Query page when you navigate to another page. For example, suppose you display a personalized saved search of the Query page by selecting a saved search (such as Saved Search X) from the Saved Searches panel. Then you select the **Personalize** button to display the Personalize Saved Searches page, but choose **Cancel** without making any changes. When you return to the Query page, the base definition is displayed and not Saved Search X. If you want to retain the saved search, you must modify the controller for the page to pass the user-selected saved search as a parameter and set that saved search as the current personalization whenever the user returns to the Query page.

Non-Personalizable Saved Searches

To display a Saved Searches panel on your Search page that only displays seeded saved searches, (either Oracle-level or Admin-level), and does not allow users to create their own personalizable saved searches, follow the implementation instructions described above to create a (User Personalizable) Saved Searches panel. Then programmatically hide the **Save Search** button on the Search panel and the **Personalize** button on the Saved Searches panel using the following `OAQueryBean` APIs in the controller's `processRequest` method:

- `hideSaveSearchButton(boolean hide)`
- `hidePersonalizeButton(boolean hide)`

By default, the value for `hide` is set to **False** and the **Save Search** and **Personalize** buttons are shown.

Note: Release 12.1 introduced a new feature that allows the same functionality to hide the buttons through Administrative Personalizations instead of using the APIs via the processRequest method.

Shareable Saved Searches

OA Framework allows you to extend a user-personalizable query region so that it is shared by multiple pages. Previously, only personalization administrators were allowed to create a personalized saved search of a shared region that was also visible to all the pages that shared that region. A personalized saved search of a shared region created by an end user, however, was always saved as a per instance saved search and was visible only in the page from which the saved search was created/updated. As of Release 12, you can now set a property on the query region that allows a saved search created by an end user to be stored directly on the shared region, rather than as a per instance saved search. This allows the end-user to see the personalized saved search in all pages where the query region is shared.

Note: *You may ONLY enable the shareable saved searches feature for new query regions that you create in Release 12.*

To enable this feature, call the `oracle.apps.fnd.framework.webui.beans.layout.OAQueryBean storeViewsOnSharedRegion` API as follows: (This API needs to be set on each query region as defined on each page that extends it.)

```
queryBean.storeViewsOnSharedRegion(true);
```

This API stores any saved search created by the end-user on the shared query region. As a result, any change a user makes to a saved search (create, update or delete) of that query region will be reflected on all the pages that share that query region.

Restrictions

- This feature was introduced in Release 12, and should only be enabled for new pages that do not have any existing per instance saved searches created for it. This generally means that the newly created query region has never shipped to customers previously. OA Framework provides no backward-compatibility support of this feature for previously shipped query regions in earlier releases, which may have existing per instance saved searches created for it. It is therefore mandatory that you not implement this feature on pre-existing query regions.
- When this feature is enabled for a shared query region, a saved search created by an end-user will be stored on the shared region. OA Framework does not support personalization administrators creating a per instance saved search of that shared region. As a result, a personalization administrator should refrain from creating a per instance saved search on a shared query region that has the shareable saved searches feature enabled. In other words, when an administrator creates a personalization, he or she should set the scope of the personalization context to the shared region and not to the page itself. **It is the responsibility of the product team that implements this feature to document this restriction for its users.**
- OA Framework throws an error if you or a personalization administrator ignores either of the previous two restrictions. For example:
  - If you still attempt to implement the shareable saved searches feature on a page that has previously shipped to customers (and has existing per instance saved searches created for it at the customer site), and someone tries to duplicate one of the existing per instance saved searches on the page where the shareable saved searches feature is implemented, OA Framework throws an error, warning that it detects this discrepancy.
  - If you enable the shareable saved searches feature for a shared region and a personalization administrator ignores your documented restriction and proceeds to create a per instance saved search of the shared query region (where the personalization context scope is set to Page), OA Framework throws an error when an end-user attempts to duplicate the per instance saved search that the administrator created.

Usage Notes

Rows to Display - If you define a query region that includes a Saved Searches panel, end users of that query region will be able save their search and personalize their search results as saved searches. In defining their personalized saved searches, they can specify the number of rows to display in the search results table. If you want to override the number of rows to display, as defined by the view, and instead, use the

value set in the base metadata of the actual table or advanced table, or to a programmatically-set value, you can use the attribute `BYPASS_NOR_ON_VIEW` in the `oracle.apps.fnd.framework.webui.OAWebBeanConstants` interface. Set the `BYPASS_NOR_ON_VIEW` attribute programmatically on the query web bean's child table or advanced table. By using this attribute, the search results table will bypass the number of rows to display as set by the view, and render the number of rows to display as specified by the metadata of the table or advanced table.

OAQueryBean Behavior in Detail

OA Framework generates the WHERE clause automatically for your searches in both the results based search and the auto customization criteria modes. It also generates the WHERE clause automatically for the criteria entered through the Personalization Framework on the Saved Searches panel in the two modes above.

This section discusses the rules used by OA Framework to generate your WHERE clause and the various methods that you can use to override it.

Criteria Object Overview

OA Framework stores the metadata information required for generating your WHERE clause in a `criteria` object. The `criteria` object is an array of `java.util.Dictionary`. Each element in the array identifies a *criterion* or a portion of your WHERE clause. Each criterion is comprised of the following name-value pairs. All constants are available on the `oracle.apps.fnd.framework.OAViewObject` class.

Constant Name	Value
`CRITERIA_ITEM_NAME`	The item name or id of the item that the criterion is defined for.
`CRITERIA_VIEW_ATTRIBUTE_NAME`	The view attribute name of the item that the criterion is defined for.
`CRITERIA_CONDITION`	The actual condition that is used for binding in the SQL query. Examples: " = ", " like " etc
`CRITERIA_VALUE`	The transformed value of the item as computed by OA Framework based on the condition chosen by the user.
`CRITERIA_JOIN_CONDITION`	The join condition used to join the various criterion within a WHERE clause. Its value is set to "AND" if "Match All" is used and "OR" if "Match Any" is used in the Advanced Search panel. For the Simple Search panel, its value is set to "AND".

Note: Beginning in Release 12.2.6, you may hide the join condition "Match All / Any" by calling the API shown below. This hides the "Match All / Any" radio group shown in the Advanced Search panel. The default search condition will be "ALL" when you hide the join condition for the query bean. You can only hide the join condition programmatically. This feature does not have a corresponding declarative property and is not personalizable.

```
queryBean.setMatchConditionHidden(true);
```

The table below describes how OA Framework computes the `CRITERIA_VALUE` and the `CRITERIA_CONDITION` based on the condition chosen and the value entered by the user in the Advanced Search panel.

Note: Of the various options below, the Simple Search panel uses the *starts with* option internally for Strings and the *is* option for Numbers and Dates.

Condition (as chosen by the user)	Applicable Data types	CRITERIA _CONDITION (computed by the OA Framework)	Value (as entered by the user)	CRITERIA_VALUE (computed by the OA Framework)
is	String, Number, Date	=	Jamie Frost	"Jamie Frost"
is not	String	<> (returns only non-null values)	Jamie Frost	"Jamie Frost"
starts with	String	like	Ja	"Ja%"
ends with	String	like	Ja	"%Ja"
contains	String	like	Ja	%Ja%
before	Date	<	1/12/03	1/12/03
after	Date	>	1/12/03	1/12/03

greater than	Number	>	123	123
less than	Number	<	123	123

Default WHERE Clause Generation

OA Framework generates the WHERE clause using the criteria object described above and adds it to any other WHERE clause that you may already have using an AND condition.

The WHERE clause is generated using the rules below. The rules are explained using an example of a criteria object that is used in the "Projects" search pages.

Note: The Search can be made case sensitive by calling setCaseSensitive(true) method on the query bean in the controller. `queryBean.setCaseSensitive(true);` and testing using isCaseSensitive().

The criteria object can be represented in a tabular form as:

CRITERIA_ ITEM_NAME	Selective Search Criteria	CRITERIA_ VIEW_ ATTRIBUTE_ NAME	CRITERIA_ CONDITION	CRITERIA_ VALUE	ACTUAL DB COLUMN NAME
projectId	false	projectId (data type: NUMBER)	=	"101"	PROJECT_ID
projectStatus	false	projectStatus (data type: VARCHAR2)	like	"%Approved%"	PROJECT_STATUS
projectName	true	projectName (Data type VARCHAR2)	like	"Security%"	SECURITY
startDate	false	startDate (Data type DATE)	>	"1/1/04"	START_DATE

Rule 1: If your `viewattribute` is of type Number, Date, or DateTime, then OA Framework generates a simple WHERE clause using the `CRITERIA_CONDITION` based on the `CRITERIA_VALUE`. So, in the example above, the WHERE clause for the "projectId" view attribute is generated as:

```
PROJECT_ID = 101
```

and the WHERE clause for the "startDate" view attribute is generated as:

```
START_DATE > "1/1/03"
```

OA Framework finds the right database column that corresponds to a view attribute even if you use aliases.

that the index is used.
i.e. generation of the WHERE clause is independent of Search Criteria property being set to false or true

Overriding the WHERE Clause Generation

To generate a WHERE clause that is different from the one generated by OA Framework, use the `getCriteria()` and the `getNonViewAttrCriteria()` methods on `OAQueryBean`. Due to timing issues, use these methods in the controller associated with the query region only.

The methods are different in the following ways:

	`getCriteria()`	**`getNonViewAttrCriteria()`**
Generating WHERE clause	You will get a handle to the entire criteria dictionary.	You will get a handle only to a subset of the dictionary (criteria of the items that do not have view attributes)
Executing the query	Call `executeQuery()` after building the criteria.	OA Framework adds any other additional WHERE clauses and executes the query for you. Do not call `executeQuery`() in this case.

Follow these steps to use one of the methods above:

Step 1: Determine the method to use.

Determine whether you want to handle the entire criteria or the criteria for a subset of attributes that don't really belong to the table, but is something that you would like to search on. Use the `getCriteria()` method for the former case and the `getNonViewAttrCriteria()` method for the latter case.

Step 2: Define the searchable field in the search panels.

Define the item as a searchable field in the Simple or Advanced search panels. For an example, define the following fields:

- `projectIdSimpleSearch` on the Simple Search panel.
- `projectIdAdvSearch` on the Advanced Search panel.

Step 3: Using the `getNonViewAttrCriteria()` method.

Modify the above example and state that projectId is a field that is not present on the viewobject, not displayed in the results table, but something you want to search on.

Option 1 - Use this option if you want to search on a specific field, but *not* save its search as a user personalization view.

Process this field in the Simple or Advanced search panel only.

Step 3.1: Do not define any simple or advanced search mappings for this field.

Step 3.2: Handle the **Go** button press in your query region's controller's `processFormRequest()` using:

```
processFormRequest (OAPageContext pageContext, OAWebBean webBean)
{
  // Make sure that you are in the right panel
  OAQueryBean queryBean = (OAQueryBean) webBean;
  queryBean.getCurrentSearchPanel();
  // Handle the "Go" button click on the simple search panel
  if (SEARCH.equals (currentPanel) &&
  pageContext.getParameter(queryBean.getGoButtonName()) != null)
  {
     // retrieve your item and process it here.

     String value = pageContext.getParameter (<projectIdSimpleSearch>);

     // Build the appropriate where clause for projectIdSimpleSearch
     // and set it on your vo and execute its query
     // using the initQuery method.
     ...
  }

  // Handle the "Go" button click on the advanced search panel
  if (ADVANCED_SEARCH.equals (currentPanel) &&
  pageContext.getParameter(queryBean.getGoButtonName()) != null)
  {
     // retrieve your item and process it here.

     String value = pageContext.getParameter(<projectIdAdvSearch>);

     // build the appropriate where clause for projectIdAdvSearch
     // and set it on your vo and execute its query
     // using the initQuery method.
     ...
  }
}
```

The table below lists the constants that you can use to determine your current search panel. All constants are defined in the `oracle.apps.fnd.framework.webui.OAWebBeanConstants` class.

Constant Name	Current Panel
`OAWebBeanConstants.SEARCH`	Simple Search panel
`OAWebBeanConstants.ADVANCED_SEARCH`	Advanced Search panel
`OAWebBeanConstants.CUSTOMIZE`	Saved Searches panel.

Option 2 - Use this option if you want to search on a specific field and save its search as a user personalization saved search.

Process this field in the Saved Searches panel in addition to the Simple or Advanced search panel.

Step 3.1: Define an associated hidden field (`oa:formValue`) in the table region, with its `userCustomizable` property set to **true** and `viewUsageName` and `viewAttributeName` set to null.

For our example, define a hidden field with its ID set to `projectId` in the table region.

Step 3.2: Add the appropriate mappings for this field, including the simple and/or the advanced search mappings. For our example, add both a simple and an advanced search mapping for the `projectId` table field:

- Simple search mapping -- Search item:`projectIdSimpleSearch`; Results item: `projectId`
- Advanced search mapping -- Search item: `projectIdAdvSearch`; Results item: `projectId`

Step 3.4: Handle the criteria dictionary and build your WHERE clause for the `projectId` attribute:

```
public void processFormRequest (OAPageContext pageContext, OAWebBean webBean)
{
  // Find the current panel
  OAQueryBean queryBean = (OAQueryBean) webBean;
  String currentPanel = queryBean.getCurrentSearchPanel();

  // if you are on the search or the advanced search panels,
  // handle the criteria on the Go button press.
  if ((SEARCH.equals(currentPanel) || ADVANCED_SEARCH.equals(currentPanel)) &&
       pageContext.getParameter(queryBean.getGoButtonName()) != null)
  {
     handleCriteria (pageContext, webBean);
  }

  // If you are on the  saved searches panel, handle the criteria on the personalize go
  // button press.
  if ((CUSTOMIZE.equals(currentPanel) &&
       pageContext.getParameter(queryBean.getPersonalizeGoButtonName()) != null)

  {
     handleCriteria (pageContext, webBean);
  }
}

public void handleCriteria (OAPageContext pageContext, OAWebBean webBean)
{
  OAQueryBean queryBean = (OAQueryBean) webBean;

  // This gives you the current non-view attribute criteria
  Dictionary[] dic = queryBean.getNonViewAttrCriteria(pageContext);

  // If the dictionary is empty, then it means that no non-view criteria
  // is available, so return.
  if (dic == null || dic.isEmpty())
        return;

  // Otherwise process the dictionary to build your where clause.
  int size = dic.length;

  // Iterate through the dictionary to set your where clauses
  for (int i=0; i < dictSize; i++)
  {
    // Item for which the criteria is defined.
    String itemName = (String) dic[i].get(OAViewObject.CRITERIA_ITEM_NAME);

    // Condition is the SQL condition - examples: like , = etc

    String condition = (String) dic[i].get(OAViewObject.CRITERIA_CONDITION);

    // Value is the value entered with the appropriate % based on condition
    Object value = dic[i].get(OAViewObject.CRITERIA_VALUE);

    // Join condition is either AND or OR depending on what user chooses

    String joinCondition = (String) dic[i].get(OAViewObject.CRITERIA_JOIN_CONDITION);

    // You can use the following pieces of code if you need to find the actual
    // database column name of the item.

    String viewAttributeName = (String) dic[i].get(CRITERIA_VIEW_ATTRIBUTE_NAME);
    String columnName =  vo.findAttributeDef(viewAttributeName).getColumnNameForQuery();

    // Now use the above information to build your where clause.

    String whereClause = ...

    // Finally invoke a custom method on your view object to set the where clause
    // you should not execute the query if you call getNonViewAttrCriteria.
    // where clause
  }
```

```
}
```

Step 3.5: Repeat the code above that handles the criteria in the Saved Searches panel in your `processRequest()` in order to handle the criteria associated with the default personalization (if any).

```
public void processRequest (OAPageContext pageContext, OAWebBean webBean)
{

  OAQueryBean queryBean = (OAQueryBean) webBean;


  // Find the current panel
  OAQueryBean queryBean = (OAQueryBean) webBean;
  queryBean.getCurrentSearchPanel();

  // If you are on the  saved searches panel, handle the criteria
  // for the default personalization
  if ((CUSTOMIZE.equals(currentPanel) &&
               queryBean.getDefaultCustomization() != null)

  {
     handleCriteria (pageContext, webBean);
  }
}
```

Step 4: Use the `getCriteria()` method.

This is very similar to the Option 2 - `getNonViewAttrCriteria`. The key difference being that the `getCriteria()` method returns the complete criteria defined by the user. The code sample above holds good as well.

Multiple Query Regions

Case 1: Multiple query regions *not* constructed with Auto Customization Criteria mode

To display multiple query (`OAQueryBean`) regions that are not constructed with Auto Customization Criteria mode at the same time on a single page, include the following `processRequest()` code.

Important: If you are including a HGrid inside one of your query regions, this code must be added to a Controller that is located above the queryBean in the hierarchy.

Note: Create the query regions declaratively as described in the Advanced Search, Simple Search and User Personalizable Saved Searches sections above.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  OAQueryBean query1 = (OAQueryBean)
      webBean.findIndexedChildRecursive("queryRegion1");

  query1.setAdvancedSearchButtonName("AdvancedSearch1");
  query1.setClearButtonName("ClearButton1");
  query1.setGoButtonName("GoButton1");
  query1.setPersonalizationsPoplistName("PersPoplist1");
  query1.setPersonalizeButtonName("PersButton1");
  query1.setPersonalizeGoButtonName("PersGoButton1");
  query1.setSaveSearchButtonName("SaveSearchButton1");
  query1.setSaveSearchButtonText("Save Search 1");
  query1.setSimpleSearchButtonName("SimpleSearch1");
  query1.setViewPersonalizationsButtonName("ViewPersButton1");
  query1.setAddPoplistName("AddPoplist1");
  query1.setAddButtonName("AddButton1");
  query1.setAdvRadioGroupName("RadioGroup1");

  OAQueryBean query2 = (OAQueryBean)
      webBean.findIndexedChildRecursive("queryRegion2");

  query2.setAdvancedSearchButtonName("AdvancedSearch2");
  query2.setClearButtonName("ClearButton2");
  query2.setGoButtonName("GoButton2");
  query2.setPersonalizationsPoplistName("PersPoplist2");
  query2.setPersonalizeButtonName("PersButton2");
  query2.setPersonalizeGoButtonName("PersGoButton2");
  query2.setSaveSearchButtonName("SaveSearchButton2");
  query2.setSaveSearchButtonText("Save Search 2");
  query2.setSimpleSearchButtonName("SimpleSearch2");
  query2.setViewPersonalizationsButtonName("ViewPersButton2");
  query2.setAddPoplistName("AddPoplist2");
  query2.setAddButtonName("AddButton2");
```

```
  query2.setAdvRadioGroupName("RadioGroup2");

  // In each subsequent query region after the first, add this line of code
  // to ensure that OA Framework to properly name the criteria items.

  query2.setAttributeValue(OAWebBeanConstants.IS_NON_FIRST, Boolean.TRUE);
}
```

Case 2: Multiple query regions, constructed with Auto Customization Criteria mode

To display multiple query regions that are constructed with Auto Customization Criteria mode at the same time on a single page, you must perform the following steps for each of the secondary query regions. (All query regions except for the first one on a page are considered secondary query regions.)

Note: Create the query regions declaratively as described in the Advanced Search, Simple Search and User Personalizable Saved Searches sections above.

Step 1: Set the `Add Indexed Children` property of the container for the query bean to **false**.

Step 2: Call the `oracle.apps.fnd.framework.webui.OAQueryUtils createSecondaryQueryBean` API in the `processRequest` method of your controller. This call creates the secondary query web bean, adds it to the containing parent web bean, and sets all the necessary parameters such as `IS_NON_FIRST`. You do not need to manually set these parameters, as in Case 1 above. In fact, if you made any previous calls to manually set these parameters, you should remove them to avoid conflict with the `createSecondaryQueryBean` API call.

Important: If you are including a HGrid inside one of your query regions, this code must be added to a Controller that is located above the queryBean in the hierarchy.

Quick Search

Declarative Implementation

You cannot implement the Quick Search declaratively, you cannot select the appropriate beans for the Quick Search region, and you cannot set the Quick Search region on the pageLayout.

Runtime Control

Instantiate Quick Search

To create a Quick Search and associate it with a page, do the following:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{

  super.processRequest(pageContext, webBean);

  OAPageLayoutBean pageLayout = pageContext.getPageLayout();

  // Note that there's no need to call prepareForRendering() when setting the
  // Quick Search component since the OA Framework doesn't manipulate it in
  // any way.

  pageLayout.setQuickSearch(quickSearchRN);
}
```

Handle Go Button Press

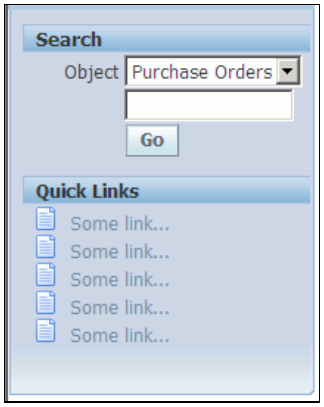
See the basic search implementation instructions in the Simple Search - Manual Search section.

Side Navigation Search

Currently, there is no `OAQueryBean` support for a search implemented in the Side Navigation.

Note: You can define the side navigation search region declaratively, however, you must associate it with the side navigation component programmatically.

Figure 15: Side Navigation Search in the OA Framework ToolBox Tutorial Application.



To implement the side navigation search region as shown in Figure 15 above:

Step 1: Create the reusable search region. Select **File > New** from the main JDeveloper menu.

- Step 1.1: In the **New** dialog, expand the Web Tier node, select **OA Components** and then select **Region** from the **Items** list. Select the **OK** button.
- Step 1.2: In the New Region dialog, enter a name that complies with the OA Framework File Standards, then select the package and set the style to **header**. Select the **OK** button.

Step 2: Select your header region in the JDeveloper Structure pane, right-click and select **New > Region**. Give this region a standards-compliant `ID` and set its `Region Style` to messageComponentLayout.

Step 3: Select your messageComponentLayout region, right-click and select **New > Item**. Give this item a standards-compliant `ID` and set its `Style` to **messageChoice**. Configure the poplist to query a list of objects, (see Standard Web Widgets for information about creating poplists), and set its `Prompt` to **Objects**.

Step 4: Select the messageComponentLayout region again, right-click and select **New > Item**. Configure this **messageTextInput** field as described in Standard Web Widgets. You should *not* specify a `Prompt` property for this field, but you should set its `Additional Text` value for accessibility compliance. Finally, set the `Length` to a reasonable value (this example is **15**) so that the side navigation does not render overly wide.

Step 5: Select your messageComponentLayout region in the JDeveloper Structure pane, right-click and select **New > Region**. Give this region a standards-compliant `ID` and set its `Region Style` to **messageLayout**.

Step 6: Select your messageLayout region, right-click and select **New > Item**. Give this item a standards-compliant `ID` and set its `Style` to **submitButton**. Set the `Attribute Set` to the standard OA Framework Button attribute set **/oracle/apps/fnd/attributesets/Buttons/Go**.

Step 7: To associate a declaratively defined search region with a side navigation, do the following in a page-level controller:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  // Instantiate a side navigation component.
  OASideNavBean sideNav =
    (OASideNavBean)createWebBean(pageContext,
                                 OAWebBeanConstants.SIDE_NAV_BEAN,
                                 null, "hpSideNav");

  // Instantiate the declaratively defined search region.  Note that the second
  // "create" parameter is the fully qualified name of the reusable search region.

  OAHeaderBean search = (OAHeaderBean)createWebBean(pageContext,
                         "/oracle/apps/fnd/framework/toolbox/tutorial/webui/HomeSearchRN"
                                                   "SearchRN", true);

  // every component needs a name
  // using JDeveloper OA Extension

  sideNav.addIndexedChild(search);

  // You don't have to do anything to get the correct CSS style for the
  // header text/line (this renders automatically based on the color of the
  // background).  You do have to set the header text size (this cannot
  // be changed by using a CSS style).

  search.setSize(2); // 2 is the smallest size

  // Get a handle to the page layout component and set its "start" to the
  // side navigation we created.
```

```
  OAPageLayoutBean pageLayout = pageContext.getPageLayoutBean;

  // Note that you must call prepareForRendering() before setting the start or it
  // won't render.

  pageLayout.prepareForRendering();
  pageLayout.setStart(sideNav);
}
```

Step 8: If the underlying query is not restricted (the query includes a WHERE clause for the results in a performant query), you *must* implement Selective Search Criteria. To do this, follow the procedure described in the Simple Search - Manual Search section.

Step 9: To handle the **Go** button press, see the basic search implementation instructions in the Simple Search - Manual Search section.

Personalization Considerations

As discussed in the Multiple Query Regions section, all secondary query regions on a page (that is, all query regions except for the first one on a page) should have the attribute `OAWebBeanConstants.IS_NON_FIRST` set to `Boolean.TRUE`, whereas the first query region on the page should not have this attribute set or have it set to `Boolean.FALSE`.

If you use OA Personalization Framework to add a predefined secondary query region to a page that does not have an existing query region, then the newly added query region becomes the first query region on that page. This, however, leads to unexpected behavior, as the existing controller code associated with the newly added query region will have the attribute `OAWebBeanConstants.IS_NON_FIRST` set to `Boolean.TRUE`.

To work around this issue, extend the query region's controller which currently sets the attribute `OAWebBeanConstants.IS_NON_FIRST` to `Boolean.TRUE`. After the `super.processRequest` call, get a handle to the queryBean and set the attribute `OAWebBeanConstants.IS_NON_FIRST` to `Boolean.FALSE` as shown in this example:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  OAQueryBean query1 = (OAQueryBean)
      webBean.findIndexedChildRecursive("queryRN1");

  query1.setAttributeValue(OAWebBeanConstants.IS_NON_FIRST, Boolean.FALSE);
}
```

Use OA Personalization Framework to update the query region's Controller Class property with your new controller.

Searching Using Descriptive Flexfields

Setup

Note: This optional feature requires specific uptake from product development teams that wish to use it.

Declarative changes are required to enable search with DFF criteria. See 'Declarative Implementation' section below. In summary, here are the highlights of declarative changes required in order to enable search with DFF criteria.

1. To add a DFF item as a search criterion, developers should follow the same steps as for defining standard new search criteria under a search region.
2. The DFF search item must be configured as a search criterion. The steps will depend on the search mode:
    - **RBS Mode** — On the corresponding DFF item that already exists in the search result table, ensure the 'Search Allowed' property is set to 'True'.
    - **ACC Mode** — On the DFF item added to the CriteriaRow in the Simple/Advanced region, ensure that the 'Flexfields' properties are set according to the corresponding DFF item in the search results table.

Declarative Implementation

Query Bean with RBS (Results Based Search) Construction Mode

Step 1: Follow the same steps outlined in Section 'Declarative Implementation: Results Base Search' (also in the OA Framework Developer's Guide) to implement the Simple Search Panel with RBS Construction Mode.

Step 2: To enable search on a DFF item, set the 'Search Allowed' property to 'True' and this item will be included in search panels.

Query Bean with ACC (Auto Customization Criteria) Construction Mode

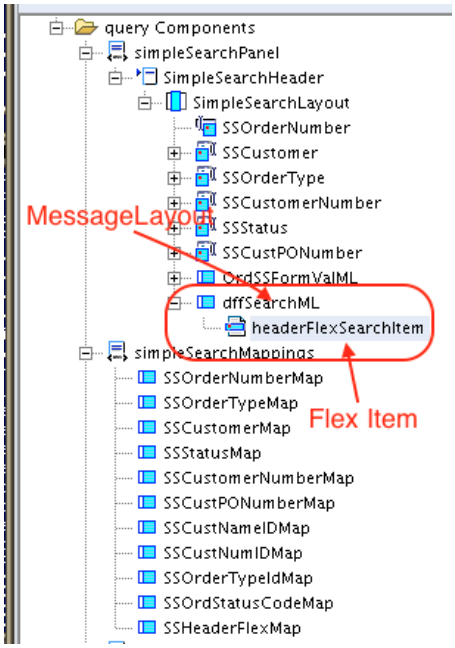
Follow the steps outlined in the section Declarative Implementation: Results Based Search.

a. Simple Search Panel: Declarative Implementation

Follow the same procedure for adding any criteria under Simple Search Panel, and provide criteria mapping.

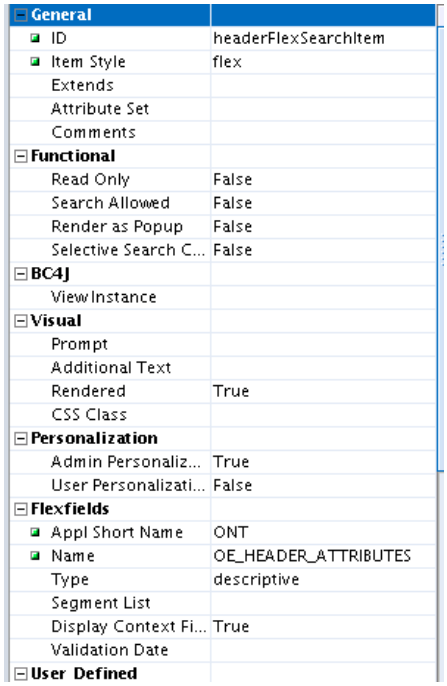
Step 1: Add a MessageLayout under the SimpleSearchHeader region

Note: A flex item cannot be added under MessageComponentLayout directly, so you must first add a MessageLayout and then add the Flex item under it.

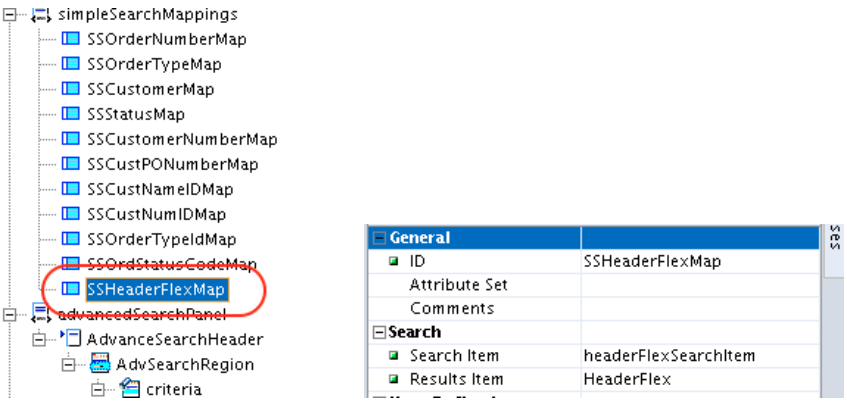


Step 2: Add the Flex item under the MessageLayout, and then configure the Flex item to be in sync with the Flex Item under the search result table.

Note: Leave the ViewInstance property empty for the Flex Item.

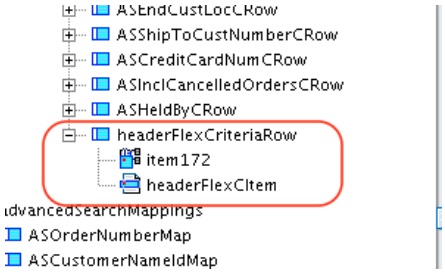


Step 3: Add a Simple Search mapping between the above 'headerFlexSearchItem' and the HeaderFlex in the table:

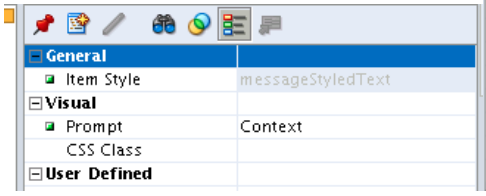


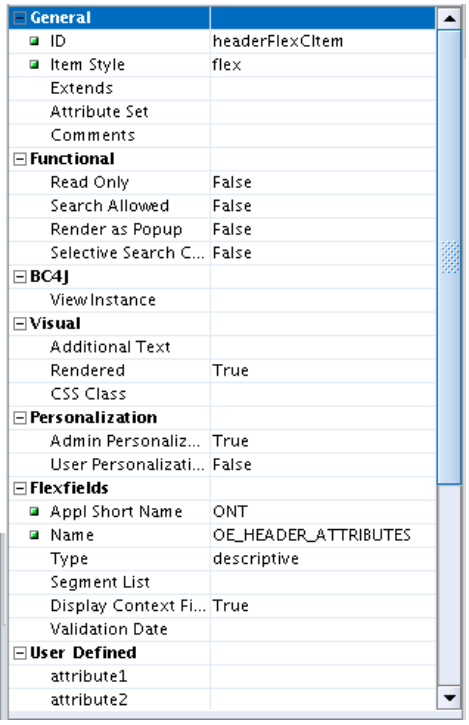
b. Advanced Search Panel: Declarative Implementation

Step 1: Add a new Criteria Row under AdvancedSearchPanel > AdvancedSearchHeader > AdvSearchRegion > Criteria

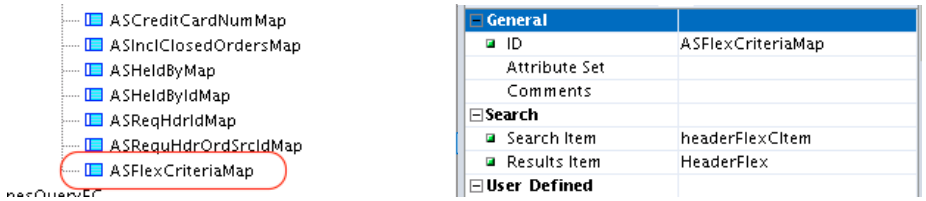


Step 2: Configure the first item under the 'headerFlexCriteriarRow' with the Prompt 'Context', or any value you wish the Flex Criteria to look like in the Add new criteria poplist. We have used 'Context' here, to match the Flex column name in the Search Result table for HeaderFlex. Configure the Flex item according to the HeaderFlex, but leave ViewInstance property empty.





Step 3: Add a criteria row mapping under advancedSearchMappings for the headerFlexCriteriaRow.



List Search

No extra configuration is required if the DFF criteria have already been added as part of the Classic Search implementation.

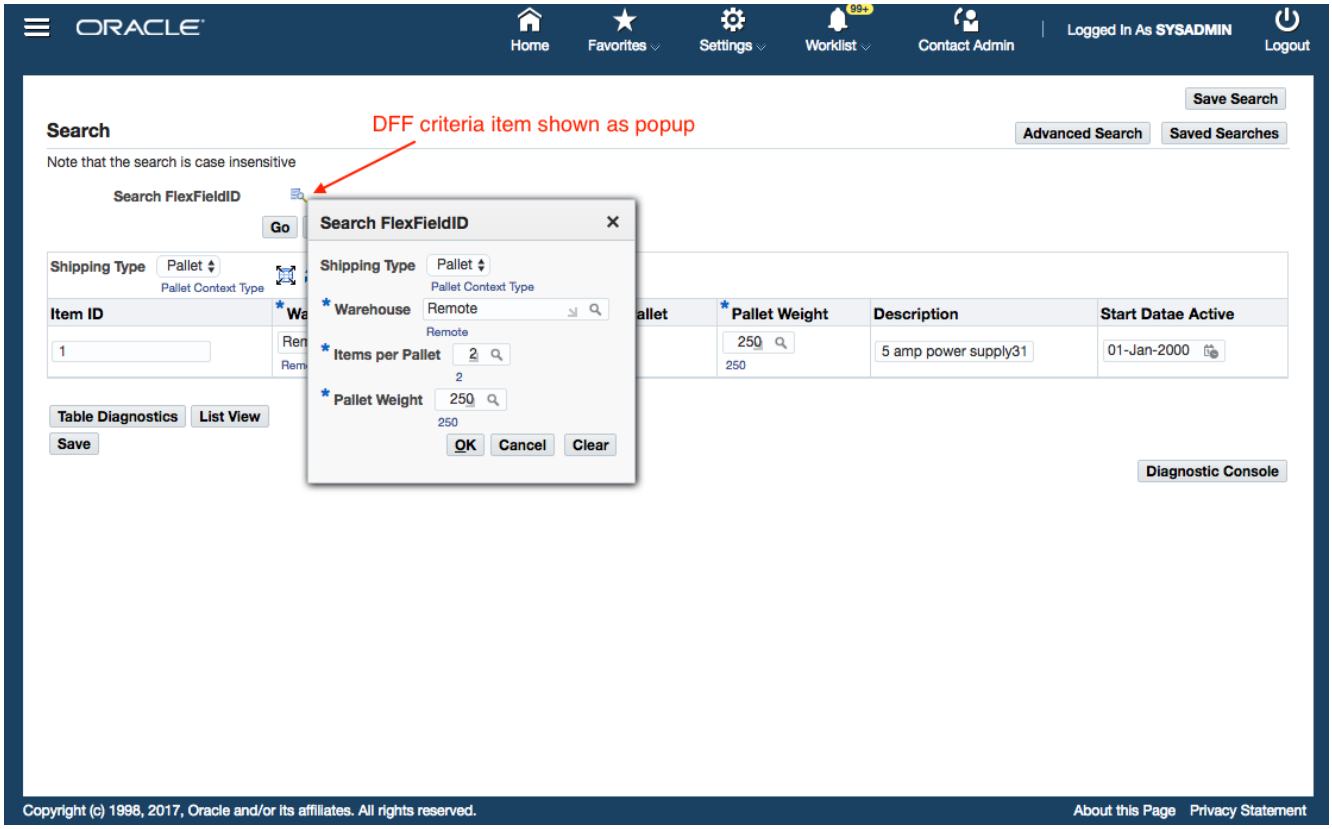
Search Behavior: Classic Search

Simple Search Panel

Under the Simple Search panel, the DFF criteria item will be rendered as a DFF popup, as shown in the screenshot below.

Note the following important points:

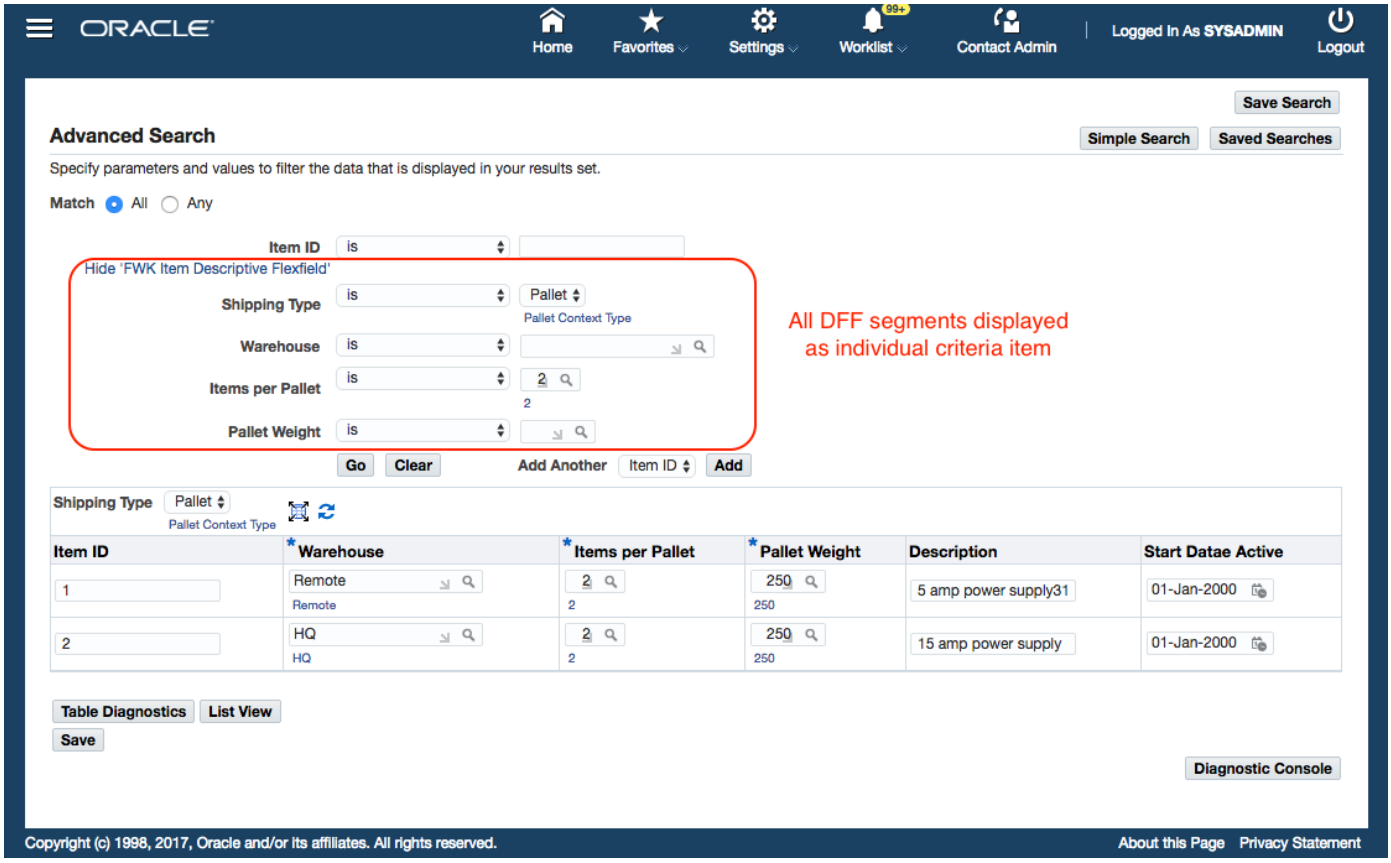
- There is a 'Clear' button in the popup that clears the DFF segment's value. In addition, the 'Clear' button in the Simple search panel also clears any existing DFF segment value in the DFF popup.
- DFF validation will still be performed in the DFF popup. As a result, to continue with the search the user needs to enter a valid segment value in addition to all required segment values.



Advanced Search Panel

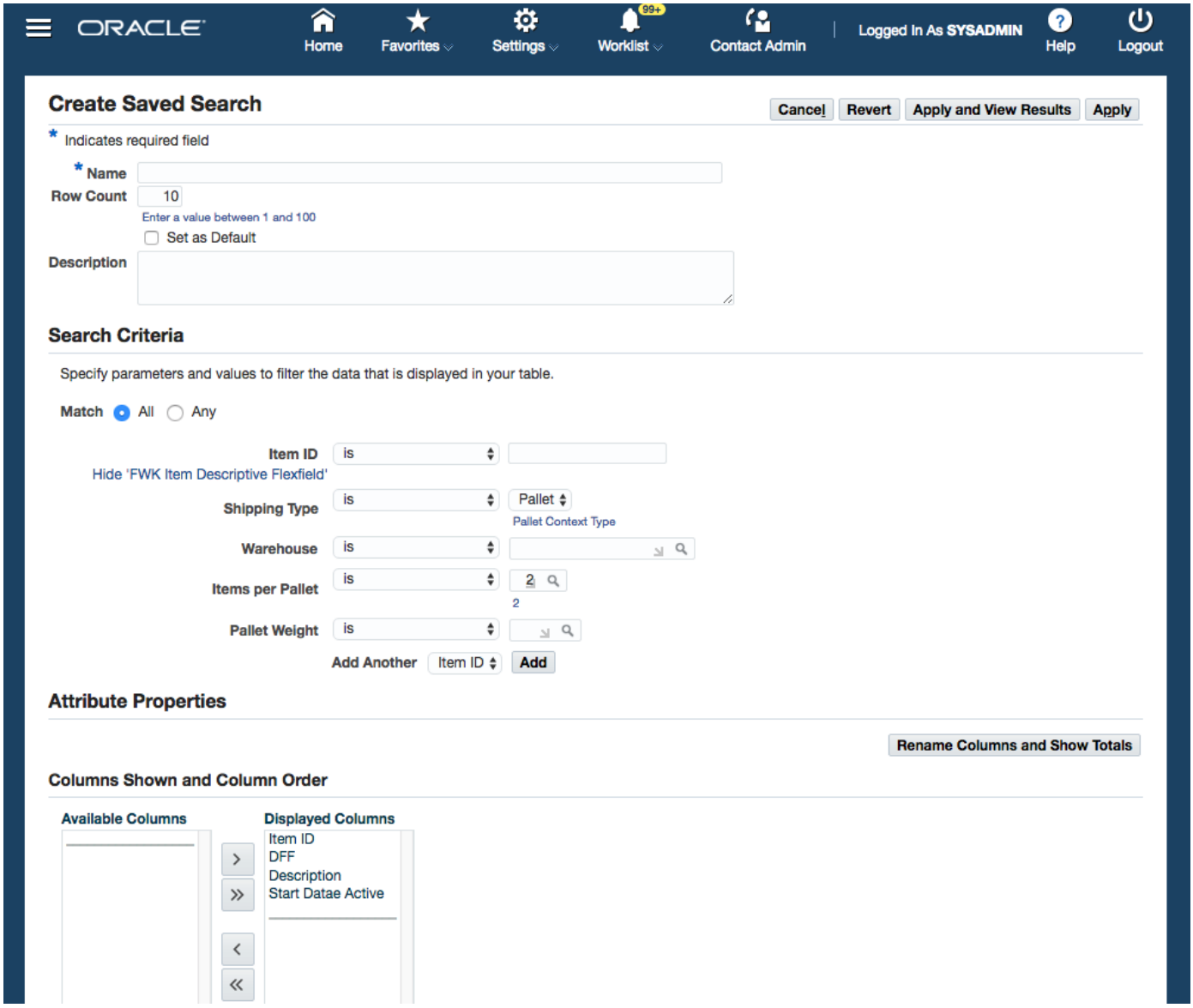
Under the Advanced Search Panel, each DFF segment (global, context and contextual segments) will be displayed as an individual criterion, with the condition choice being rendered against each segment.

Note: DFF validation is relaxed under the Advanced Search Panel, and user would not be required to enter a value for a required segment to perform search. However, validation for each segment will still be in place, so as a result the user needs to enter a valid segment value as per the definition of the segment's valueset.



Save New Search

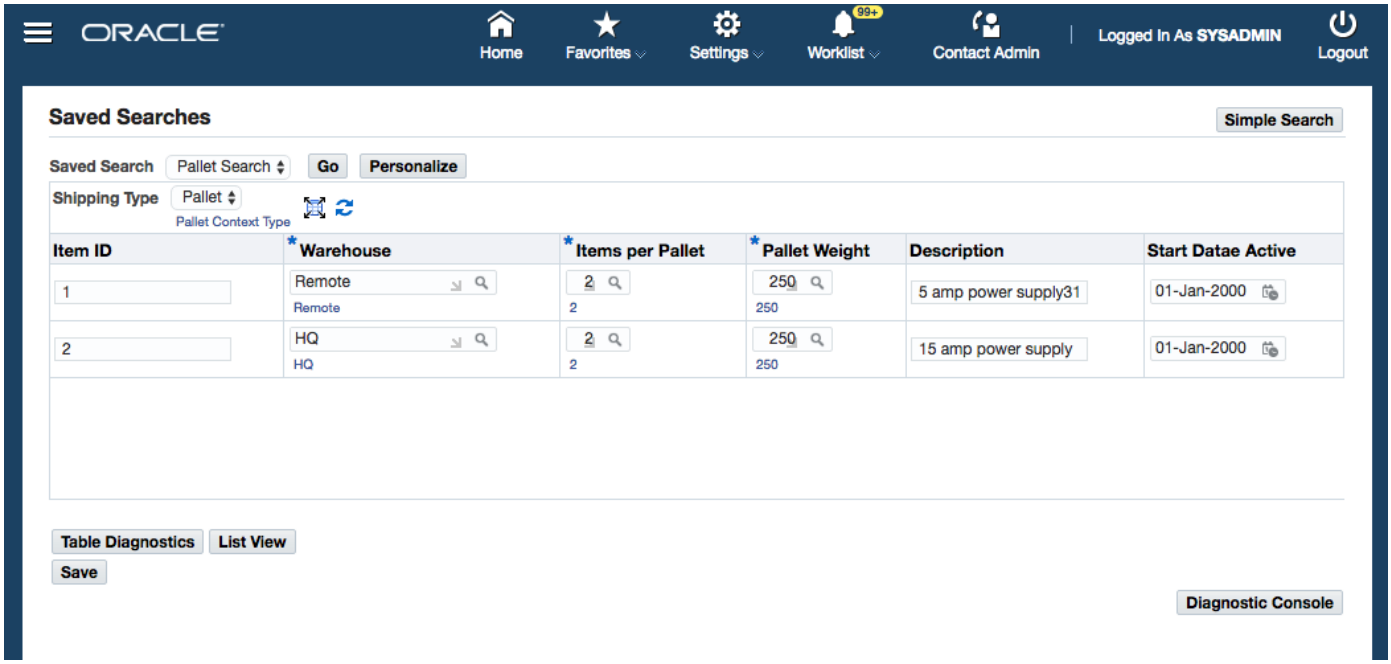
The behavior of a DFF search criterion in the Create Saved Search / Update Saved Search page is similar to that in Advanced Search Panel.



Saved Search Panel

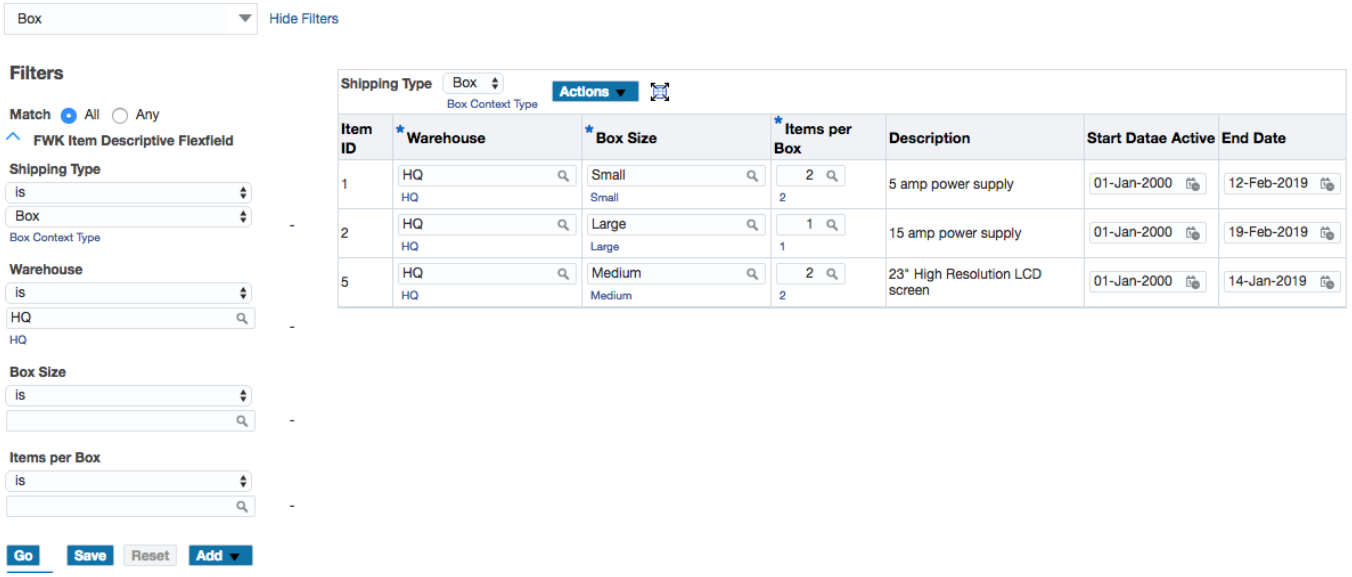
A query is executed based on the saved DFF search criteria in the Saved Search Panel.

Note: For query execution performed by OA Framework, the handling of DFF search criteria will also be done by OA Framework. However, if product developers have implemented criteria handling, then that logic needs to be enhanced to handle the DFF search criteria as well.



Search Behavior: List Search

The following screenshot illustrates the behavior here.



Known Issues

- See a summary of key Search issues with suggested workarounds if available.

Related Information

- BLAF UI Guidelines
  - Search and Query Templates
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OAQueryBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAHeaderBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OAAdvancedSearchBean`
  - `oracle.apps.fnd.framework.webui.OAQueryUtils`
- OA Framework ToolBox Tutorial / Sample Library
  - Search Lab
  - `oracle/apps/fnd/framework/toolbox/samplelib/webui/SearchRBSPG`
  - `oracle/apps/fnd/framework/toolbox/samplelib/webui/SearchACCPG`
  - `oracle/apps/fnd/framework/toolbox/tutorial/webui/PoSearchPG`
  - `oracle/apps/fnd/framework/toolbox/tutorial/server/PoSimmpleSummaryVOImpl`

- oracle/apps/fnd/framework/toolbox/tutorial/server/SearchAMImpl

Separator Line

The separator line visual component used to divide contextual or static content from primary content on an OA Framework-based page is obsolete as of Release 12.

Shuttle

Overview

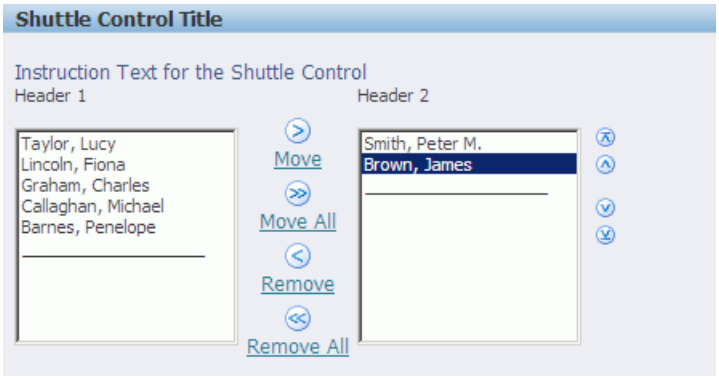
As described in the BLAF UI Guideline: Shuttle and/or Reorder Controls, the Shuttle and/or Reorder feature lets the user move items between two lists, and optionally reorder the items in the list(s).

The main function of the Shuttle is to move items from one list to the other. When you implement a Shuttle region, you define two lists:

- A Leading list - typically titled "Available {ObjectType | Items}"
- A Trailing list - typically titled "Selected {ObjectType | Items}"

You can implement the Shuttle region to optionally allow the user to reorder the contents of the Trailing list by using icons on the side of the Trailing list to move the selected items up to the top of the list, up one slot in the list, down to the bottom of the list, or down one slot in the list.

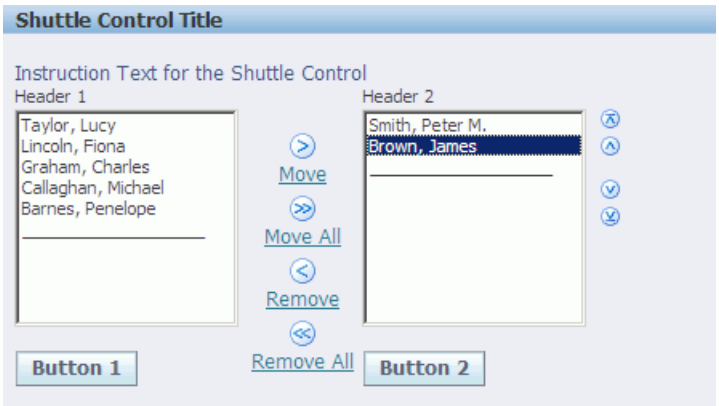
Figure 1: Shuttle region with Reorder controls for the Trailing list.



If you wish to implement a Shuttle region with just one list, to achieve only Reorder functionality, then you need only define a Leading list.

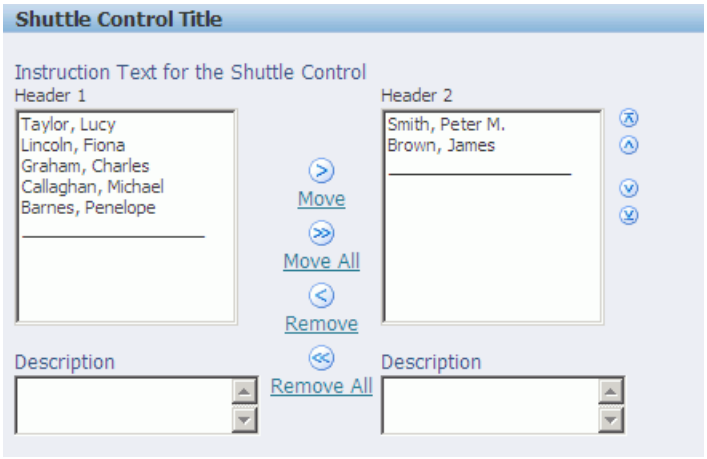
You can also optionally define buttons or icons in the footer below each of the Shuttle lists as shown in the Figure 2 below. These buttons or icons take up only one row below the lists with no wrapping. Simply define a Leading footer named child or Trailing footer named child in your Shuttle region. This automatically creates a flowLayout region by default, to which you can add a button or image.

Figure 2: Shuttle region with buttons in the footer of each list.



Another variation you can implement is to display a description of a selected item beneath a Shuttle list, as shown in the Figure 3 below.

Figure 3: Shuttle region that displays a description of selected content below each list.



Contents

- Declarative Implementation
- Runtime Control
  - Adding a Leading List Data Filter
  - Getting the Leading or Trailing List
  - Caching
- Personalization Considerations
- Touch Device Considerations
- Known Issues
- Related Information

Declarative Implementation

Following is a brief outline of the steps to create a Shuttle region.

Step 1: Create a page with a pageLayout region using OA Extension. Make sure the `Form` property on the pageLayout region is set to **True**.

Step 2: Select the pageLayout region in the Structure pane, and choose **New** > **Region** from the context menu. Set the following properties on this new region (Required properties are marked with *):

- *`ID` - set the shuttle's ID property in accordance with the OA Framework File / Package/ Directory Standards.
- *`Region Style` - set the region style to **shuttle**.
- `Add Indexed Children` - make sure this property is set to **True**, so that OA Framework automatically generates the web beans under this web bean hierarchy. The default is True.
- `Available Header` - specify the header text of the first (leading) list.
- `Selected Header` - specify the header text of the second (trailing) list. If you want to implement just one list for Reordering, that is, you do not want to shuttle items between two lists, then leave this property blank.
- `Ordering Allowed` - set to **True** if you want to enable ordering of the contents of the Selected (second or trailing) list. If you implement only one (leading) list in the shuttle region to create a Reordering region, then setting this property to True enables ordering on that leading list. The default is False.

Step 3: The Shuttle region can have a maximum of two list web beans, referred to as the leading and trailing lists. The trailing list is omitted when you want to implement a shuttle region with a single list for the purpose of reordering the contents of that list. When you define a shuttle region, OA Extension automatically creates a **leading** component for you, which contains a single **list** item. Be sure to set the following properties on the **list** item (Required properties are marked with *):

- *`ID` - set the list's ID property in accordance with the OA Framework File / Package/ Directory Standards.
- *`Multi-Select Allowed` - set to **True** to allow the multiple selection of items in the list. The default is False.
- `Picklist View Definition` - specify the fully qualified view object name that is the data source to the List web bean. (For example, `oracle.apps.fnd.framework.server.FndApplicationVO`.)
- *`Picklist View Instance` - alternately specify a view instance name for the data source of the list, if the Picklist View Definition is not available. Note that setting this property overrides the Picklist View Definition property. (For example, FndApplicationVO, which needs to be present in the Application Module.)
- *`Picklist Display Attribute` - specify the view attribute name that serves as the displayed values of the list's content.
- *`Picklist Value Attribute` - specify the view attribute name that serves as the internal values of the list's content. This property, and not the `Picklist Display Attribute` property, uniquely identifies the elements in the list.
- `Picklist Description Attribute` - specify the view attribute name that serves as the descriptions of the list's content.
- `Rendered` - specify **True** to render this list.

- *`List Height` - specify the suggested display height of the list, in bytes. The default is null. If this property is not set, the height is determined based on the lengths of both lists and their minimum and maximum values. The value should be in the range of 10 to 20.

  Note: The `List Height` is only a suggested value. Your browser application and UIX determines the final height of the list based on the number of items in the list and the size of the browser window.

Step 4: To create an optional trailing list, select the **shuttle** region in the Structure pane and choose **New** > **trailing** from the context menu. OA Extension automatically creates a **trailing** component for you, which contains a single **list** item. Refer to Step 3 for the list of properties that you should also set on the trailing list item. Make sure you specify names for the `Picklist Display Attribute, Picklist Value Attribute` and `Picklist Description Attribute` properties that match the corresponding property of the leading list item.

Note: If you do not want to pre-populate the trailing list when the page initially renders, you can leave the `Picklist View Definition, Picklist View Instance, Picklist Display Attribute, Picklist Value Attribute` and `Picklist Description Attribute` blank. OA Framework takes care of retaining user selected values in the trailing list when the page refreshes.

Note: The `Picklist Value Attribute` uniquely identifies the elements in a list, so if you want to pre-populate both the leading list and trailing list with data, be sure to set the `Picklist Value Attribute` property to a different value for each list. Setting this property to the same value for both lists will result in undesirable behaviour, such as causing the values in the leading list to disappear.

Step 5: You can include buttons or icons in the footer of each list, as shown in Figure 2. Select the **shuttle** region in the Structure pane, and choose **New** > **leadingFooter** (for a footer in the leading list) or **New** > **trailingFooter** (for a footer in the trailing list). OA Extension automatically creates a leadingFooter or trailingFooter component, respectively, each containing a flowLayout region. You can then add your buttons or icons to this flowlayout region.

Note: If you set the Rendered property to **False** for either the leading or trailing list, the leading or trailing footer also does not render, respectively. Since the footer is directly linked to its respective list, if the list is not rendered, the footer is irrelevant.

Runtime Control

There are no programmatic steps necessary to implement a basic Shuttle region as described above. There is also no need to execute the query on the view objects as OA Framework takes care of querying the view objects for you.

Adding a Leading List Data Filter

If you wish to include a filter above the Leading list, so the user can narrow down the choices in the leading list, you need to do so programmatically. The following code example illustrates how to add a poplist to the filter region of the Leading list. An `onChange` action is triggered to handle changes in the filter:

```
OADefaultShuttleBean shuttle = ...
OAFlowLayoutBean filterChild =
  (OAFlowLayoutBean)createWebBean(pageContext,FLOW_LAYOUT_BEAN);

// Create a Javascript function for the onChange handler
// for the filter
pageContext.putJavaScriptFunction("doFilterSwitch", "function doFilterSwitch()
  {submitForm('DefaultFormName',0,{FilterSwitch:'Y'}); return true; }");
OAMessageChoiceBean filter =
  (OAMessageChoiceBean)createWebBean(pageContext,MESSAGE_CHOICE_BEAN);
filter.setUINodeName("myfilter");
filter.setOnChange("doRespSwitch()");
filter.setPickListCacheEnabled(false);
filterChild.addIndexedChild(filter);
shuttle.setFilter(filterChild);
```

Then in `processFormRequest`, you can detect filter switches using:

```
if ("Y".equals(pageContext.getParameter("FilterSwitch")))
  {
     ... process the filter switch ...
  }
```

Getting the Leading or Trailing List

If you need to get a handle to the leading or trailing lists, you can call the `findChildRecursive` method on the Shuttle web bean. For example:

```
OADefaultShuttleBean shuttle = ...
OADefaultListBean leadingList =
  (OADefaultListBean)shuttleBean.findChildRecursive("myLeadingList");
```

On form submission, you can also obtain the option values present in the leading and trailing lists using the following methods:

```
public String[] getLeadingListOptionValues(OAPageContext pageContext, OAWebBean
  webBean)
public String[] getTrailingListOptionValues(OAPageContext pageContext, OAWebBean
  webBean
```

For example, the following code sample returns an array of values in the trailing list, ordered according to the order in which they appear in the list:

```
String[] trailingItems =
  shuttle.getTrailingListOptionValues(pageContext, shuttle);
```

Caching

Like a picklist, the lists in a Shuttle web bean also cache their initial values forever (that is, as static values in the JVM). If the initial data in a Shuttle's Leading or Trailing list can change during the lifetime of the JVM, you should disable caching on the Shuttle using the `oracle.apps.fnd.framework.webui.beans.form.OADefaultListBean` method `setListCacheEnabled(boolean)`.

For example, suppose you want to display a different set of data in the Shuttle's Leading list based on the value that is selected in a poplist associated with the Leading list. You would need to disable the caching for the list in the Shuttle, as illustrated in the following code sample:

```
OADefaultListBean list = (OADefaultListBean)webBean.findChildRecursive("your list here");
list.setListCacheEnabled(false);
```

Accessibility Considerations

Keyboard navigation is as follows: press [Tab] or [Shift + Tab] to navigate through leading lists, trailing lists, and buttons.

Selection: with the focus on either the leading list box or trailing list box, pressing [Up/Down Arrow] will shift the focus to the next item and select it. If Multi-Select Allowed is enabled, you can select multiple items by holding [Shift] while pressing [Up/Down Arrow].

Invocation: With the focus on any of the buttons (move, remove or reorder), you can take the corresponding action on selected items by pressing [Space].

Personalization Considerations

See a summary of Shuttle personalization considerations in the Oracle Application Framework Personalization Guide.

Touch Device Considerations

See a summary of considerations for the Shuttle feature on touch devices in the Mobile Applications topic.

Known Issues

- None

Related Information

- BLAF UI Guideline
  - Shuttle and/or Reorder Controls
- Javadoc Files
  - `oracle.apps.fnd.framework.webui.beans.form.OADefaultShuttleBean`
  - `oracle.apps.fnd.framework.webui.beans.form.OADefaultListBean`
- Lesson(s)
- Sample Code

Standard Web Widgets

Overview

As described in the Oracle Browser Look-and-Feel UI Guideline: Standard Web Widgets, this document provides instructions for adding the following core components to your page:

- Text Field
- Text Area
- Choice ('Pulldown' or 'Poplist')
- Dynamic Poplists
- List Box
- Check Box and Switch
- Radio Group / Buttons
- Spin Box

Related Information

The content below focuses on the basic instructions for creating and configuring these items. See the following documents for additional information related to special behaviors:

- To change component properties at runtime (for example, you want to make a text entry field read only), see the Dynamic User Interface documentation for recommended approaches.
- To enable partial page rendering (PPR) events when users interact with these components, also see the Dynamic User Interface documentation.
- To configure some of these items to perform form submits when selected, see Submitting the Form.
- To disable "save for changes" warnings for certain components, see the Save Model documentation.
- To configure field-level hints or tips, see Inline Messaging, Tips, Hints and Bubble Text.
- For a description of all the possible properties that you can set for each item type, see the OA Component Reference in the JDeveloper online Help. To access, select **Help > Help Topics** from the main menu. In the **Help Navigator** window, expand the **OA Component Reference** and **OA Item Styles** nodes.
- `setEnd` on any item should be used to set only an item, not a region, as its named child.

Text Field

Declarative Implementation

Data Entry Text Field

Step 1: Create an item whose `Item Style` is **messageTextInput** (see the List of Values documentation if you want to create an LOV field; see the Date Picker documentation if you want to create a text field for dates). Assign an `ID` in accordance with the OA Framework Package / File / Directory standards.

Step 2: Apply an attribute set as required by the OA Framework View Coding Standards. See Implementing the View for additional information about attribute sets.

Step 3: Specify values for the following properties:

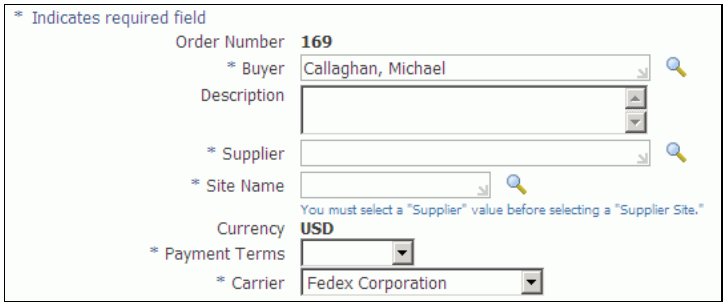
- `Prompt` - if not already specified by the attribute set
- `Required` - set this property to one of the following values:
  - **yes** to indicate a value must be entered for the field. Always displays the "Indicates required field" key for the field, as long as the field is not within a table.
  - **no** to indicate that a value does not have to be entered for this field. Does not display the "Indicates required field" key for the field. Note that if even this property is set to **no**, the "Indicates required field" key for the field may still display if the `showRequired` property is set to **True** and this messageTextInput item is not within a table.
- `showRequired` - set this property to **True** to always display the "Indicates required field" key for the field, as long as the field is not within a table. Note that even if this property is set to **False**, the "Indicates required field" key for the field may still display if the `Required` property is set to **yes** and this messageTextInput item is not within a table.

  Note: Refer to the Runtime Control section below for instructions on how to set the "Indicates required field" key for the field on a row-by-row basis within a table.

- CSS Class - set to **OraFieldText**.
- `Length` - set to the display length of the data entry text field, in bytes
- `Maximum Length` - set to maximum number of bytes per line that can be entered.

Note: If you have even one required item on your page, you must add the "Indicates required field" key as shown in Figure 1:

Figure 1: Example of required and optional fields with the "indicates required field" key



To do this, create a region beneath your main content header and set the `Extends` property to `oracle/apps/fnd/framework/webui/OAReqFieldDescRG` and set its `Width` to **100%**.

Step 4: Ensure the `Data Type` value is correct, and that the `Length` is appropriate for the amount of data that you want to display (if you want to set a limit on the number of bytes the user can enter, make sure the `Maximum Length` property is set correctly). Remember that `Maximum Length` should equal the corresponding database column length, while the `Length` property should be optimized for the UI.

Step 5: If your text field reads and/or writes to an underlying data source, set the `View Instance` and `View Attribute` names for the underlying data source.

Step 6 (optional): Set the `Secret` property to **True** if the field's data should be illegible (a password field is a common example of this).

Step 7 (optional): Set the `Initial Value` property to specify a default value (see Defaulting in Implementing the View for an overview of valid defaulting approaches and how they are resolved at runtime).

Step 8. (optional) As of Release 12.2.6, set the `Placeholder Text` property if you wish to show a short hint describing the expected value for the text input field.

Runtime Control

At runtime, OA Framework instantiates an `oracle.apps.fnd.framework.webui.beans.message.OAMessageTextInputBean`.

As of Release 12, if you have a messageTextInput item in a table column, you can control whether the "Indicates required field" key displays on a row-by-row basis by using the new `oracle.apps.fnd.framework.webui.OAWebBeanConstants` variable `SHOW_REQUIRED_ATTR`. If `SHOW_REQUIRED_ATTR` is set to **True**, the "Indicates required field" key always displays. You can programmatically set this attribute on a row-by-row basis by using a bound value. For example:

```
OAMessageTextInputBean mBean = (OAMessageTextInputBean)
  webBean.findIndexedChildRecursive( );
mBean.setAttributeValue(SHOW_REQUIRED_ATTR,
  new OADataBoundValueViewObject(mBean, viewAttr name, viewUsage));
```

Declarative Implementation

Display-Only Text Field

If there's any reason for the field to be updateable in different use case scenarios, you could simply create a **messageTextInput** field and set its `Read Only` property to **True**.

If the field is always display-only, it's better to create a **messageStyledText** item instead. In this case:

Step 1: Apply an attribute set as required by the OA Framework View Coding Standards. See Implementing the View for additional information about attribute sets.

Step 2: Specify the `Prompt` (if not specified by the attribute set) and set the `CSS Class` to **OraDataText** to ensure that it renders in bold.

Step 3: Set the `View Instance` and `View Attribute` names for the underlying data source.

Step 4: (optional) Set the `Destination URI` if you want this data value to render as a link. In this case, set the `CSS Class` property to **OraLinkText**. See Buttons (Links) for more information about working with links.

At runtime, OA Framework instantiates an `oracle.apps.fnd.framework.webui.beans.message.OAMessageStyledTextBean`.

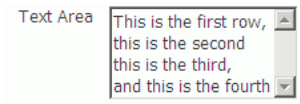
Text Area

Declarative Implementation

A text area is simply a **messageTextInput** item whose `Height` property is set to the number of text rows that you want to display. For example, the following item's `Height` property is set to **4**.

Note that you should also set the `Vertical Alignment` property to **top** to ensure that the prompt renders as shown.

Figure 2: Example of a text area with a height of 4 and a vertical alignment of top.



Set the `Length` property to the display length of the text area, in bytes, and set the `Maximum Length` property to the maximum number of bytes that can be

displayed.

At runtime, OA Framework instantiates an `oracle.apps.fnd.framework.webui.beans.message.OAMessageTextInputBean`.

If you want the read-only text area to wrap, you must add the following line of code to the `processRequest` method of your controller:

```
textBean.setWrap(SOFT_WRAP);
```

Note: `SOFT_WRAP` does not work for continuous text (without spaces and some other special characters) as per HTML5 specifications.

Choice ('Pulldown' or 'Poplist')

A poplist is a small list of items from which a single value may be selected. The list of values is obtained from an associated 2-attribute view object including a display value, and an internal developer key.

For example, we created "positions" for the ToolBox Tutorial as shown in Figure 3.

Figure 3: Example of a required poplist with a "blank" option.

c

Tip: The browser automatically sizes the poplist to accommodate the largest item. You cannot override the display width.

This poplist is based a view object that queries lookup codes. The associated SQL statement shown below returns the values in the following table.

```
SELECT lookup_code,
       meaning
FROM   fwk_tbx_lookup_codes_vl
WHERE  lookup_type = 'FWK_TBX_POSITIONS'
```

Developer Key	Display Value
BUYER	Buyer
PRESIDENT	President
VICE_PRESIDENT	Vice President
SALES_REPRESENTATIVE	Sales Representative
DIRECTOR	Director
GROUP_MANAGER	Group Manager

Declarative Implementation

Step 1: Create a view object to be used by your poplist.

- Per the OA Framework Package / File / Directory standards, you should create this object in a `*.poplist.server` package. For example, we created this ToolBox Tutorial `PositionsVO` poplist in the `oracle.apps.fnd.framework.toolbox.poplist.server` package.
- As described above, the view object should have two attributes: one for the poplist display value, and one for the internal developer key.

Step 2: Create an item whose `Item Style` is **messageChoice** . Assign an `ID` in accordance with the OA Framework Package / File / Directory standards.

Step 3: Apply an attribute set as required by the OA Framework View Coding Standards. See Implementing the View for additional information about attribute sets.

Step 4: Specify the `Prompt` (if not specified by the attribute set).

Step 5: Specify the view object that you created in Step 1.

- If you want a poplist's data to be cached on the JVM, set the `Picklist View Definition` property to the fully qualified name of the view object definition. For example, **oracle.apps.fnd.framework.toolbox.poplist.server.PositionsVO**. Cached poplists are shared by all users of the JVM, and are identified by a unique key comprised of the entire poplist view object SQL query string, WHERE clause parameter values, language, and orgID.
- If you want your poplist to be used exclusively by the current session, set the `Picklist View Instance` property instead (for example, **PoplistVO1**). In this case, remember to add your poplist view object to the root UI application module for the page (or application module for the shared regions) that will include this poplist.

See the Cache Invalidation topic below for additional instructions if you want to enable automatic updates to the poplist view object values when underlying database values change. Note that you can leverage cache invalidation for both kinds of poplists (cached on the JVM and private to the current session).

Step 6: Map the poplist to its view object display and developer key attributes.

- Set the `Picklist Display Attribute` to the view object attribute whose value you want to display in the poplist.
- Set the `Picklist Value Attribute` to the view object attribute whose value you want to use as the internal developer key.

To use our lookup code example above, we would set the `Picklist Display Attribute` to **Meaning**, and the `Picklist Value Attribute` to **LookupCode**.

Step 7: (optional) If the poplist should read/write its selected value from an underlying data source, set the `View Instance` and `View Attribute` values accordingly as you would for any other data entry component. The View Instance value should not be same as the Picklist View Instance value.

Note: The `View Attribute` value must correspond to the `Picklist Value Attribute` value. For example, assume you want to use the Positions poplist shown above in Figure 3 to assign a position while creating an employee. The `Picklist Value Attribute` should map to the **LookupCode** value. Similarly, the **POSITION_CODE** column (that you intend to update with the poplist selection) in the **FWK_TBX_EMPLOYEES** table stores a lookup code value.

Step 8: Set the `Initial Value` if you want the poplist to render with a default selected value. Remember that this effectively performs a selection of the

`Picklist Value Attribute` value, and must therefore be among a valid value for the underlying view object attribute. For example, **VICE_PRESIDENT** (the developer keys value that maps to the **LookupCode** view object attribute) is a valid value for the PositionsVO described here. **Vice President** (the display value that maps to the **Meaning** view object attribute) is not.

Step 9: Set the `Required` property to **yes** if you want the required indicator to render for the poplist. If you don't want a blank option to render in this case, you should specify an `Initial Value`.

Step 10: (optional): If you want to create a poplist *without* a required indicator and *without* a blank option, set the `Required` property to **no** and set the `Add Blank Value` property to **False** (by default, optional poplists include a blank value as shown in Figure 3 above). If you choose to suppress the blank value for an optional poplist, be sure to specify an `Initial Value`.

Note that you can also set this value programmatically by calling the `setAllowBlankValue(false)` method on the `oracle.apps.fnd.framework.webui.beans.message.OAMessageChoiceBean`.

Tip: If you set the `Add Blank Value` property to **True** on a poplist that you have designated as being required, and you specify a default value, the OA Framework ignores this setting. The combined `Required` and `Initial Value` settings take precedence.

Cache Invalidation

Previously, poplist data was cached on the middle tier in a proprietary OA Framework cache that could not be refreshed without bouncing the Java Virtual Machine (JVM). As a consequence, poplists could not reflect changes in the underlying data source while users were actively working. Now, poplist data is cached using the Oracle Applications Java Caching Framework, and as a result, you can implement cache invalidation to automatically refresh poplists when the data they query changes. For example, if a poplist queries the values A, B, C, and D, and a new value E is inserted in the underlying table, your poplist can automatically reflect the addition of this fifth value the next time it renders its data.

Tip: Most poplists would benefit from this functionality.

To implement poplist cache invalidation, the owner of the underlying database table must first perform these prerequisite steps (for example, the Applications Technology Group owns **FND_LOOKUPS**, and will be providing the following for this shared data):

Note: For additional information on database invalidation, refer to section 3.4.5 Configuration for DB Invalidation Feature of the Oracle Applications Java Caching Framework.

Prerequisite: Apply patch 3688496 to any environment in which you plan to deploy this caching functionality.

Step 1: Create business event(s) using Oracle Workflow (please consult the Java Business Event System Developer's Guide for additional information). Typically, you should create distinct update, delete and insert events, although it is acceptable to create a subset of these events as appropriate for the table at hand.

Step 2: As described in the Java Business Event System Developer's Guide, create the required `wf_event.raise(...)` API calls in PL/SQL.

- The `p_event_name` parameter must match one of the event names from Step 1.
- The `p_event_key` parameter must uniquely identify the poplist data being updated. For example, for FND_LOOKUPS, `p_event_key` would be a lookup type.

The owner of the poplist must then implement the following two methods for the poplist's associated view object:

```
/**
* Returns the database event names that are raised when the database
* data represented by this View Object is updated.
* @return the database event name.
*/
public String[] getCacheEventNames();
```

Note: `getCacheEventNames()` must return the list of event names registered in Step 1 above. For example, if your view object is based on **FND_LOOKUPS**, the `getCacheEventNames()` method should be implemented to return the following:

```
{ "oracle.apps.fnd.lookup.type.update",
  "oracle.apps.fnd.lookup.type.delete",
  "oracle.apps.fnd.lookup.code.insert",
  "oracle.apps.fnd.lookup.code.update",
  "oracle.apps.fnd.lookup.code.delete" }

/**
* Returns the key that uniquely identifies the data in this view
* object instance.
* @Return the key that identifies the data.
*/
Public String getCacheKey();
```

Note: `getCacheKey()` must return the same value as the `p_event_key` parameter used for raising the data change events. For example, if your view object is based on **FND_LOOKUPS**, the `getCacheKey()` method should return the lookup type being selected by the view object.

Runtime Control

To get the selected value in the poplist, add logic like the following:

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
   ...
   String selectedValue = pageContext.getParameter("<PoplistItemName>");
   ...
}
```

Disabling Poplist Caching

If you need to explicitly disable poplist caching, use the following API:

```
poplistBean.setPickListCacheEnabled(false)
```

You should *explicitly* disable poplist caching in the following cases:

1. **When you use the view object row getter method to encrypt the value for a poplist.**

   The poplist data object that OA Framework creates the first time is cached in the JVM and reused until you explicitly disable caching. When you encrypt the value of a poplist in the view object row getter method, the encrypt method uses the current Oracle E-Business Suite user session ID as a key to encrypt the poplist value. When you decrypt the values in their controller to further process the poplist, the decrypt method will also use the current Oracle E-Business Suite user session ID as the key to decrypt the value.

   Since the poplist data object is cached at a JVM level, the data object will always contain the cached value with the encryption done with the first Oracle E-Business Suite user session ID and will reuse that value to render the poplist. Since the decrypt method also uses the current Oracle E-Business Suite user session ID as the key and the encrypted value that is returned is a cached value, the decrypt method will fail and return a null value. To avoid this scenario, you should always disable poplist caching when you encrypt the poplist values in their view object row getters.

2. **When the poplist data (the query string or WHERE clause parameters of the poplist view object) keeps changing with user or thread-specific context.**

   Only data that can be shared across user threads within the JVM should be stored in the cache. Otherwise, the cache size increases redundantly, thereby increasing memory consumption.

Dynamic Poplists

Dynamic poplists are always associated with repeating rows of data, such as in a classic table, advanced table, HGrid or Gantt chart. They display a different set of choices in each row, usually based on some other attribute of the same row. To make a poplist dynamic, use the method `setListVOBoundContainerColumn` on `oracle.apps.fnd.framework.webui.beans.message.OAMessageChoiceBean` to provide the mapping between the bind variables in the poplist view object's query and columns under the container.

Consider a container, such as a table, that has two columns. One column displays a department, while the other column displays all the employees of that department in a poplist. The view object used for the poplist (`Pick List View Definition`) needs the department number (the "Deptno" field of the container) bound for bind variable :1. This can be done with the following code example in the controller's processRequest method:

```
OATableBean table = ...
OAMessageChoiceBean empPoplist =
  (OAMessageChoiceBean)table.findChildRecursive("EmpName");
empPoplist.setListVOBoundContainerColumn(0, /* bind index */
  table,"Deptno");
```

This code binds the first (0th) bind index of the employee poplist view object to the value displayed by the column named Deptno in the table.

The method `setListVOBoundContainerColumn` can be called multiple times if the query used for the poplist has multiple bind variables. Note that this approach effectively simulates a BC4J view link except that it relies on the UI table columns for the bind values rather than the view object attribute values directly.

Note: To show different poplist data for different rows, OA Framework disables poplist caching. See the Javadoc for the `setPickListCacheEnabled` method on `oracle.apps.fnd.framework.webui.beans.OAWebBeanPickList` for details on the effect of disabling poplist caching.

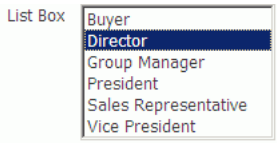
Touch Device Considerations

See a summary of considerations for the Poplist feature on touch devices in the Mobile Applications topic.

List Box

A list box lets a user select one or more values provided by an associated view object. By default, the list box lets users choose a single value; you can configure it to select multiple values as shown in the code example below.

Figure 4: Example of a list box with one selected value.



You must create list boxes programmatically using the `oracle.apps.fnd.framework.webui.beans.form.OADefaultListBean` as shown in the following example.

```
import oracle.apps.fnd.framework.webui.beans.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.form.OADefaultListBean;
...

public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   // Always call first

   super.processRequest(pageContext, webBean);

    // Create the list bean (be sure to give it a name).
     OADefaultListBean list =
       (OADefaultListBean)createWebBean(pageContext,
                                        OAWebBeanConstants.DEFAULT_LIST_BEAN,
                                        null, "positionsList");
```

```
    // Specify the view object that you will use to populate the list bean values.
    list.setListViewObjectDefinitionName("oracle.apps.fnd.framework.toolbox.poplist.server.PositionsVO");

    // Specify the display and internal value attributes in the associated view    object.
    list.setListValueAttribute("LookupCode");
    list.setListDisplayAttribute("Meaning");

    // Configure the list box to allow selection of multiple values.
    list.setMultiple(true);

    // Even though you created the component with an ID, you must explicitly
    // make this call to setName().
    list.setName("posList");

    // In this example, we're adding the list box to a messageComponentLayout region, so we
    // get the declaratively defined messageLayout region that will contain this
    // component. Note that we declaratively set a prompt for the list box on the
    // messageLayout region.
    OAMessageLayoutBean listBoxLayout =
      (OAMessageLayoutBean)webBean.findChildRecursive("ListBoxLayout");
    listBoxLayout.addIndexedChild(list);

}
```

Note that you cannot simply add more items directly to the list box; it must obtain its values from a view object. If you don't want to query your view object values from the database, you can manually populate values in a transient view object as described in View Objects in Detail.

To identify the user's selection when the page is submitted, you would add the following logic to your `processFormRequest()` method:

```
 OADefaultListBean list =
   (OADefaultListBean)webBean.findChildRecursive("positionsList");
 String name = list.getName();

 String[] selectedValues = pageContext.getParameterValues(name);
```

To retrieve all the values in the list box, call `list.getOptionsData()`.

with the focus on list box, pressing [Up/Down Arrow] will shift the focus to the next item and select it. If Multi-Select Allowed is enabled,then hold shift key while pressing [Up/Down Arrow] to select multiple items

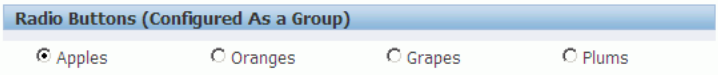
Accessibility Considerations

Keyboard navigation is as follows: with the focus on the list box, pressing [Up/Down Arrow] will shift the focus to the next item and select it. If Multi-Select Allowed is enabled, you can select multiple items by holding [Shift] while pressing [Up/Down Arrow].

Radio Group / Buttons

Radio button items can be added to your page individually, or as a named set with mutually exclusive values. In this case, you can place the individual radio buttons anywhere on the page, and they still behave as a set.

Figure 5: Example of individual radio groups configured as a mutually exclusive set.



A radio group, on the other hand, is like a poplist, but always renders as a vertical list of radio buttons.

Figure 6: Example of a radio group displaying mutually exclusive values for a payment terms view object.



Radio Buttons Implementation

To add an individual radio button to your page:

Step 1: Create an item and set its `Style` to **messageRadioButton**. Assign an `ID` in accordance with the OA Framework Package / File / Directory standards.

Step 2: Set the `Prompt` to the label you want to display for your radio button.

Step 3: (optional) If the checkbox should read/write its selected value from an underlying data source, set the `View Instance` and `View Attribute` values accordingly as you would for any other data entry component.

Step 4: Set the `Checked Value` and `Unchecked Value` properties as appropriate for your data if you want them to be something other than the default "on" (for checked) and null (for unchecked). Optionally set the set the `Initial Value` property to whichever value represents the desired state.

If you want your radio buttons to behave as a group, you must programmatically assign them all the same name by calling `setName()` for each radio button (note that calling `setUINodeName()` doesn't work) in your `processRequest()` method. You must also assign an identifying value for each bean (you cannot set this declaratively). For example, in the ToolBox Sample Library `oracle.apps.fnd.framework.toolbox.samplelib.webui.StandardWidgetsCO` controller, we configure a set of radio buttons as follows:

```
// Configure the radio buttons as a group by 1) assigning them all the
// same name and 2) assigning values to each radio button.
```

```
OAMessageRadioButtonBean appleButton =
      (OAMessageRadioButtonBean)webBean.findChildRecursive("GroupButtonOne");
appleButton.setName("fruitRadioGroup");
appleButton.setValue("APPLES");

OAMessageRadioButtonBean orangeButton =
   (OAMessageRadioButtonBean)webBean.findChildRecursive("GroupButtonTwo");
orangeButton.setName("fruitRadioGroup");
orangeButton.setValue("ORANGES");

OAMessageRadioButtonBean grapeButton =
   (OAMessageRadioButtonBean)webBean.findChildRecursive("GroupButtonThree");
grapeButton.setName("fruitRadioGroup");
grapeButton.setValue("GRAPES");

OAMessageRadioButtonBean plumButton =
   (OAMessageRadioButtonBean)webBean.findChildRecursive("GroupButtonFour");
plumButton.setName("fruitRadioGroup");
plumButton.setValue("PLUMS");
```

You can then obtain the selected radio button in your `processFormRequest()` as follows:

```
String radioGroupValue = pageContext.getParameter("fruitRadioGroup");
```

Note: If you configure individual radio buttons as a group, never select more than one radio button declaratively or programmatically. In this case, only the last radio button selected renders as being selected.

Radio Group Implementation

Configuring a radio group is similar to configuring a poplist:

Step 1: Create a view object to be used by your radio group.

- Per the OA Framework Package / File / Directory standards, you should create this object in a `*.poplist.server` package. For example, we created this ToolBox Tutorial `PositionsVO` poplist in the `oracle.apps.fnd.framework.toolbox.poplist.server` package.
- As described above, the view object should have two attributes: one for the poplist display value, and one for the internal developer key.

Step 2: Create an item whose `Item Style` is **messageRadioGroup** . Assign an `ID` in accordance with the OA Framework Package / File / Directory standards.

Step 3: Apply an attribute set as required by the OA Framework View Coding Standards. See Implementing the View for additional information about attribute sets.

Step 4: Specify the `Prompt` (if not specified by the attribute set).

Step 5: Specify the view object that you created in Step 1.

- If you want a radio group's data to be cached on the JVM, set the `Picklist View Definition` property to the fully qualified name of the view object definition. For example, **oracle.apps.fnd.framework.toolbox.poplist.server.PositionsVO**. Cached radio groups are shared by all users of the JVM, and are identified by a unique key comprised of the entire radio group view object SQL query string, WHERE clause parameter values, language, and orgID.
- If you want your radio group data to be used exclusively by the current session, set the `Picklist View Instance` property instead (for example, **PoplistVO1**). In this case, remember to add your poplist view object to the root UI application module for the page (or application module for the shared regions) that will include this radio group.

Step 6: Map the radio group to its view object display and developer key attributes.

- Set the `Picklist Display Attribute` to the view object attribute whose value you want to display in the radio group.
- Set the `Picklist Value Attribute` to the view object attribute whose value you want to use as the internal developer key.

Step 7: (optional) If the radio group should read or write its selected value from an underlying data source, set the `View Instance` and `View Attribute` values accordingly, as you would for any other data entry component.

Step 8: Set the `Initial Value` if you want the radio group to render with a default selected value. Remember that this effectively performs a selection of the `Picklist Value Attribute` value, and must therefore be among valid values for the underlying view object attribute.

Step 9: Set the `Required` property to **yes** if you want the required indicator to render for the radio group.

Note: If the `Initial Value` is set on the Radio Group, or there is an existing value from the underlying data source specified in Step 7 above, then the `Required` property of the radio group is set as **False** (i.e. not required), and the required indicator is not rendered. Be aware that both 'required' and 'not required' are valid inputs. Validation code should be done by the product team.

Checkbox and Switch

Checkbox

A checkbox is a simple widget that has a "selected" and an "unselected" state. You can add a single checkbox to a region, or you can add a group of checkboxes identified with a label as shown below.

Figure 7: Example of a series of checkboxes rendered with a common label.



To add an individual checkbox to your page:

Step 1: Create an item and set its `Style` to **messageCheckBox**. Assign an `ID` in accordance with the OA Framework Package / File / Directory standards.

Step 2: Set the `Prompt` to the label you want to display for your checkbox.

Step 3: (optional) If the checkbox should read/write its selected value from an underlying data source, set the `View Instance` and `View Attribute` values accordingly as you would for any other data entry component.

Step 4: (optional) Set the `Checked Value` and `Unchecked Value` properties if you want them to be something other than the default "on" (for checked) and null (for unchecked). Set the set the `Initially Checked` property to **true** or **False** as applicable.

The easiest way to add a series of checkboxes rendered with a label is to use a **messageComponentLayout** region. Wherever you want to add your checkbox group, add a **messageLayout** region and set its `Prompt` value (this is the checkbox group label). Then add your individual checkboxes as described above. Note that, unlike the radio group options, each checkbox is still a discrete component; they are only related by layout proximity to the shared label.

Tip: To ensure that your **messageLayout** prompt renders at the top of your checkbox list as shown in Figure 7 above, programmatically set its alignment as follows in your `processRequest()` method (you cannot set the vertical alignment for this component declaratively):

```
OAMessageLayoutBean checkBoxLayout =
  (OAMessageLayoutBean)webBean.findChildRecursive("CheckboxSetLayout");
checkBoxLayout.setVAlign(OAWebBeanConstants.V_ALIGN_TOP);
```

Note: Set the `Required` property to **yes** if you want the `Required` indicator to render for the checkbox. Be aware that both 'required' and 'not required' are valid inputs. Validation code should be done by the product team.

In 12.2.10, both the Toggle Button and Toggle Switch can both be set at design time, programmatically and via personalization. Also they are not limited to only be a part of List Search. Any page that has a checkbox can be personalized to display as a toggle button or toggle switch.

Display Mode Options

There are three variations: checkbox, switch button, and toggle button, with the following options at design time for Display Mode.

- (Default): The component remains messageCheckBox, which has a standard interface, so the default UI corresponds to the (Default) value
- Button: (Button = Toggle Button)
- switchButton: (switchButton = Toggle Switch). Introduced in Release 12.2.10, and described further below.

Programmatically, at runtime the option for setDisplayMode is:

- default: setDisplayMode(default);

The following properties are used in the rendering of the toggle button:

Property Name	Valid Values	Comments
Display Mode	Default, button.	• When button mode is set, the UI renders the checkbox as a button set. • If default mode, it will render as a normal check box (HTML check box). This is not supported for List Search. Developers are expected to add a button mode.
True Value	The code that can be used to execute the query. True value attribute. Display name is 'Yes' (default). If this is left blank, the value defaults to 'On'.	• When selected, this value is bound to the VO Query.
False Value	The code that can be used to execute the query. True value attribute. Display name is 'No'(default) If left blank, the value defaults to 'Off'.	• When selected, this value is bound to the VO Query.
Initial Value	The default value that can be set if there is a True Value or False Value.	• If set, the page renders with this selection initially. If it is not set, the default value will be False Value. • Inline with HTML check box.
Prompt	The label for the field.	• Same as in the HTML checkbox.

In addition, note the following points:

- Other properties such as Action Type, PPR are not supported for this type of rendering. An HTML checkbox should be used instead.
- The query condition is defaulted to IS, and there is no way to change the condition in the UI.
- OA Framework executes the query with an object value of 'True' or 'False' value based on the selected state in the UI.

Switch

With Release 12.2.10, the checkbox type of Off/On action was given an additional form of display that can be used as either:

- An alternative to checkboxes in interfaces for touch devices.
- An addition to checkboxes in interfaces for non-touch devices.

The new offering is referred to as the **switch** component, and offers a more modern user experience on both touch and non-touch devices.

The following screenshots illustrate the new component in use.

Figure 8: Traditional use of checkboxes on non-touch device.

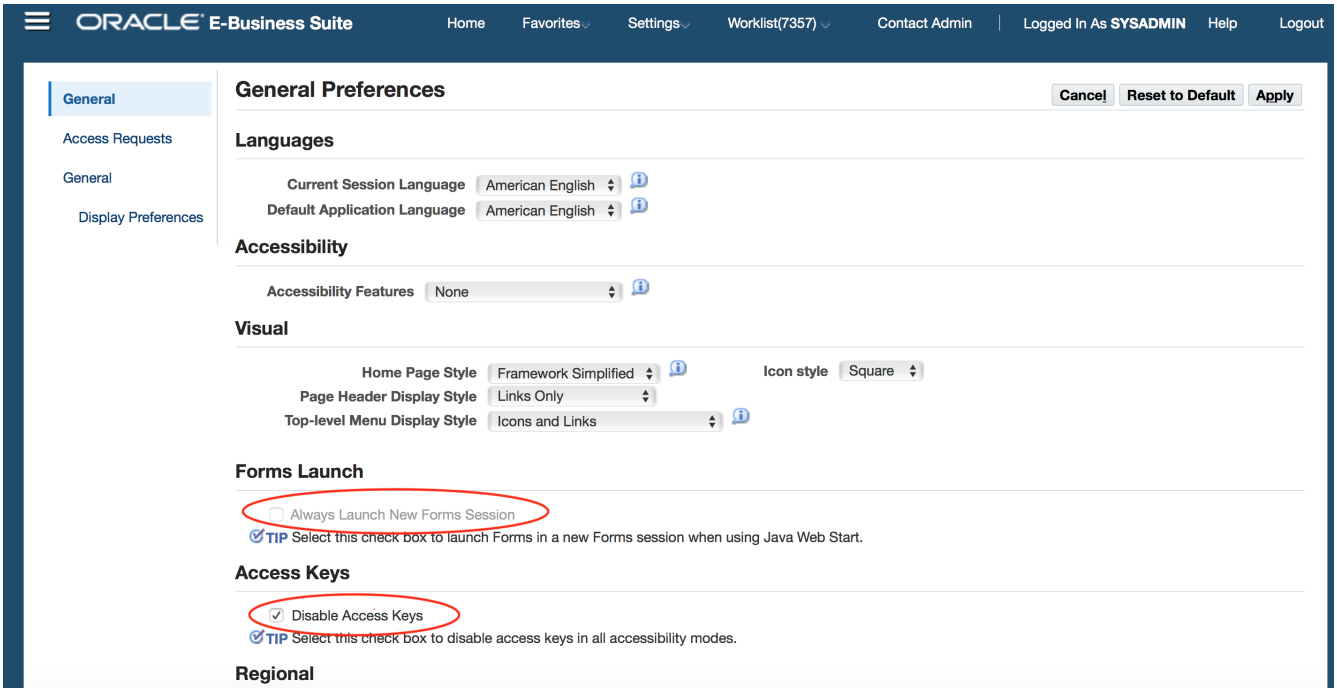


Figure 9: Two example of switch component, first set to Off and second to On.

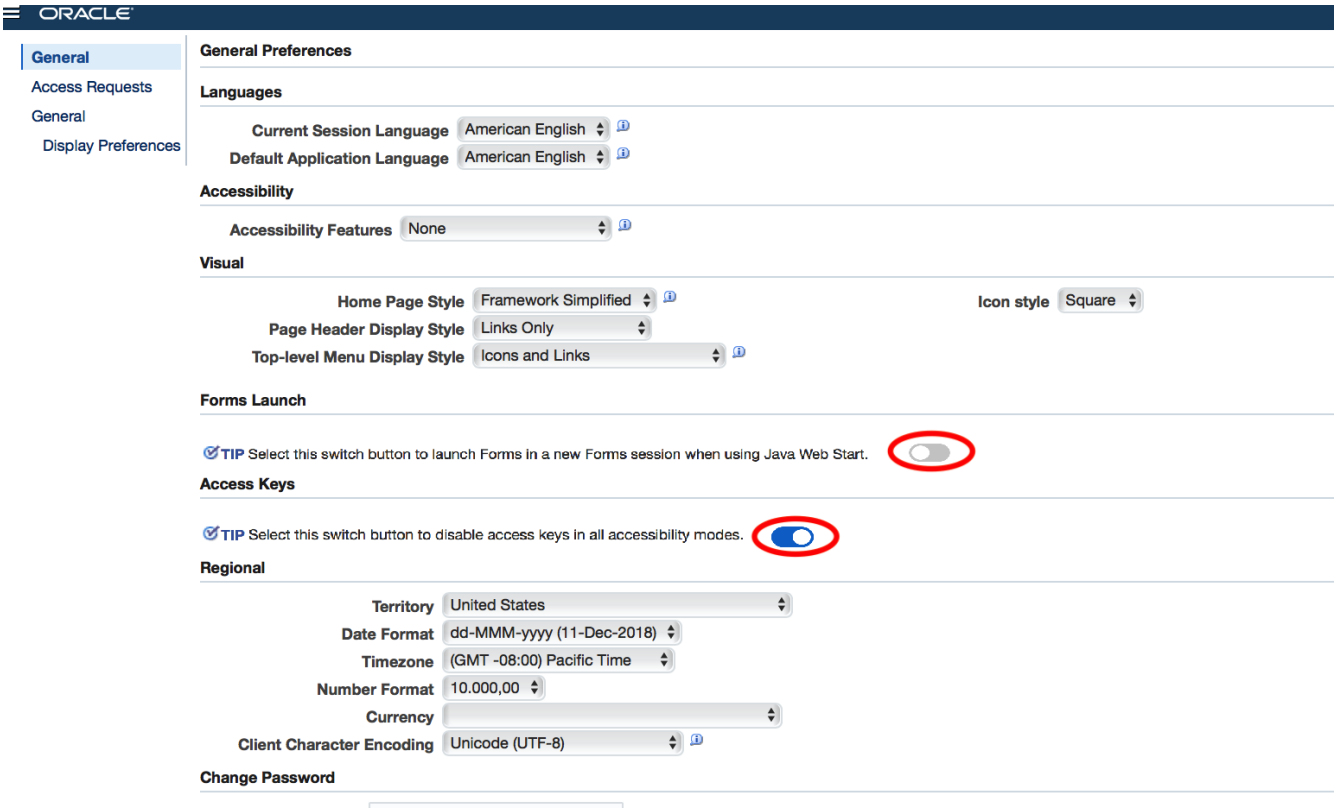


Figure 10: Closeup of switch component set to On.



Figure 11: Closeup of switch component set to Off.

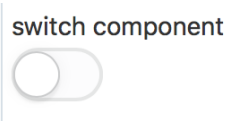


Figure 12: Switch component set to On in an Apple iPad display.

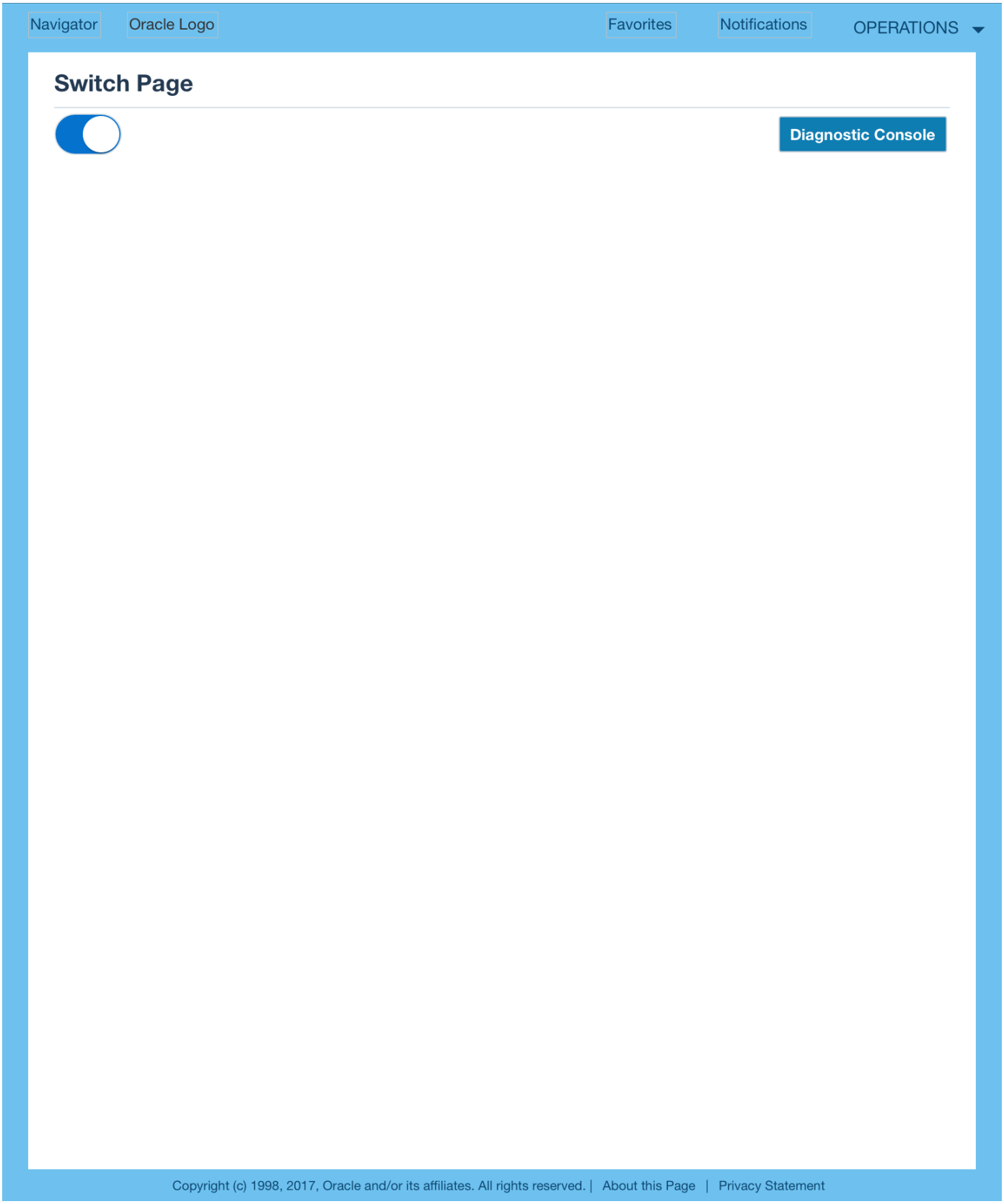


Figure 13: Switch component set to Off in an Apple iPad display.

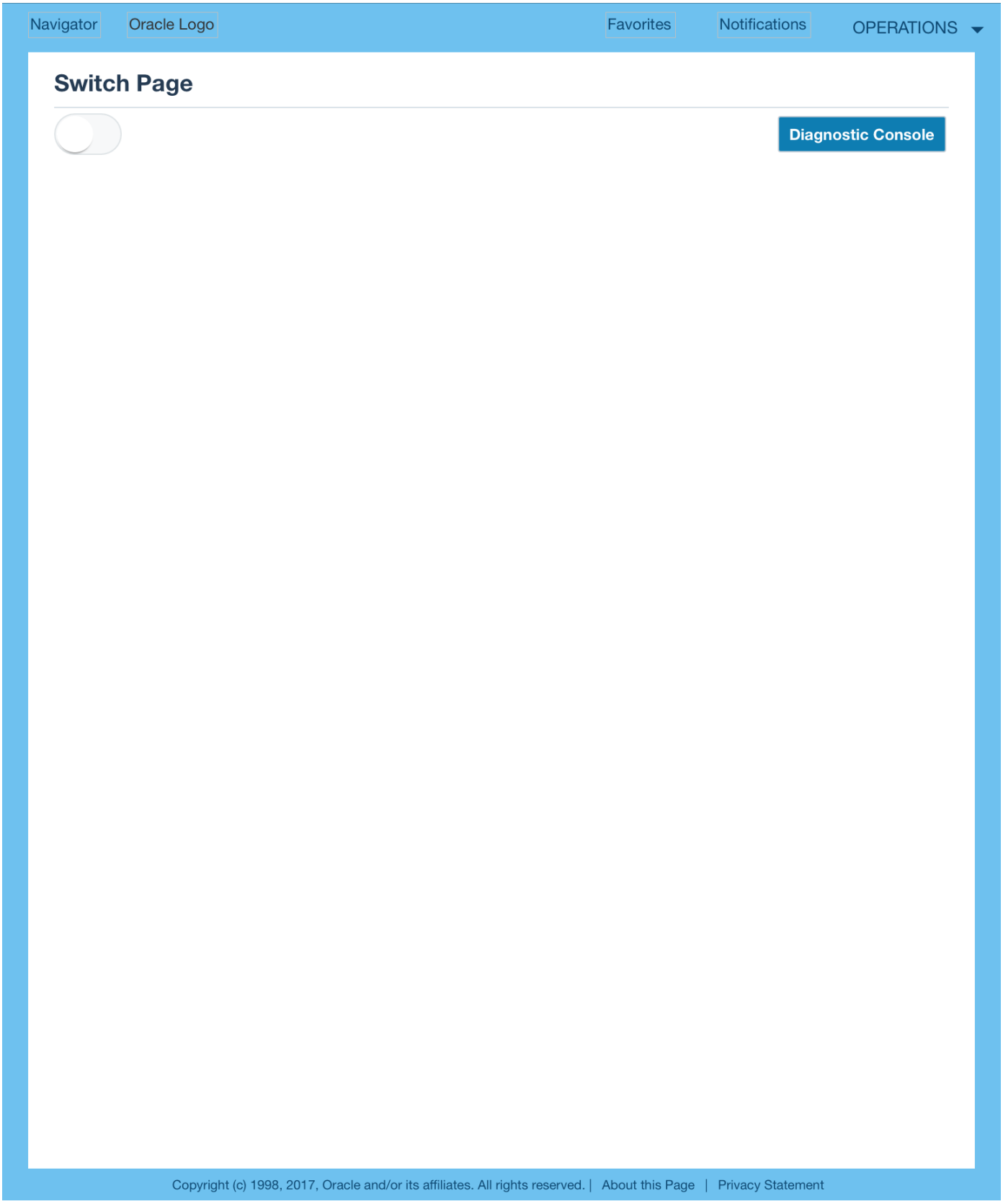


Figure 14: Switch component set to On in a desktop (Mac) display.

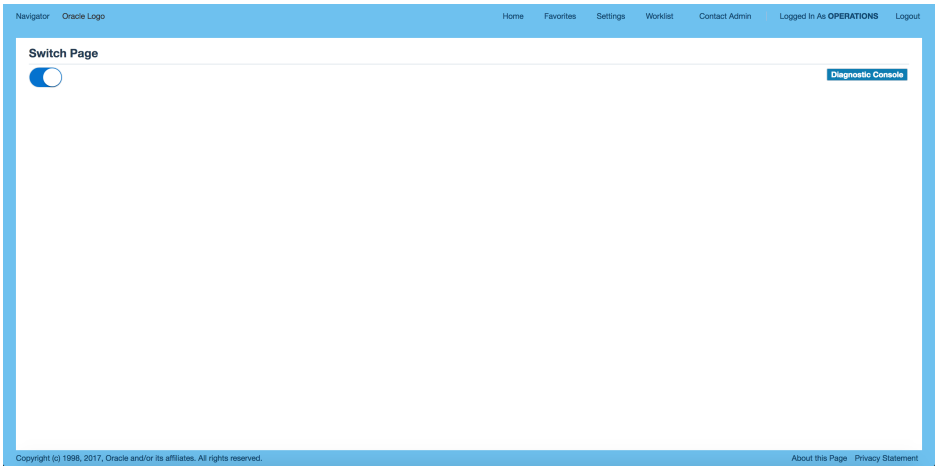


Figure 14: Switch component set to Off in a desktop (Mac) display.

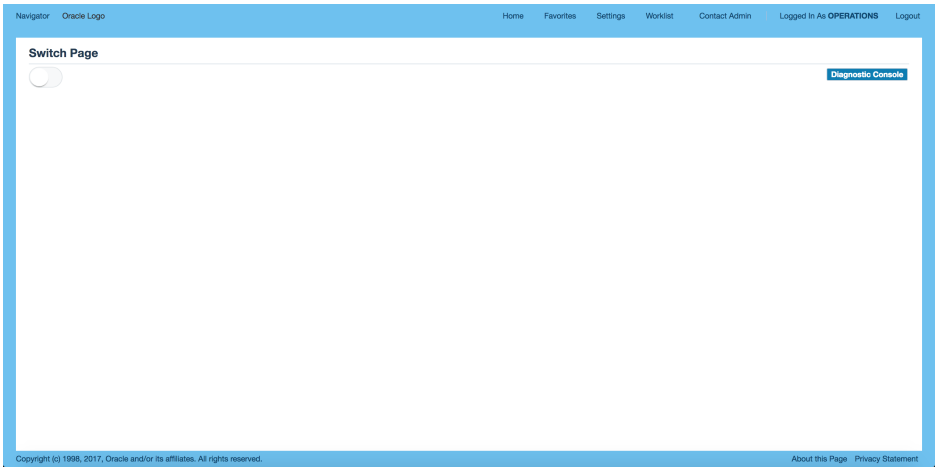


Figure 15: Checkbox Button Properties Required for Switch Display

General	
ID	checkbox
Item Style	messageCheckBox
Extends	
Attribute Set	
Comments	
Functional	
Required	no
Read Only	False
Disabled	False
Search Allowed	False
Sort Allowed	no
Initial Sort Sequence	none
Disable Server Side Validation	False
Selective Search Criteria	False
Data	
Data Type	VARCHAR2
Initial Value	
Checked Value	
Unchecked Value	
Initially Checked	False
BC4J	
View Instance	
View Attribute	
Sort By View Attribute	
Export View Attribute	
Navigation	
Destination Function	
Warn About Changes	True
Visual	
Prompt	

Spin Box

From Release 12.2.4, you can add a spin box to a page. A spin box is a widget that allows a user to select a numeric value from a predefined range. You can specify a minimum and maximum value to define the numeric range of the spin box and you may specify the step size by which users can increase or decreases its value.

1152

A user specifies a numeric value by selecting the Increase or Decrease icon or by entering a numeric value within the predefined range, directly into the text field of the spin box.

If you associate an action with the spin box, the page submits once the user specifies a value in the spin box and tabs out of the field. If you do not associate an action with the widget, the numeric value submits only when a subsequent action submits the page.

Accessibility Considerations

If you have the **Self Service Accessibility Features** profile set to **Standard Accessibility** or **Screen Reader Optimized**, you may still use your keyboard to access a spin box. When the focus (cursor) is on the spin box input field:

- [Up Arrow] - increases the current value by the defined step size, as long as the maximum value has not been reached. If the maximum value displays, nothing happens.
- [Down Arrow] - decreases the current value by the defined step size, as long as the minimum value has not been reached. If the minimum value displays, nothing happens.
- [Tab] - If you associate an action with the spin box, the action triggers.
- [Page Up] - same as [Up Arrow].
- [Page Down] - same as [Down Arrow].
- [Home] - set to maximum value.
- [End] - set to minimum value.

Declaration Implementation

Step 1: Create an item and set its `Style` to **messageSpinBox**. Assign an `ID` in accordance with the OA Framework Package / File / Directory standards.

Step 2: Specify values for the following properties:

- `Prompt` - the label you want to display for your spin box.
- `Rendered` - set this to **True**.
- `Read-only` - set this to **False**.
- `Disabled` - set this to **False**.
- `MinValue` - the minimum value that a user may input. Supports any valid double or long value.
- `MaxValue` - the maximum value that a user may input. Supports any valid double or long value.
- `Stepsize` - the value that is added to or subtracted from the current value when a user selects the Increment or Decrement icon. Default value is 1. Supports any valid double or long value.
- `Action Type` - set to the action type ("fireAction", "firePartialAction" or "none") to fire when a user specifies a value for the spin box.

Step 3: (optional) If your spin box reads and/or writes to an underlying data source, set the `View Instance` and `View Attribute` names for the underlying data source.

Step 4. (optional) As of Release 12.2.6, set the `Placeholder Text` property if you wish to show a short hint describing the expected value for the spin box input field.

Runtime Control

To create a spin box programmatically, use the `oracle.apps.fnd.framework.webui.beans.message.OAMessageSpinBoxBean` as shown in the following example.

```
import oracle.apps.fnd.framework.webui.beans.OAWebBeanConstants;
import oracle.apps.fnd.framework.webui.beans.message.OAMessageSpinBoxBean;
...

public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
   // Always call first
   super.processRequest(pageContext, webBean);

   // Create the message spin box webbean.
   OAMessageSpinBoxBean spin = new OAMessageSpinBoxBean();

   // Specify spin box ID and Name
   spin.setID("myspinbox");
   spin.setUINodeName("myspinbox");

   // Configure the max, min, and step size to allow user to select
   // a value between the defined range
   spin.setMaxValue(3000);
   spin.setMinValue(1000);
   spin.setStepSize(100.5);

   spin.setColumns(4);
   spin.setRequired(REQUIRED_YES);

   //Specify a prompt that will show beside the spin box
   spin.setPrompt("New Salary");

   // Specify the view object that you will use to populate the spin box webbean values.
   spin.setViewUsageName("EmployeeCreateVO1");
   spin.setViewAttributeName("EmployeeId");

   // As spin box supports client action, here we specify a FirePartialAction for the spin box
   // with event 'spinUpdateEvent'
   spin.setPrimaryClientAction(spinbox, new FirePartialAction("spinUpdateEvent", true, new String[]{"myspinbox"}));
```

```
    // To add the spin box to a messageComponentLayout region,
    // get the declaratively defined messageComponentLayout region that will contain this
    // component.
    OAMessageComponentLayoutBean main = (OAMessageComponentLayoutBean)webBean.findChildRecursive("MainRN");
    main.addIndexedChild(spin);
}
```

Personalization Considerations

- See a summary of Standard Web Widgets personalization considerations in the Oracle Application Framework Personalization Guide.

Related Information

- BLAF UI Guideline(s)
  - Standard Web Widgets
- Developer's Guide
  - Dynamic User Interface
  - Submitting the Form
  - Save Model
  - Inline Messaging, Tips, Hints and Bubble Text
  - List of Values
  - Date Picker
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageChoiceBean` (poplist)
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageCheckBoxBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageTextInputBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageRadioGroupBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageRadioButtonBean`
  - `oracle.apps.fnd.framework.webui.beans.message.OAMessageSpinBoxBean`
  - `oracle.apps.fnd.framework.webui.beans.form.OADefaultListBean`
  - `oracle.apps.fnd.framework.webui.beans.OAWebBeanPickList`
- OA Framework ToolBox Tutorial / Sample Library
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.StandardWidgetsPG`
  - `oracle.apps.fnd.framework.toolbox.samplelib.webui.StandardWidgetsCO`

Submitting the Form

Overview

For components that do not inherently perform a form submit, you can force them to do so by associating a `oracle.cabo.ui.action.FireAction` element with it. The `FireAction` element encapsulates all information required for submitting a form including the event name, form post parameters, and the validation that needs to be associated with the form submit. When set on a component, it submits the enclosing form when the component is selected.

The `FireAction` behavior is supported for the following items (in most cases, the form will be submitted when the user selects -- clicks -- the corresponding item):

- link
- button
- resetButton
- selectionButton
- submitButton (already performs a form submit, but you could use this to set parameter values)
- singleSelection
- image
- messageStyledText
- staticStyledText
- messageCheckBox
- messageChoice (will submit the form when the user makes a poplist selection)
- messageRadioGroup
- messageTextInput (will submit the form when the user navigates out of the field)
- Hierarchy column in an HGrid or a Gantt (see the HGrid and Charts / Graphs documentation for special component-specific instructions)

Accessibility Consideration: From an accessibility standards perspective, full page form submits are allowed *only* on those items that normally result in navigation when selected: links and buttons. If you must configure one of the other item types to perform a form submit (a poplist or a checkbox, for example) you must also provide instruction text *above* this item that clearly describes what happens when the item is selected/activated.

Note: Although allowed in earlier OA Framework releases, you should no longer use the UIX `submitForm` API. All new and preexisting code should be converted to leverage the `FireAction` using the instructions below. If you *absolutely cannot* use the `FireAction` solution to force a form submit as required in your page, you can use special form submit bound values.

FireAction Implementation

To configure a component to submit the form that would otherwise not do so (for example, if you want a link item to issue an HTTP POST instead of an HTTP GET), follow these simple instructions.

Declarative FireAction

Step 1: Select the item for which you want to enable the form submission -- or the wrapping link you created in Step 1 above -- in the JDeveloper structure pane to open the property inspector. Set the `Action Type` property to **fireAction**. Ensure that the `Submit` property is set to **True**.

Step 2 (optional): Set the `Event` property to any name that uniquely identifies this item's form submission event (if you are configuring this for a single item on the page, you may use the default value). This is a predefined form parameter that the OA Framework automatically adds to the request when the user performs an action that triggers for the form submit. You'll see in Step 4 how to check this value to ascertain which item raised the event.

Step 3 (optional): If you need to add any supplemental values to the request when the form is submitted, you can quickly create parameters by selecting the **...** button in the `Parameter` property to open a **Parameters** window. In the **Parameters** window, specify the **Name** and **Value** of each parameter that you want to add to the request. Note that the values can be static text, or bound values using SPEL syntax (see Chapter 3: Implementing the View if you need information about SPEL).

Tip: If you need to encrypt your parameter value, you can use the following SPEL syntax variant:

```
${oa.encrypt.<ViewInstanceName>.<ViewAttributeName>}
```

Behind the scenes at runtime, the OA Framework creates an `oracle.apps.fnd.framework.webui.beans.form.OAFormParameterBean` for each parameter you define in this step.

Note: You should leverage this feature instead of adding individual **formParameter** items to the page yourself.

Step 4: Implement event handling code in the `processFormRequest` method. For example, the following code shows how to

check for a **fireAction** (submit form) event with the name "SF" for which two parameters were defined:

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
   super.processFormRequest(pageContext, webBean);

   if ("SF".equals(pageContext.getParameter(EVENT_PARAM)))
   {
     String fixedValue = pageContext.getParameter("FixedValue");
     String boundValue = pageContext.getParameter("BoundValue");
   }
}
```

If your page was defined prior to OA Framework Release 11.5.9, then please read the 11.5.10 release notes for instructions on how you can convert your submitForm to a FireAction.

Programmatic FireAction

If the parameters or the values that are required to submit the form need to be computed programmatically, then you can create and set the fireAction element on your bean at runtime using the following method.

```
public void setFireActionForSubmit (String eventName, Hashtable params, Hashtable
paramsWithBinds,
boolean clientUnvalidated, boolean serverUnvalidated)
```

Parameter Name	Description
`eventName`	The name of the event that is generated by this submit. The default event name is `update`.
`params`	Hashtable of `java.lang.String` name-value parameters.
`paramsWithBinds`	Hashtable of parameters whose values are databound. You can use the `OADataBoundValueFireActionURL` class for substituting any tokens that are associated with your data source (view object) for this purpose. This is explained in detail in the example below.
`clientUnvalidated`	Defines whether or not validation is performed on the client (browser) when the action is invoked. You should set this to **false** if you want to enable validation.
`serverUnvalidated`	Defines whether or not validation is performed on the server when the action is invoked. You should set this to **false** if you want to enable validation.

Tip: Make sure that the attribute values you pass to your client action parameters are not formatted (such as for numbers, currencies, and so on) so they can be handled correctly on the form POSTS. If you have a case where an attribute value is formatted, you should create a new attribute value that mirrors the original. Omit the formatting in the new attribute value and set it's datatype as appropriate to use as the client action parameter.

For example, you want to create a trashcan image as a column of your table and delete a specific row when the trash can image on that row is chosen. You also want to post two parameters with your delete event: param1 that has a value set to the id of the page, and param2 that has the primary key view attribute value for that row. Lets call the name of this viewAttribute as "Sequence". You should be able to set a token for this view attribute that evaluates to the correct view attribute value at runtime.

```
// Get the pageName that you want to submit.
String pageName = pageContext.getRootRegionCode();


// Create a hashtable of string name-value parameters, if any.
Hashtable params = new Hashtable (1);
params.put ("param1", pageName);

// Create a hashtable of parameters that are data bound.
```

```
// You can use the OADataBoundValueFireActionURL to resolve
// any databinding that is associated with the view object.
// The webBean in the constructor is the webBean on which you set
// the fireAction.
Hashtable paramsWithBinds = new Hashtable(1);
paramsWithBinds.put ("param2",
        new OADataBoundValueFireActionURL(webBean, "{$Sequence}"));

// set the fire action on the trash can image. The name of
// the event when the form is posted is "delete".
// The client and the server validation is turned on by
// setting the clientUnvalidated and serverUnvalidated
// properties to false.
trashCanImageBean.setFireActionForSubmit ("delete",
        params, paramsWithBinds,false, false);
```

Alternately, if you need the `FireAction` object for any programmatic access, then you can use the following sample code to do so.

```
// get a handle to the fireAction object.
FireAction fireAction = OAWebBeanUtils.getFireActionForSubmit
       (trashCanImageBean, "delete", params, paramsWithBinds,
        false, false);

// make any additional changes to the FireAction object
....

// finally set it on your bean
trashCanImageBean.setAttributeValue
        (PRIMARY_CLIENT_ACTION_ATTR, fireAction);
```

Bound Value Implementation

See Bound Values in Chapter 4 for a general introduction to the concept and use of bound values in the OA Framework.

OABoundValueSubmit

If you have a use case where you cannot use a `FireAction`, and yet you still must force a form submit, you may use the `oracle.apps.fnd.framework.webui.OABoundValueSubmit` bound value instead. For example, you would use this if you have product-specific (approved!) Javascript before and/or after the form submission.

When you use the `OABoundValueSubmit` object:

- Your request parameters are automatically secured.
- `OAFormParameterBeans` are automatically created for the parameters; you don't need to explicitly create any.

  Note: For every form submit, the values of each `OAFormParameterBean` created for your submit parameters are always cleared and reset as necessary for the relevant form submit, before they are submitted. Therefore, your check for these parameters should include a check for the absolute value, and not just a "not null" check, of the parameter.

  For the same reason, you must not use parameter names that conflict with the parameters names of other submit buttons on the page. See the javadoc for `OAFormParameterBean` for a detailed explanation of its behavior.

For additional information, see the `OABoundValueSubmit` Javadoc.

OABoundValueEnterOnKeyPress

If you have a case where you want to submit the form when the user selects the Enter key, you may use the `oracle.apps.fnd.framework.webui.OABoundValueEnterOnKeyPress` bound value (you associate this bound value with the **ON_KEY_PRESS_ATTR** attribute for an active component that needs to submit the form when Enter key is selected).

Note: Internally, this bound value leverages the `OABoundValueSubmit` so the same behavior with regards to request parameters also applies to this.

For example, assume you want to submit the form when the cursor is in a particular text input field and the user selects the Enter key. The following code shows how to configure this for the text field while adding two parameters (param1 and param2) to the request. See the `OABoundValueEnterOnKeyPress` Javadoc for additional information.

```
import java.util.Hashtable;
import oracle.apps.fnd.framework.webui.OABoundValueEnterOnKeyPress;
import oracle.apps.fnd.framework.webui.OAWebBeanConstants;
...
```

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);
  ...
  Hashtable params = new Hashtable();
  params.put ("param1", "value1");
  params.put ("param2", "value2");
  someTextField.setAttributeValue (OAWebBeanConstants.ON_KEY_PRESS_ATTR,
      new OABoundValueEnterOnKeyPress(pageContext,
                                      formName, // enclosing form name
                                      params, // request parameters
                                      false, // client unvalidated
                                      false)); // server unvalidated

}
```

Blocking on Submit

Whenever a submit action takes place on a page, subsequent submits can be blocked. When using a blocking on submit technique, when the submit action takes place, the cursor becomes busy and prevents any other submit action until the current submit event has been handled. The block on submit behavior is not enabled by default on a page. However, For Partial Page Refresh (PPR) events alone, it is enabled by default.

To implement the block on submit behavior on a specfic page, add the following code to the processRequest() of that page:

```
import oracle.apps.fnd.framework.webui.beans.OABodyBean;
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
     {
super.processRequest(pageContext, webBean);
OAWebBean body = pageContext.getRootWebBean();
if (body instanceof OABodyBean)
     {
((OABodyBean)body).setBlockOnEverySubmit(true);
     }
...
...
     }
.
```

Personalization Considerations

- None

Known Issues

- None

SubTab Navigation

Overview

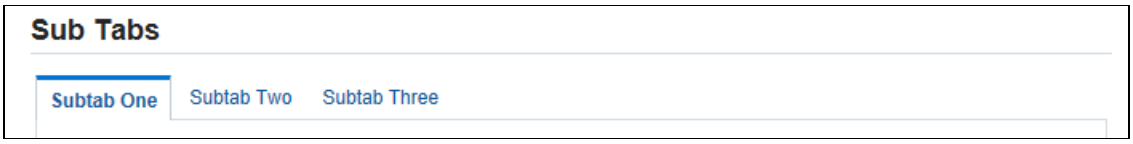
As described the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Tabs / Navigation, the subtab navigation component lets you present a *single page's contents* in a tabbed layout (which should not be confused with the presentation of multiple pages and content areas using application-level tabs). For example, in an "Employee" page, you might use a subtab layout to organize the data into meaningful groups that users can access as needed:

- Identification (displays by default)
- Assignment(s)
- Compensation

In Figure 1, you would replace the title "Sub Tabs" with "Employee" and replace the labels for each sub tab as follows:

- Subtab One > Identification
- Subtab Two > Assignment(s)
- Subtab Three > Compensation

Figure 1: Example of a page with subtabs for managing associated data



Subtab Layout and Partial Page Rendering

When partial page rendering (PPR) is enabled, OA Framework refreshes the entire subtab when you switch between subtabs.

Release 12.2.5 Enhancements

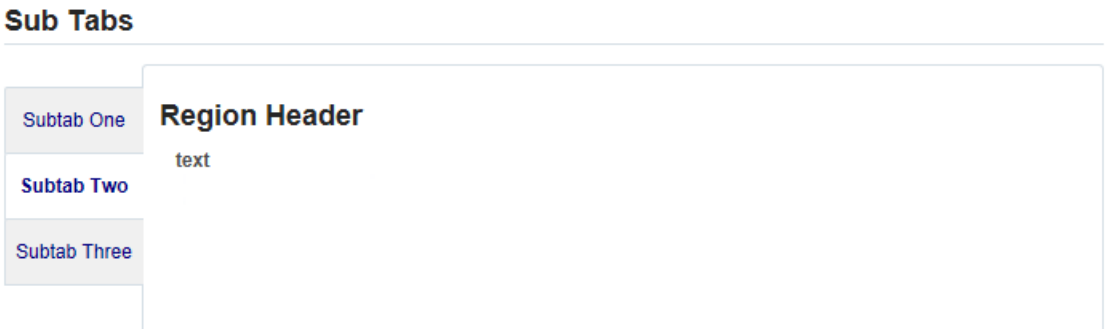
In Release 12.2.5, the subtab navigation component offers the following new enhancements:

- A choice of vertical as well as horizontal (default behavior prior to Release 12.2.5) orientation of the subtab layout. Vertical subtabs render to the left of the page, while horizontal subtabs render above the page.
- A choice of four possible display styles for the subtabs:
  - iconOnly - each subtab only displays an icon. This style enhances usability for mobile touch device users.
  - textOnly - each subtab displays text only (default behavior prior to Release 12.2.5).
  - iconAndTextAdjacent - each subtab displays an icon with adjacent text.
  - iconAndTextBelow - each subtab displays an icon with text below it.

Note however, that certain display styles are valid only for particular orientations, as summarized in the table below.

Orientation / Display Type	Icon Only	Text Only	Icon and Text Below	Icon and Text Adjacent
Vertical	Valid	Valid	Valid	Invalid
Horizontal	Invalid	Valid	Invalid	Valid

Figure 2. Example of a page with vertical subtabs for managing associated data



Declarative Implementation

To add subtabs to your page:

Step 1: Create a **pageLayout** region as you normally would.

Step 2: Select the **pageLayout** region in the JDeveloper Structure pane, right-click and select **New > Region.** Name your new region in accordance with the OA Framework File naming standards, and set its `Style` property to **subTabLayout**.

Step 3: Add one or more container regions to your **subTabLayout** region (for example, you might add three **messageComponentLayout** regions as children of the **subTabLayout** region).

- Each container region that you add corresponds to one of the subtabs, and holds the content to be displayed when that subtab is selected.
- If the container that you add is a **header** or one of its subclasses (**defaultSingleColumn**, for example) it's up to you whether or not you display the header's text beneath the subtab label (if you do, the header text and the subtab text are the same as shown in Figure 2 below). If you would prefer to hide the header's text value as shown in Figure 1 above, set its `Hide Header` property to **True**.
- Each container region that you add (or any of its children) can have its own controller. See Handling Tab Selection below for information about how the OA Framework initializes web bean hierarchy when sub tabs are used.

Figure 2: Example of a header container beneath the sub tab with its Hide Header property set to "False."



Step 4: Add whatever content you want to display beneath each of the sub tab containers.

Step 5: (optional) Set the **subTabLayout** region's `Initial Subtab` property if you want to display any subtab other than the first by default (note that the subtabs are identified using a zero-based index where the first tab is 0, the second is 1 and so on). Be careful not to specify an "out of bounds" index!

Step 6: (optional) Beginning in Release 12.2.5, you can set the **subTabLayout** region's `Orientation` property as **vertical** or **horizontal** to control the subtab layout's orientation. The default is **horizontal**.

Step 7: (optional) Beginning in Release 12.2.5, you can set the **subTabLayout** region's `displayType` property. The valid values are **textOnly**, **iconOnly**, **iconAndTextAdjacent** and **iconAndTextBelow**. The default is **textOnly**. Refer to the chart above to set the `Orientation` and `displayType` properties in valid combinations. JDeveloper flags any invalid combination of these two properties as a validation error.

Step 8: Add the subtab links (the number of links should match the number of content containers that you added in Step 3 above). To do this:

1. Select the **subTabLayout** region, right-click and select **New > SubTabs**. JDeveloper creates a **subTabs** named child.
2. Select the **subTabs** named child, right-click and select **New > SubTabBar**.
3. Select the **subTabBar** region, right-click and select **New > Link**. Set the **link** region's `Text` property to the label that you want to display in the subtab.
4. Beginning in Release 12.2.5, if you previously set the **subTabLayout** region's `displayType` property to **iconOnly**, **iconAndTextAdjacent** or **iconAndTextBelow**, then set the **link** region's `iconURI` property to identify the icon you wish to display in the subtab. Enter the path of the icon's filename, such as /OA_MEDIA/icon.png.

   Important: Icon files may be in PNG or GIF format, but must be saved to the $OA_MEDIA directory.

   Important: You must also create a set of two icons for each subtab. For each icon you create, you must create a matching shaded (greyed-out) icon. It is important that you name the matching shaded icon file with a **_sh** extension. In the rendered page, if a user selects a subtab, the subtab will display its colored icon. When the subtab is deselected, it will display its corresponding shaded icon. You need only specify the non-shaded icon in the `iconURI` property of the **link** region.

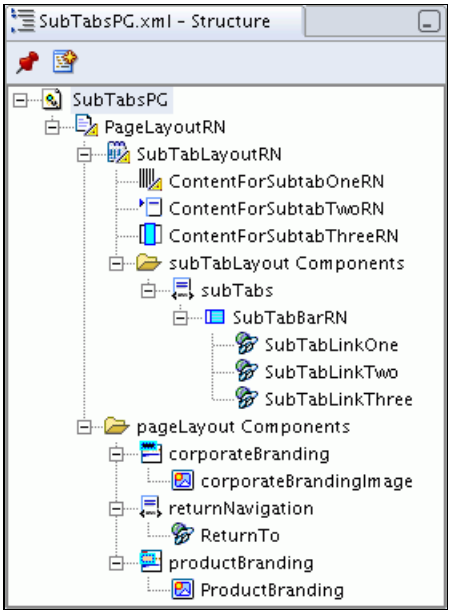
   For example, suppose the `iconURI` property is set to **report_manager.png**. Two icons, **report_manager.png** and **report_manager_sh.png** must be saved to $OA_MEDIA. OA Framework automatically probes for the string "**_sh.png**" to find the colored icon's counterpart so it is not necessary to associate **report_manager_sh.png** with any property on the **link** region.

5. Repeat Step 8 for each subtab content region that you created in Step 3.

Note: Previously, you specified the subtab label by setting the text value in the content containers added in Step 3 (you could not add special subtab links as described in Step 6 above). Any of these subtab layout implementations continue to work as expected.

When you've completed this step, the content in the JDeveloper Structure pane displays as shown below in the ToolBox Sample Library example:

Figure 3: JDeveloper structure of a subTabLayout region



Step 9: Enable partial page rendering (PPR) for the subtabs (if you have not already done so, please read the Dynamic User Interface: PPR documentation). For each link that you defined in Step 6 above, set the `Action Type` property to **firePartialAction**. Make sure the `Submit` property is set to **True**, and specify appropriate values for the `Event`, `Disable Client Side Validation` and `Disable Server Side Validation` properties.

Step 8: Save your work.

Usage Notes / Restrictions

- Each page supports one and only one **subTabLayout** region.
    - Do not try to nest **subTabLayout** regions beneath one another.
    - Do not try to use the Auto-Repeating Layout feature with this region.
    - Do not stack vertical **subTabLayout** regions one above another.
- Only certain Display Types are allowed under a particular orientation. Any attempt to programmatically set an invalid display type for an orientation, will be flagged by a Developer Mode exception.
- While a subtabLayout renders fine in the printable facet, you can not select subtab links in the printable facet.
- You cannot add individual items to the **subTabLayout** (a button, text field, image and so forth). You must add a container region, even if you intend to display just a single item.
- Each subtab is automatically configured as a link which performs a form submission. This configuration ensures that the pending data changes are preserved as the user navigates between tabs (each time the user selects a sub tab, the OA Framework writes any user-entered data back to the underlying view object(s) as described in Implementing the View.
- If you add more content regions than links, the OA Framework doesn't render the regions. If you add more links than content regions, the OA Framework renders the links, but they cannot be selected (clicked) by the user. In both cases, if you are running your page in Developer Test Mode, warnings display.
- To set the selected tab when you forward to this page, or navigate to it with a URL, add the following parameter to the URL: `".......&" + OASubTabLayoutBean.OA_SELECTED_SUBTAB_IDX + "=" + N` where `N` is index of the sub tab to be selected.

  Tip: If you want to set the selected index again in the same request, remember to simply reset the previous parameter value by setting the new value in one of the `setForwardURL*()` methods.

Runtime Control

When the OA Framework instantiates a page with a **subTabLayout**, it create an `oracle.apps.fnd.framework.webui.beans.layout.OASubTabLayoutBean` containing an `oracle.apps.fnd.framework.webui.beans.layout.OASubtabBarBean` with links for each tab. It also directly contains the regions that hold each tab's contents.

The following illustrates the web bean hierarchy for the sub tab components shown in Figure 2 above.

```
OASubTabLayoutBean
 |--------------- OASubtabBarBean (Named child of sub tab layout bean)
     |-------------- OALinkBean (Tab One)
     |-------------- OALinkBean (Tab Two)
     ...

 |-------------- OAMessageComponentLayoutBean (Indexed child, Tab One's container)
 |-------------- OAMessageComponentLayoutBean (Indexed child, Tab Two's container)

 ...
```

With partial page rendering enabled, the `OASubTabLayoutBean` renders displayed subtabs as needed. In this case, when the user selects a subtab:

- The form is submitted as a PPR event.
- If the target subtab has not yet been accessed, the OA Framework recursively calls `processRequest()` on the regions, and their nested regions, within that subtab to render the content.
- The OA Framework then calls `processFormRequest()` on the rendered bean hierarchy as in a regular form submit.

  Note: This means that, for a subtab option selected for the first time, any `processRequest()` and `processFormRequest()` code that you have in the subtab's web bean hierarchy will be executed in sequence.
- Since objects are created as needed, if a user never selects a given subtab, its data objects and web beans are never created.

If you do not enable partial page rendering, by default, the `OASubTabLayoutBean` is configured to render displayed subtabs as needed. In this case:

- The OA Framework creates data objects and web beans only for the selected subtab. If you have a controller associated with each subtab container region, the OA Framework calls the `processRequest()` method only for the content to be displayed (so any initialization code that you have for a given sub tab is executed when the sub tab is selected).
- When a new subtab is selected (and the form is submitted), the OA Framework forwards back to the same page and renders the new subtab.
- Since objects are created as needed, if a user never selects a given subtab, its data objects and web beans are never created.

Alternatively, you can enable a mode whereby the OA Framework creates the entire entire web bean hierarchy and all of the data objects for the subtab regions when the page initially renders:

- The OA Framework calls the `processRequest()` method associated with *all* the nested regions each time the page renders (so any initialization code that you have in `processRequest()` methods associated sub tab regions is executed at the outset).
- When a new subtab is selected (and the form is submitted), the OA Framework does not forward back to the same page. Instead, it calls the `processFormRequest` method in the `OASubTabLayoutBean` and the nested container bean associated with the current selected index is executed.

Instantiate

There is no reason to instantiate an `OASubTabLayoutBean` and its components yourself; you should create this declaratively. However, if you decide to create this web bean programmatically despite our discouragement, you must ensure that on the link web bean, you set the following:

```
set linkBean.setText("Tab");
```

Control Visual Properties

For Release 12.2.5, you may set the orientation and display type programmatically for existing SubTabLayout web beans created in code, however, for new development you should always create SubTabLayout web beans declaratively to leverage their ability to be personalized. Refer to Usage Notes / Restrictions for additional information about setting the display type.

Assuming subTabBean is the web bean which has a handle to OASubTabLayoutBean, you can set the orientation and displayType using the following API:

```
subTabBean.setOrientation(ORIENTATION_HORIZONTAL);
//for horizontal orientation. ORIENTATION_HORIZONTAL is a predefined constant in UIX.
subTabBean.setOrientation(ORIENTATION_VERTICAL);
//for vertical orientation. ORIENTATION_VERTICAL is a predefined constant in UIX.
subTabBean.setDisplayType(ICON_ONLY);
//or any other constant mentioned below.
```

Predefined constants which can be used for display type include:

```
public static final String ICON_ONLY="iconOnly";
public static final String TEXT_ONLY="textOnly";
public static final String ICON_TEXT_BELOW="iconAndTextBelow";
public static final String ICON_TEXT_ADJACENT="iconAndTextAdjacent";
```

If you want to set the current selected (displayed) subtab, call `setSelectedIndex(OAPageContext pageContext, int index)` on the `OASubTabLayoutBean` in a `processRequest()` method. Remember that the index is zero-based.

To hide or show a subtab, use the `hideSubTab()` method in `OASubTabLayoutBean` as shown (note that the following example assumes the controller is associated with a web bean *above* the subtablayout region):

```
processRequest(OAPageContext pageContext, OAWebBean webBean)
{

  // Always call this first.
  super.processRequest(pageContext, webBean);


  OASubTabLayoutBean subTabLayout =
     (OASubTabLayoutBean)webBean.findChildRecursive("<YourSubTabLayoutRegionName>");

  if (subTablayout != null)
  {
   // Hide the third tab
   subTabLayout.hideSubTab(2, true);

   // Hide the first tab
   subTabLayout.hideSubTab(0, true);

   // Show the second tab
   subtabBean.hideSubTab(1, false);
   }
}
```

Control Behavior and Data

Since the subtab links each perform a form submit (which causes data to be written to the underlying data source and validated), you might want to disable validation on the subtab component. To do this, call `setUnvalidated(true)` on the `OASubTabLayoutBean` in a `processRequest()` method (see Implementing the Controller for detailed information about the `setUnvalidated()` method).

Note: You cannot change the destination URL for the sub tabs; they always submit the form to the current page.

To change the subtab creation mode, call one of the following as appropriate:

```
// Enable the full web bean hierarchy sub tab creation mode

OASubTabLayoutBean.setAttribute(MODE_ATTR, SUBTAB_FORM_SUBMISSION_MODE);


// Enable the "as needed" sub tab creation mode.  This is the default mode.
OASubTabLayoutBean.setAttribute(MODE_ATTR, SUBTAB_REDIRECT_MODE);
```

Handling Tab Selection Events

Given the way that sub tabs are implemented, it really isn't necessary for you to handle the sub tab link selection. You just need to make sure that any subtab region initialization is properly implemented in your controllers' `processRequest()` methods.

If you want to know whether the form was submitted due to a subtab click (as opposed to any other event which submits the form), you can call `isSubTabClicked(OAPageContext pageContext)` on the `OASubTabLayoutBean`.

Accessibility Support

Horizontal and Vertical Subtabs

- At runtime, users can navigate from one horizontal subtab to the next by using accelerator keys on the keyboard. Note that the accelerator keys do not cycle to the first subtab from the last subtab or to the last subtab from the first subtab.
  - For **Microsoft Internet Explorer**, use **[Alt]** + **[.]** and **[Alt]** + **[,]**. These keys only focus on the subtab elements but do not activate them. To display the subtab content, you also need to press the **[Enter]** key.
  - For **Google Chrome**, use **[Alt]** + **[.]** and **[Alt]** + **[,]**. These keys select and also activate the next and previous subtab in the browser respectively.
  - For **Mozilla Firefox**, use **[Alt]** +**[Shift]**+**[.]** and **[Alt]** +**[Shift]**+**[,]**. These keys select and activate the next and previous subtab in the browser respectively.
- Users can navigate from one vertical subtab to the next by using the up, down, left or right arrow keys and then press **[Tab]** to access the content within the vertical subtab layout.

Personalization Considerations

- See a summary of SubTab Navigation personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guidelines
  - Tabs / Navigation
- Javadoc
  - `oracle.apps.fnd.framework.webui.beans.layout.OASubTabLayoutBean`
  - `oracle.apps.fnd.framework.webui.beans.layout.OASubTabBarBean`
  - `oracle.apps.fnd.framework.webui.OASubTabLayoutHelper`
- OA Framework ToolBox Tutorial / Sample Library

Switchers (Application and Context)

Overview

A switcher is a control, that allows the selective display of information. There are three main types of switchers available:

- **Application Switchers** - as described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Switchers , allows you to quickly switch back and forth between applications.
- **Context Switchers** - as described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Switchers, allows you to switch to a different context in the page or pages of an application.
- **Table Content Switchers** - enables a table column to have multiple display alternatives, of which only one is ever displayed at runtime. See the Dynamic User Interface document for additional information about this feature.

Contents

- Application Switchers
  - Declarative Implementation
  - Runtime Control
  - Personalization Considerations
- Context Switchers
  - Declarative Implementation
  - Runtime Control
  - Personalization Considerations
- Known Issues
- Related Information

Application Switchers

Declarative Implementation

OA Framework currently does not provide declarative support in implementing an Application Switcher.

Runtime Control

You can only implement an Application Switcher programmatically. The following code, added to the `processRequest()` method in the pageLayoutBean, creates an Application Switcher programmatically and adds it to the global button bar bean, after the global buttons.

```
pageLayoutBean.prepareForRendering(pageContext();
//Global buttons have already bean added.
UINode globalButtonBar = pageLayoutBean.getGlobalButtons();

OAApplicationSwitcherBean appSwitcher = (OAApplicationSwitcherBean)createWebBean(....);
OAOptionBean optionBean1 = (OAOptionBean)factory.createWebBean(pageContext, OPTION_BEAN);
optionBean1.setText(...);
optionBean1.setValue(...);
optionBean1.setSelected(true);
OAOptionBean optionBean2 = (OAOptionBean)factory.createWebBean(pageContext, OPTION_BEAN);
optionBean2.setText(...);
optionBean2.setValue(...);
OAOptionBean optionBean3 = (OAOptionBean)factory.createWebBean(pageContext, OPTION_BEAN);
optionBean3.setText(...);
optionBean3.setValue(...);
appSwitcher.addIndexedChild(optionBean1);
appSwitcher.addIndexedChild(optionBean2);
appSwitcher.addIndexedChild(optionBean3);

globalButtonBar.addIndexedChild(appSwitcher);
```

The `oracle.apps.fnd.framework.webui.beans.nav.OAApplicationSwitcherBean` does not render as a global button, but rather as a choice (also known as a poplist or pulldown) with a label and Go button. The Application Switcher bean takes the indexed option beans as values for the poplist.

Personalization Considerations

- See a summary of Switchers personalization considerations in the Oracle Application Framework Personalization Guide.

Context Switchers

A context switcher allows users to switch the content of a page or subsequent pages based on some specified context.

For example, OA Framework implements a context switcher to allow users to switch responsibilities when they navigate from one page (A) to another (B), where the function context of page B belongs to a responsibility other than the current responsibility. Refer to the Menus and Page Security document for additional information on function security to understand when the Responsibility context switcher may render on your page.

You can implement a context switcher on your page, but be aware that OA Framework can also add a Responsibility context switcher to your page as described in the scenario above. The Responsibility context switcher occupies the same area as any other context switcher you may implement on your page, but renders after it. Refer to the Oracle BLAF UI Guideline: Switchers for proper placement of the Context switcher on the page.

Declarative Implementation

To implement a Context switcher:

Step 1: Create a new standalone region. The region style can be set to any layout style, flowLayout, being the most common. Note that you can even set the style to a non-layout style if a layout is not required for your needs.

Step 2: Add to the region, a poplist containing context values.

Step 3: Add to the region, a Go button.

Step 4: Programmatically create the web bean corresponding to the standalone region you defined.

Step 5: Programmatically set the standalone region web bean as the Context switcher using:

`oracle.apps.fnd.framework.webui.beans.layout.OAPageLayoutBean.setContextSwitcher(UINode)`.

Runtime Control

Other than Step 4 and 5 of the declarative implementation instructions, there are no specific runtime control steps necessary to implement a Context switcher in a page.

Personalization Considerations

- See a summary of Switchers personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- None

Related Information

- BLAF UI Guideline(s)
  - Switchers: Application and Context
- Javadoc File(s)
  - `oracle.apps.fnd.framework.webui.beans.nav.OAApplicationSwitcherBean`

Tabs / Navigation

Overview

In an OA Framework application, the menu of available pages is presented to the user in a tab-based model. Beginning in Release 12.2.5, the menu is simplified with a tablet-optimized layout that improves spacing between user interface elements, provides more margins and padding in the page layout, and provides better visual cues for menu selection. In addition, Release 12.2.5 allows you to choose between a classic menu layout, where the "Level 1" menus display as links (Figure 1), or a new menu layout where the "Level 1" menus display as icons and links (Figure 2).

You can change the display of the "Level 1" menu by setting the profile option FND: Top-Level Menu Display Mode or by specifying the **Top-level Menu Display Style** in the **Preferences** page. For more information about the **Preferences** page, refer to the *Oracle E-Business Suite User's Guide*.

To display the "Level 1" menu as icons and links, you must also assign a seeded icon for each "Level 1" menu item. If you do not specify a seeded icon for a function or menu "Level 1" menu, a default icon displays in its place. All Oracle-seeded icons are found in the $OA_MEDIA directory.

Figure 1: OA Framework ToolBox Tutorial tabs - "Level 1" menus displayed as links.

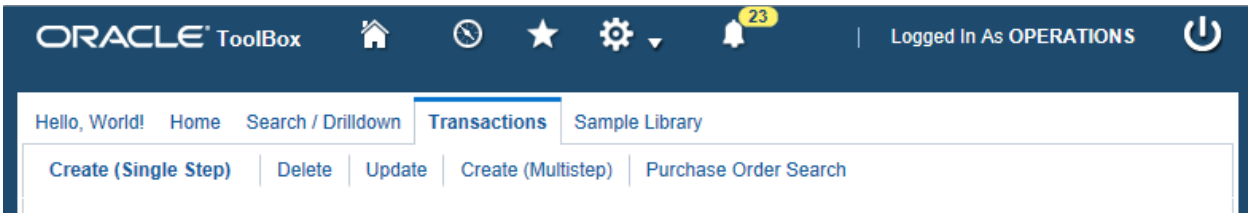
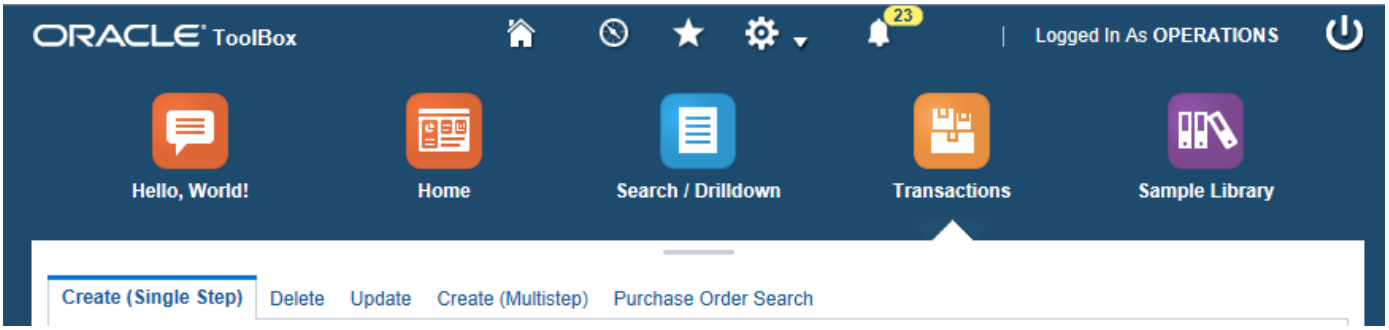


Figure 2: OA Framework ToolBox Tutorial tabs - "Level 1" menus displayed as links and icons.



Within this basic model, individual applications are free to choose from a range of valid design options based on their complexity and expected usage patterns (see Oracle Browser Look and Feel (BLAF) UI Guideline: Tabs/Navigation for a complete description of your design options).

Contents

- Menu Concepts
- Declarative Implementation
- Menu Examples
- Runtime Control
- Rendering a "Level 1" Menu as a Set of Icons
- Accessibility Support
- Personalization Considerations
- Known Issues
- Related Information

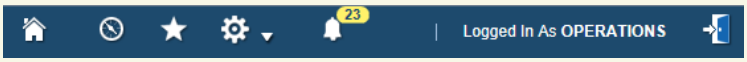
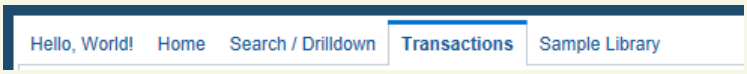
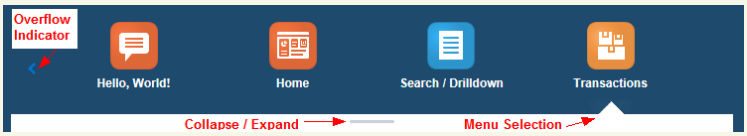
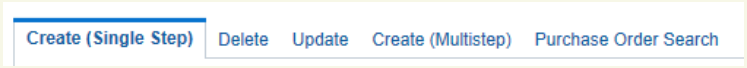
Prerequisite Reading

- Anatomy of an OA Framework Page
- Implementing the View

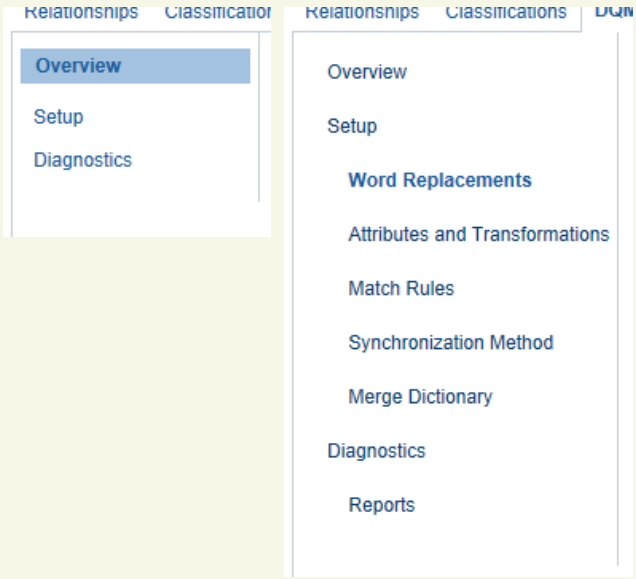
Menu Concepts

Menu Components

Depending on the design, menus are made up of the following components:

Component	Description
Global Menu	 Provides access to tasks and content that are applicable to the entire application. See Buttons (Global) for information on adding global buttons to your application.
Tab	 Represents the highest level of content division within an application (also known as "Level 1" menus) displayed as classic tabs with links.  Represents the highest level of content division within an application (also known as "Level 1" menus) displayed as icons and links. If the window is not wide enough to display all the items of the menu, an Overflow Indicator appears. A notch below the menu designates the currently selected menu. You can also select the Collapse/Expand bar beneath the menu to collapse or expand the "Level 1" menu. If, however: • A user selects a link in the current menu page that navigates to an entirely different top-level menu, the "Level 1" menu expands. • A user selects a link that navigates to a sub-menu within the same "Level 1" menu that was hidden, the "Level 1" menu remains collapsed. • A user selects an action button that stays on the same page, the "Level 1" menu remains collapsed.
Horizontal Navigation	 Filters the content associated with a given tab or icon (as known as "Level 2" menus) For example, we included horizontal navigation beneath the "Transactions" tab or icon to provide access to several different transactions in the ToolBox Tutorial.

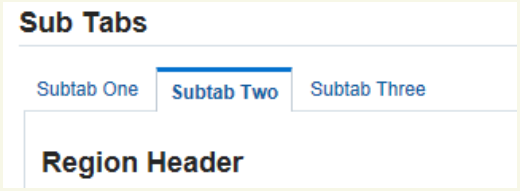
Side Navigation



Filters content or actions (also known as "Level 3" menus). The side navigation can be used below a horizontal navigation element, below a tab without a horizontal navigation element, or on its own without any tabs as a simple list of actions/content.

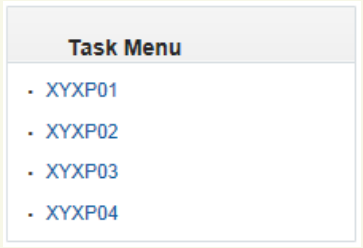
Side navigation menus can also include submenus (Levels, 4, 5, 6 +) that let you organize content into a hierarchy. See the illustration's "Setup" link and its "Word Replacements", "Attributes and Transformations", and other children.

Sub Tab



A tab-like control for switching content or action views in the page's content area. sub tabs can be used with a horizontal navigation element, with a tab and horizontal navigation elements, or with a side navigation.

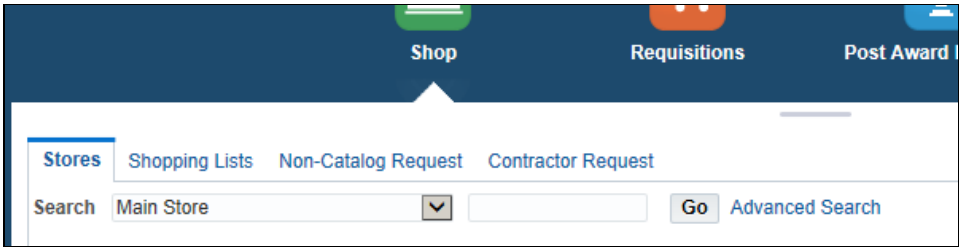
Task / Property Menu



A bulleted list of tasks or properties displayed in the page's content area. The task/property menu is very scalable, and can be used in combination with any of the other menu elements.

Tip: If you want to know how to implement a "Quick Search" region beneath the menu as shown below, this special **pageLayout** property is described in the Search document. You do not define this as part of the menu itself.

Figure 3: "Quick Search" region shown beneath a horizontal navigation menu entry.



Menu Context

When a user navigates to a specific page within an application for the first time (assuming the user has function security access), the OA Framework instantiates a `oracle.apps.fnd.functionSecurity.NavigationContext` object to keep track of the root level application menu and the current selected function so the menu can be rendered correctly. The `NavigationContext` object persists for the life of the session until a user selects a new menu option, or you reset it manually by either calling methods on the `NavigationContext` object at runtime, or by setting URL parameters.

Note: Any function accessed from the Oracle E-Business Home Page (previously known as the Personal Home Page or simply PHP), a portlet, or from a test JSP must include the following URL parameters specifying the "Home Page" (root application) menu, and the page function to highlight in the menu. If you fail to do this, your page will render, but your menus will not.

- `OASF=<SelectedFunctionName>` - this tells the Framework to select this function in the given "Home Page" menu context.
- `OAHP=<HomePageMenuName>` - this is used ONLY with the OASF parameter, and it is used to establish the current menu context. It should point to a "Home Page" menu.

For example, the following URL (split into lines for ease of reading) from the OA Framework ToolBox Tutorial `test_fwktutorial.jsp` illustrates how to use these parameters:

```
<a href="<%=URLMgr.processOutgoingURL("OA.jsp?OAFunc=FWK_TOOLBOX_HELLO
&OAHP=FWK_TOOLBOX_TUTORIAL_APP&OASF=FWK_TOOLBOX_HELLO
&transactionid=" + transactionid, macKey)%>">Hello, World!</a><br>
```

See Setting the Menu Context in the Runtime Control section below for information on manipulating these and other menu context values.

Applications Security

See the Page Security document for information about securing access to the pages in your application menu.

Declarative Implementation

Once you have a menu design, you should plan your implementation. The planning step is important, because you need to build your menus from the leaf nodes to the root menu entry, so you need to know how your menu is to be constructed before you start building it.

As described in Implementing the View, OA Framework application menus are actually comprised of Oracle E-Business Suite functions and menus. First, we look at how to create functions and menus in general terms, then we focus on how to implement specific menu types.

Tip: Menus are cached in the middle tier. If you change a menu for an application that you're running within JDeveloper, remember to terminate the OC4J server (**Run > Terminate**) and re-run your .jsp page to see your menu changes. If you change a menu for a deployed application (as opposed to one that you're running within JDeveloper), remember to bounce the web application server's listener and JVM to see your menu changes.

Creating Functions

Regardless of what kind of menu you need to construct, each and every OA Framework page on the menu *must* have a corresponding function.

Note that a single page can be called by many functions (each potentially passing different parameters through the URL), which means it can be used in many different menus. Generally, it's easiest to create all of your functions before you start building your menus. The instructions below guide you through creating a function using the **Functions** page from the **Functional Administrator** responsibility. You may also create a function using the **Functions** form from the **System Adminstrator** responsibility. While the UI differs between the **Functions** page versus the **Functions** form, both serve the same purpose.

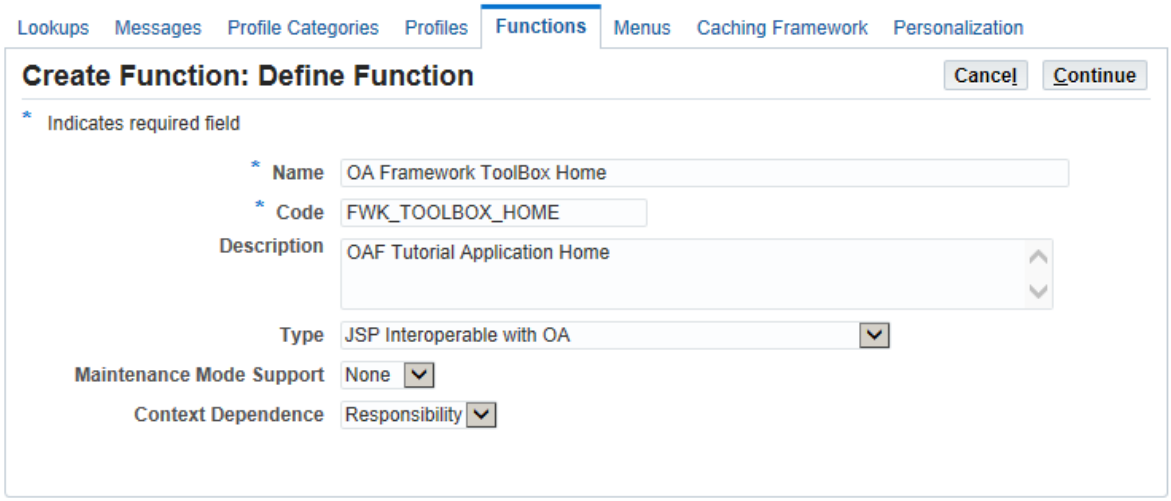
To create a function:

1. Start Oracle E-Business Suite in your development environment and log in as a user with the "Functional Administrator"

responsibility.

2. Select the "Functional Administrator" responsibility.
3. Select **Core Services > Functions** to navigate to the **Functions** page.
4. To add a new function in the **Functions** page, select **Create Function**.
5. Enter values for the following properties in the **Create Function: Define Function** page. Note that you can accept all the other default values.
    - **Name** - this is the unique, user-friendly version of the function name that displays when administering menus (for example, Workflow Monitor Status Diagram).
    - **Code** - this is the unique developer key for the function, and as such it must be a unique value. You can name this whatever you wish, as long as you follow any naming conventions used by your product team (a typical example of this name is: WF_MONITOR_STATUS_DIAGRAM, which is the product short code followed by a descriptive name for the associated page). Note that this is the value that we will use when defining URLs for page access.
    - **Description** - provides a short description of the associated page.
    - **Type** - for OA Framework pages, this should be set to **JSP Interoperable with OA**.

    Figure 4: Create Function: Define Function page.



6. Select **Continue**.
7. Enter values for the following properties in the **Create Function: Details** page.
    - **HTML Call** - provides the mapping to the associated page. At runtime, whenever this function is invoked, the OA Framework knows to display the page identified in this property.

      The value that you specify should comply with the syntax OA.jsp?page=/oracle/apps/<appShortName>/<yourPackages>/webui/<PageName>/ as illustrated in the following examples:

      OA.jsp?page=/oracle/apps/dem/hello/webui/HelloWorldPG and OA.jsp?page=/oracle/apps/fnd/framework/toolbox/tutorial/webui/SupplierPG.

      Note that you can include static URL parameters in your function's HTML Call, or you can include tokens for runtime value substitution:

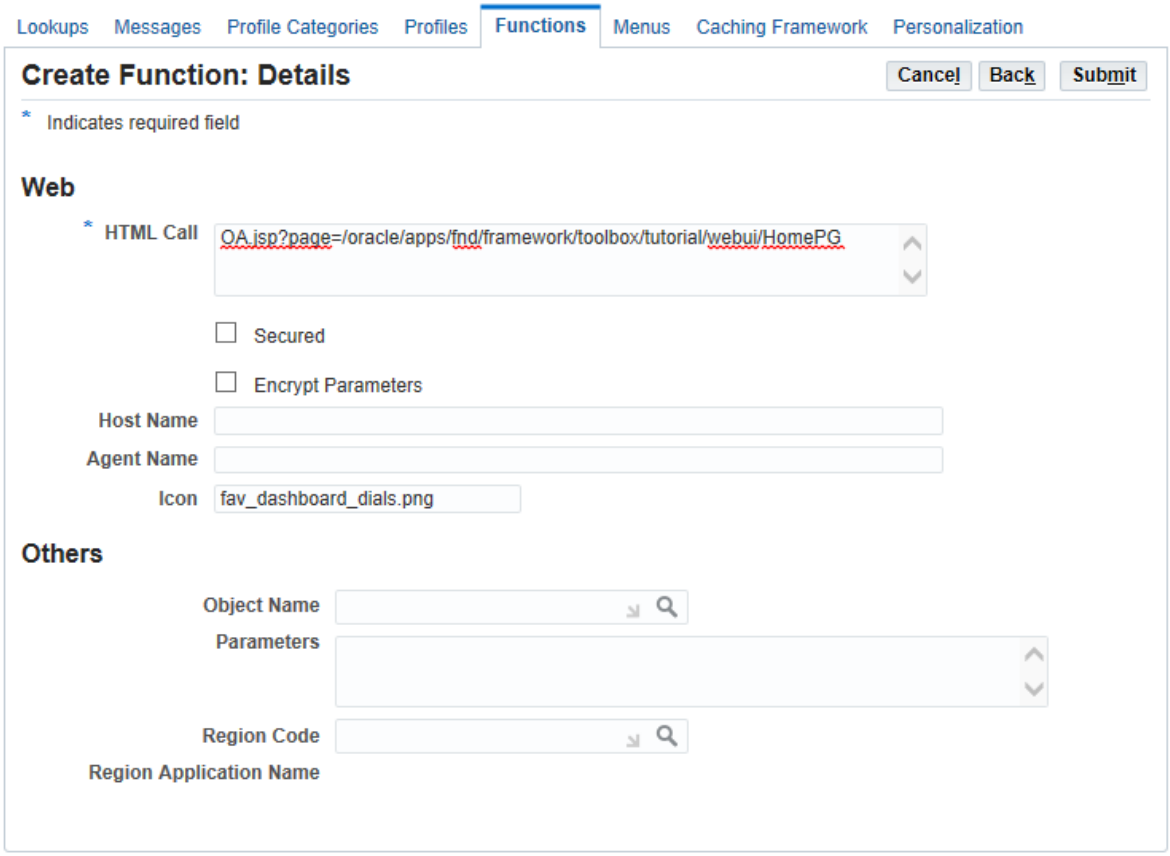
      OA.jsp?page=/oracle/apps/fnd/framework/toolbox/tutorial/webui/SupplierPG
      `&param=staticValue&param2={@tokenValue}`

      If you include tokens, you must explicitly register them with the OA Framework by calling the `oracle.apps.fnd.framework.webui.OAPageContext's setFunctionParameterDataObject(oracle.cabo.ui.data.DataObject tokens)` method in the `processRequest()` method of a controller associated with the target page. The OA Framework obtains the token value from your `DataObject` at runtime by calling `DataObject.selectValue(context, TokenKey)`, and then replaces the token with the value in your menu URL. See the

`OAPageContext.setFunctionParameterDataObject()` Javadoc for additional information including a code sample.

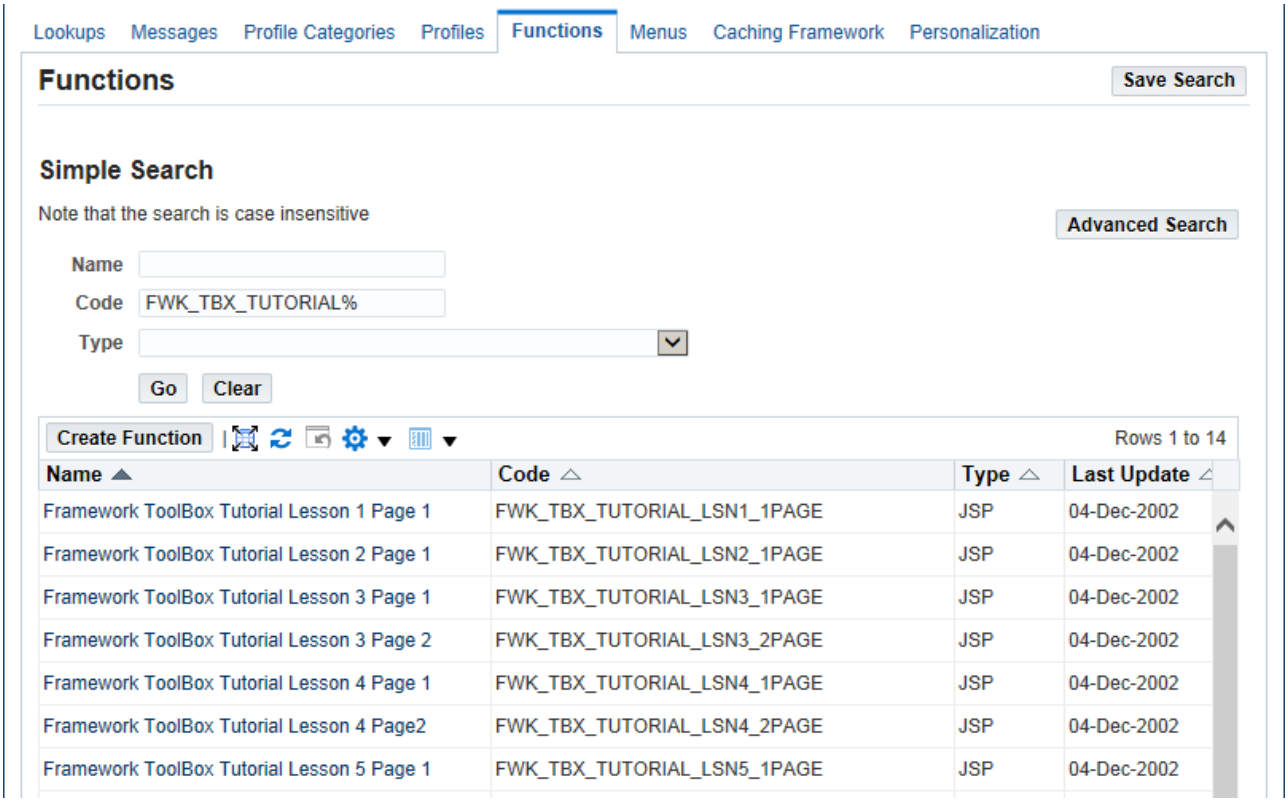
- **Icon** - if the function will be in the "Level 1" menu, provide the name of a seeded icon to assign to the function. The icon file must reside in the $OA_MEDIA directory. This icon displays when the profile FND: Top-Level Menu Display Mode is set to display the "Level 1" menu as icons and links.

Figure 5: Create Function: Details page.



8. Select **Submit** to save your work.
9. You can also query the **Functions** page for existing functions by using the Simple Search or Advanced search panel.

Figure 6: OA Framework ToolBox Tutorial functions displayed in the Oracle E-Business Suite Release Functions page.



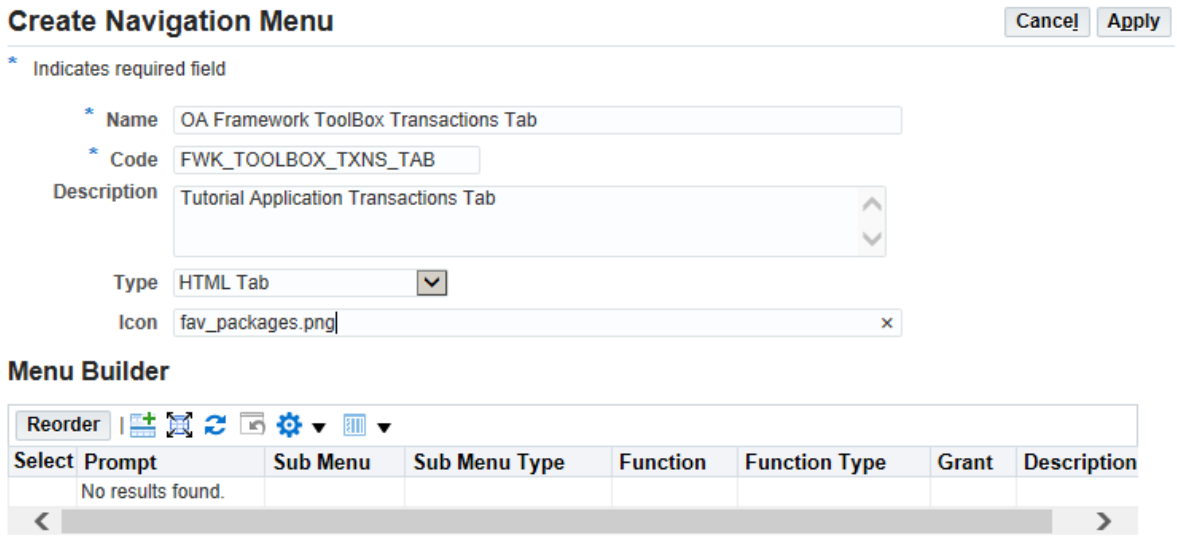
Creating Menus

The instructions below guide you through creating a menu using the **Navigation Menus** page from the **Functional Administrator** responsibility. You may also create a menu using the **Menus** form from the **System Adminstrator** responsibility. While the UI differs between the **Navigation Menus** page versus the **Menus** form, both serve the same purpose.

To create a menu:

1. Log in as a user with the "Functional Administrator" responsibility.
2. Select the "Functional Administrator" responsibility.
3. Select **Core Services > Menus** to navigate to the **Navigation Menus** page.
4. To add a new menu in the **Navigation Menus** page, select **Create Navigation Menu**.
5. Enter values for the following properties in the **Create Navigation Menu** page.
    - **Name** - this is the unique, user-friendly version of the menu name that displays when administering menus (for example, Workflow Administrator Application).
    - **Code** - this is the unique developer key for the menu. A typical example of this name is: WF_ADMINISTRATOR_APPLICATION (the product short code followed by a descriptive name for the menu).
    - **Description** - provides a short description of the menu.
    - **Type** - describes the menu's purpose as it relates to the OA Framework (we'll look at the valid options in detail below).
    - **Icon** - if this menu will be an item on a "Level 1" menu, provide the name of a seeded icon to assign to the menu. This icon displays when the profile FND: Top-Level Menu Display Mode is set to display the "Level 1" menu as icons and links.

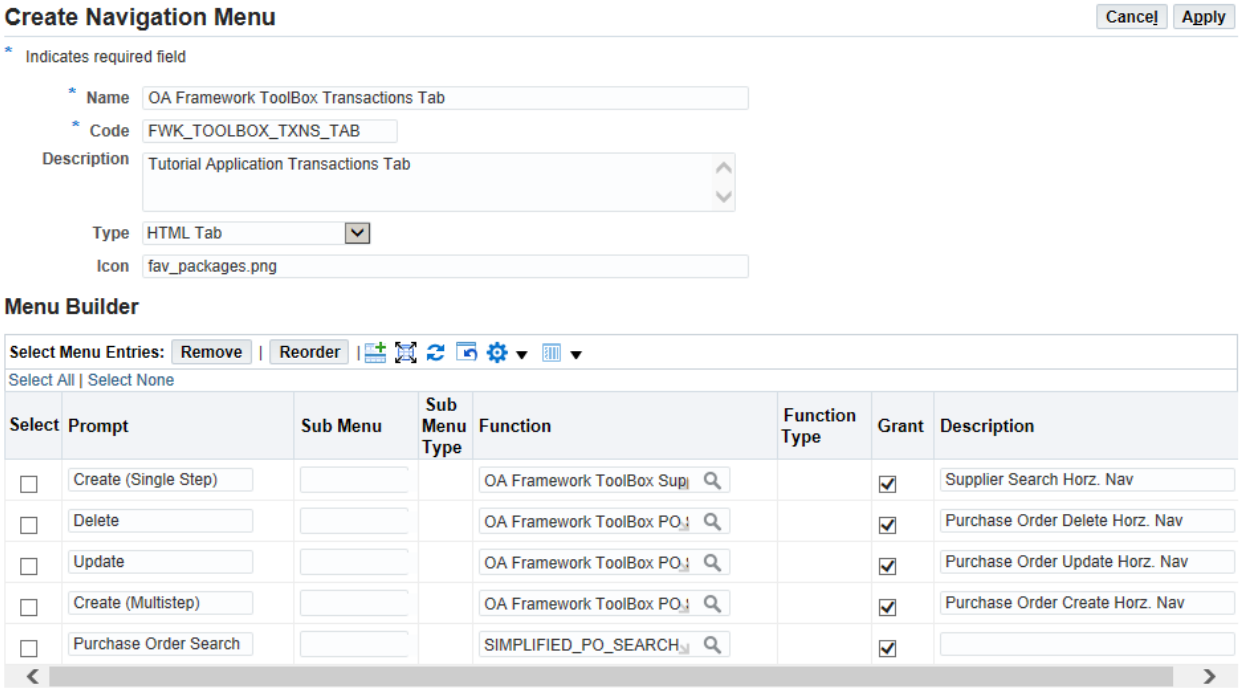
    Figure 7. Create Navigation Menu page.



6. For each menu that you create, you must add one or more menu entries to the **Menu Builder** table as shown below for the ToolBox Tutorial Application's "Transactions" tab menu definition. Select **Add Another Row** to add a menu entry that includes the following properties:

    - **Prompt** - this is the value that displays in the menu. For example, when we added the Lesson 5 function to the Lessons 4 - 7 tab menu in the ToolBox Tutorial Application, we entered "Lesson 5" as the function's prompt.
    - **Submenu** or **Function** - the user name of the submenu or function that you want to include at this point in the menu.
    - **Grant Flag** - the security rule that you want to associate with your function. Uncheck the grant flag. You will find more information on security in Chapter 4: Page Security.
    - **Description** - a brief description of the menu entry.

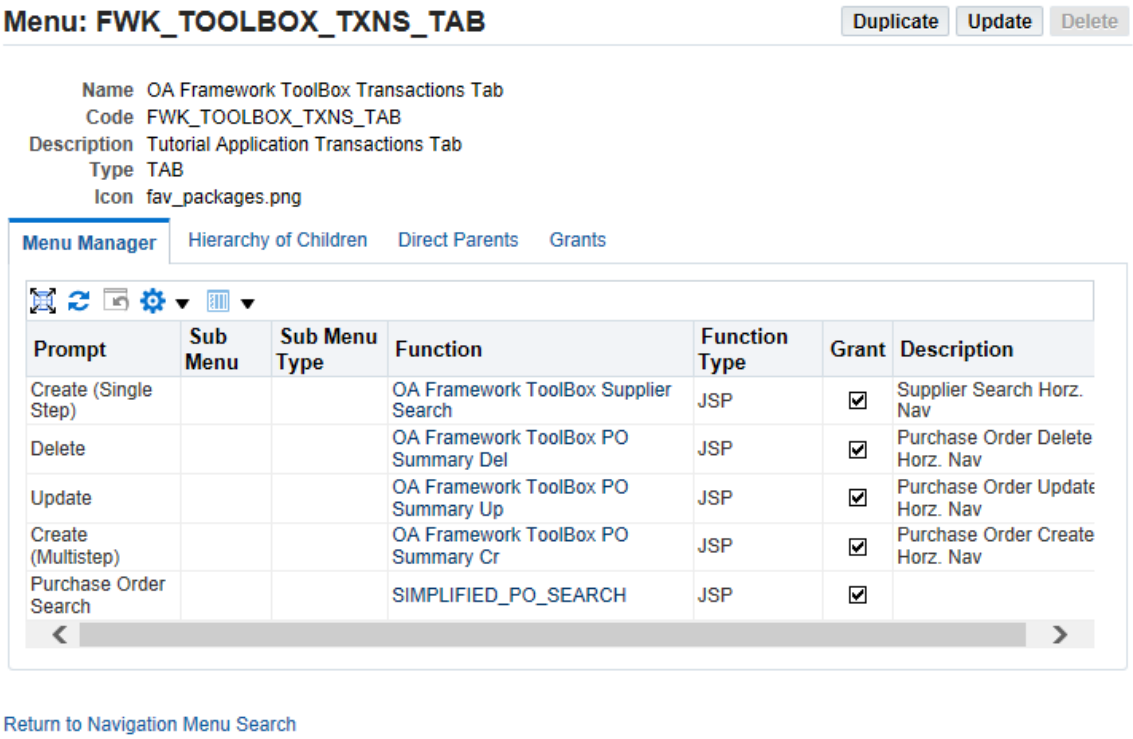
    Figure 8. Adding menus and functions to the "OA Framework ToolBox Transactions Tab" navigation menu using the Menu Builder table.



7. Select **Reorder** if necessary to indicate how the menu entries should be positioned on the menu relative to their peers. Menu entries are rendered from left to right in sequence, or from top to bottom (depending on the kind of menu being rendered).
8. Select **Apply** to save your work.
9. You can also query the **Navigation Menus** page for existing menus by using the Simple Search or Advanced search panel. In

the search results, select the link for the name of a menu to display the menu with its associated submenus and functions.

Figure 9: "OA Framework ToolBox Transactions Tab" menu showing associated functions with prompts in the Menu details page.



Implementing Specific Menu Types

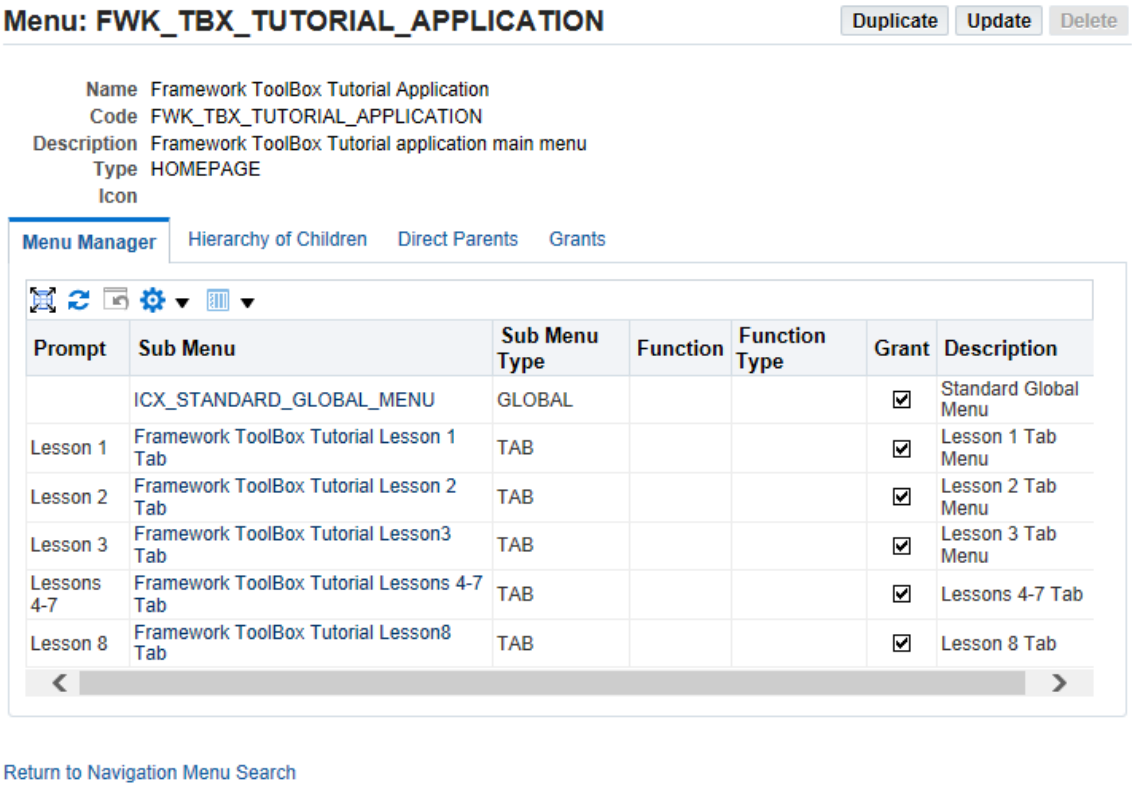
Now that you know how to create functions and menus, this section describes how to implement specific menu types. The Menu Examples section that follows shows how to combine these individual menu types into standard configurations.

Root OA Framework Application Menu

All OA Framework applications require a root menu of type "Home Page." You should add all top-level menu components (like global buttons and tabs) directly to your "Home Page" menu as illustrated for the OA Framework ToolBox Tutorial application's "Home Page" menu below.

Note: You must create a "Home Page" menu for your application regardless of whether or not it includes an actual "Home" page. Even if your application doesn't include any tabs or other menu components, you still must create this menu.

Figure 10: OA Framework ToolBox Tutorial "Home Page" menu.



Global Menu

See Buttons (Global) for instructions on adding standard global buttons like "Help" and "Preferences" and product-specific global buttons to your application.

Tabs

To include tabs in your application:

1. Create an Oracle E-Business Suite menu type of "HTML Tab" for each tab that you wish to display (the UI Guidelines state that you must have at least two tabs in an application if you have any; if you find yourself wanting to create a single tab, you should investigate a different menu design).
2. Add functions and submenus as appropriate:
    - If your tab displays a single page when selected (without any horizontal navigation or side navigation below it), you can simply add the corresponding function directly to the tab menu. In this case, you should not specify a **Prompt** value.
    - If your tab has horizontal or side navigation components, see the topics on creating these menu components below.
3. Add the "HTML Tab" menu as a submenu beneath your "Home Page" menu. Remember to specify a **Prompt** value; this is used as the tab's label text (see the prompts for the OA Framework ToolBox Tutorial lesson submenus above).

Note: If you add securing functions to your menu, remember to leave the **Prompt** field blank so they don't render as menu choices. Also, if you add multiple functions to an "HTML Tab" menu, the OA Framework always assumes the first one in sequence is associated with the menu entry. For example, if you define an "HTML Tab" menu and add the "Page A" function in sequence first and the "Page B" function in sequence second, the OA Framework will display the "Page A" function when you select the tab.

Horizontal Navigation

There are two ways to include horizontal navigation menu components in your application: you can simply add functions with a prompt beneath an "HTML Tab" menu, or you can create an "HTML Sub Tab" menu and add that to your "HTML Menu."

Note: Even though this menu type is called "HTML Sub Tab," it has nothing to do with the "Sub Tab" menu component as described in Menu Concepts above.

The following assumes that horizontal navigation components always render in relation to a tab; per the UI guidelines, they are not valid top-level menus.

If you want to add horizontal navigation components beneath a tab -- and none of the horizontal navigation components have any side navigation beneath them -- simply add the corresponding page functions directly to your "HTML Tab" menu while remembering to specify a **Prompt** value for the horizontal navigation label text. For example, Figure 4 above the OA Framework ToolBox Tutorial menu

definition for the "Lessons 4- 7" tab, which includes four horizontal navigation components (Lesson 4, Lesson 5, Lesson 6 and Lesson 7) Note the addition of securing functions without any prompts.

Alternatively, if you want to include horizontal navigation components with side navigation menus:

1. For each horizontal navigation component to include side navigation, create an Oracle E-Business Suite menu of type "HTML Sub Tab."

   Tip: You can combine "HTML Sub Tab," and plain functions as peers beneath a single "HTML Tab" menu. Each one with a prompt renders as a horizontal navigation menu entry.

2. Create and add a menu of type "HTML SideBar" to the "HTML Sub Tab" menu. Do not specify a **Prompt** value (see the Side Navigation section below for additional information about this).
3. Add the "HTML Sub Tab" submenu to your "HTML Tab" menu. Remember to specify a **Prompt** value to display as the horizontal navigation label text.

Note: If you add functions with prompts to your "HTML Sub Tab" menu -- instead of an "HTML Sidebar" menu -- the OA Framework will render them as side navigation links without any decoration (they render as plain links without bullets). You should avoid this approach, however, because it doesn't fully comply with the UI Guidelines (which expect that the links have bullets). It's also more robust to clearly define the menu hierarchy without skipping levels (we'll discuss this further in the Menu Examples section below).

Side Navigation

As described in the Horizontal Navigation section above, if you want to add a side navigation menu to your page, you should always create an "HTML Sidebar" menu for this purpose.

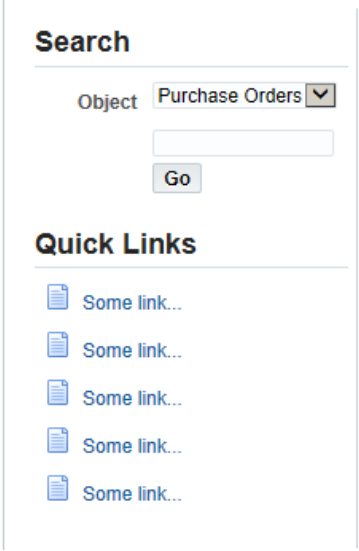
- If your side navigation includes a flat list of links, simply add a function for each like (with a prompt) to your "HTML Sidebar" menu.
- If your side navigation includes additional menu levels (4 ... N), create an "HTML SideList" menu for each level that you want to include and add functions and/or other "HTML SideList" menus as appropriate.

Note: Do not try to attach multiple "HTML Sidebar" menus to an "HTML Sub Tab" menu; the OA Framework (and the UI Guidelines) allow only one side navigation menu per horizontal menu entry. Furthermore, never add an "HTML Sidebar" menu directly to an "HTML Tab" or 'Home Page" menu.

All of that said, you can also create a side navigation programmatically to hold content that is unrelated to the menu (you cannot create a side navigation declaratively).

For example, the ToolBox Tutorial Home Page Lab illustrates how to create the following side navigation:

Figure 11: OA Framework ToolBox Tutorial Lesson 8 Home page side navigation.



In this example , we created the "Search" and "Some Link" regions declaratively, and then simply instantiated and added them to the side navigation programmatically. We then specified that our side navigation component should be displayed in the page's "start" layout area (a page includes three layout areas: start, end, and everything in the middle.

```
OASideNavBean sideNav =
  (OASideNavBean)createWebBean(pageContext, SIDE_NAV_BEAN, null, "hpSideNav");
```

```
    // We're using a header because we don't want any indentation before
    // the start of the fields. Because we're using a simple header, we
    // also need to add vertical spacing between the beans per the guidelines.

    OAHeaderBean search =
      (OAHeaderBean)createWebBean(pageContext,
                                  "/oracle/apps/fnd/framework/toolbox/tutorial/webui/HomeSearchRN",
                                  null, // ignore item for regions
                                  true); // using JRAD

    OAHeaderBean quickLinks =
     (OAHeaderBean)createWebBean(pageContext,
                                  "/oracle/apps/fnd/framework/toolbox/tutorial/webui/HomeLinksRN",
                                  null, // ignore item for regions
                                  true); // using JRAD

 sideNav.addIndexedChild(search);
 sideNav.addIndexedChild(quickLinks);

 String quickLinksText = pageContext.getMessage("AK", "FWK_TBX_T_QUICK_LINKS", null);
 String searchText = pageContext.getMessage("AK", "FWK_TBX_T_SEARCH", null);

 // Note that you don't have to do anything to get the correct CSS style
 // for the header text/line. This is handled for you automatically
 // by virtue of rendering a header bean on a dark background. The only
 // thing you do have to change is the header text size, which can't be
 // changes with a CSS class. Instead, use the setSize() method that all
 // header beans have (the default layouts are header beans).

 search.setSize(2); // 2 is the smallest header size
 search.setText(pageContext, searchText);

 quickLinks.setSize(2); // 2 is the smallest header size
 quickLinks.setText(pageContext, quickLinksText);

 // Note that you must call prepareForRendering() before setting your
 // side nav bean or it won't render.

 OAPageLayoutBean pageLayoutBean = pageContext.getPageLayoutBean();
 pageLayoutBean.prepareForRendering(pageContext);
 pageLayoutBean.setStart(sideNav);
```

Sub Tab

Unlike the menu components that we've discussed until this point, the Sub Tab is not implemented using Oracle E-Business Suite menus. Instead, the Sub Tab is simply a special container region in your page's contents.

See the Sub Tab Navigation document for additional information about implementing this component.

Task / Property Menu

Like the Sub Tab, a Task / Property menu is standard page content that is not implemented using Oracle E-Business Suite menus.

See the Bulleted List document for information that will help you implement this in-page menu.

Responsibility Menu

As described in Implementing the View, a responsibility is a group of (presumably) role-related tasks. A user is granted access to one or more responsibilities, and this association determines the tasks that he can perform.

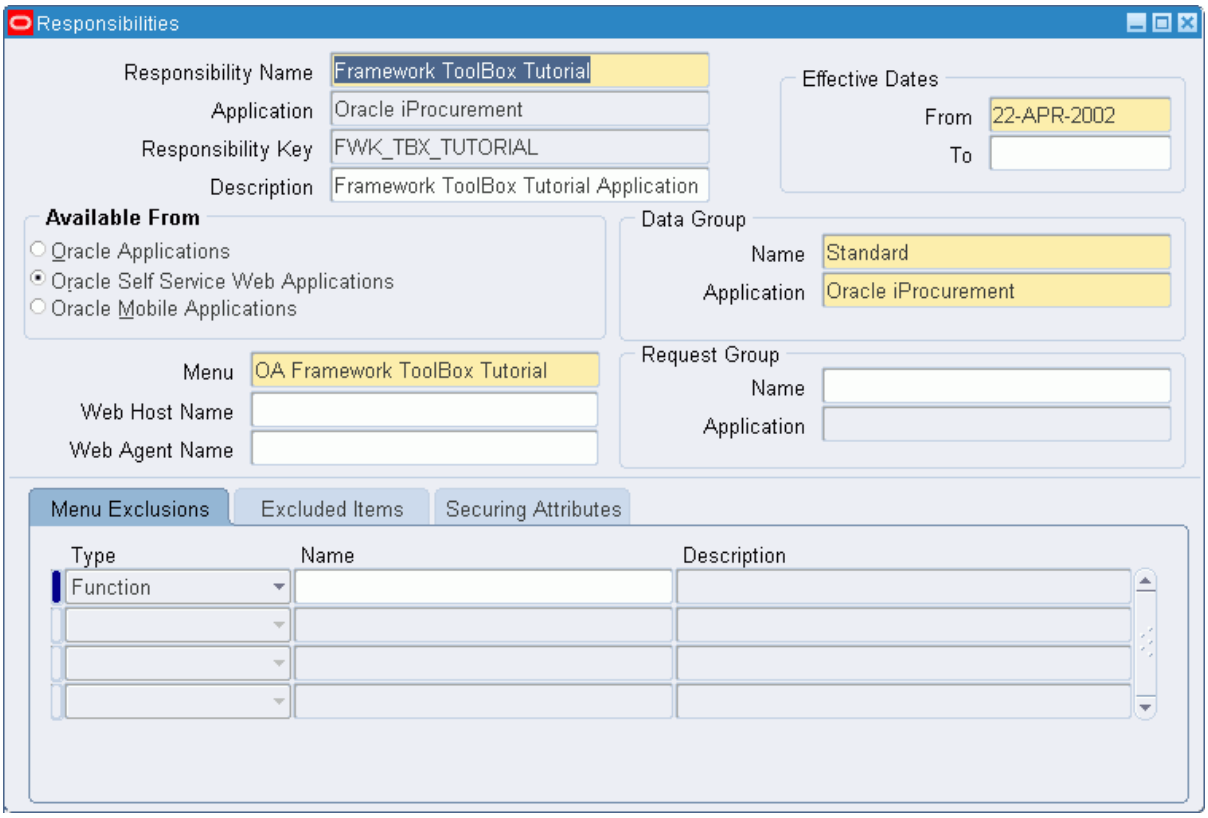
To enable access to your new "Home Page" menu:

1. Create a new menu of type "Standard," or open a preexisting menu of this type.
2. Add your "Home Page" submenu to the "Standard" menu, but do not specify a **Prompt** value. Note that you can associate multiple "Home Page" menus with a single responsibility menu.
3. Decide which page or pages you want to expose in the Oracle E-Business Home Page (for example, "Application X" might have 5 different tabs, but you only want to display a link to the "Application X Home" page in the PHP menu). Add these functions to your "Standard" menu.
4. Create a new responsibility as shown in Figure 8 below, and associate your selected "Standard" menu with it (each responsibility references one and only one menu).
5. Ensure that your user has access to the responsibility you created/opened in the previous step.

After you complete these steps, your responsibility root menu should contain your "Home Page" menus (with no prompts), and the functions (with prompts) you want a user to be able to select from in the Navigator.

Warning: It is essential that all the functions accessed from the Oracle E-Business Home Page include the OAHP and OASF parameters as described in the Menu Context section above. This enables the OA Framework to identify the "Home Page" menu and current function to select. Without these values, the menu will not render correctly.

Figure 12: OA Framework ToolBox Tutorial responsibility definition.



Summary: Rendering Rules

This section summarizes the key rules the OA Framework uses when rendering menus:

- Whenever the OA Framework shows a page, it always highlights the menu path to that page so you know visually where you are within the menu hierarchy (assuming the menu context has been established correctly).
  - It begins by rendering the selected function, and then traverses the selection path, rendering each node on the path and all of its siblings.
  - Note that the OA Framework assumes that all siblings (menus at the same level) are of the same menu type as the selected node.. For example, if you have a mix of tab and side navigation menus at the same level, and one of the tab menus is on the selection path, the OA Framework renders all of these siblings as tabs.
- Tabs and subtabs inherit the URL of the first accessible child leaf function within its menu hierarchy . If you have the following menu: **Tab X > Horizontal A > (Function1, Function2, Function3)**, the OA Framework automatically renders **Function1** when you select **Tab X** or **Horizontal A**.
- The OA Framework renders menus only if they include one or more accessible functions. If for example, your application's menu definition includes five tabs, but only four of them include accessible functions, the fifth will not render.

Pages Without Menus

If you want to create a page with no "chrome" (no tabs, global buttons and so forth -- typically required when you display content in a secondary browser window), you should define the page without a **pageLayout** region. In this case, simply create the page starting with a **stackLayout**, a **header** with a **messageComponentLayout**, or whatever other layout you need for your content. If the page has form elements, you must remember to add a form to your page. For example, a typical layout might look like:

```
stackLayout // top region for the page
|-- form
    | -- everything else
```

Without a **pageLayout** region, the OA Framework won't try to display any menu components, so you don't need to programmatically

hide them.

Menu Examples

OA Framework application menus are comprised of the following levels:

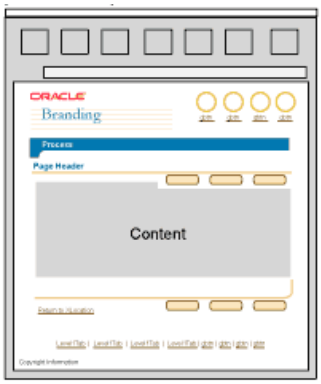
1. Tab
2. Horizontal Navigation
3. Side Navigation
4. Indented Side Navigation (as many levels as needed).

Now that you know how to create each of these menu types, you need to understand how to use them in relation to one another. The single most important rule that you should follow is this: even if your menu doesn't display a menu entry at a given level, you should not "skip" it in the structure. So, as you'll see in the individual examples below, even if an application includes only a side navigation (without any visible tabs or horizontal menu entries), the tab and horizontal menu entry must still exist in the menu structure.

No Tabs

As illustrated in Figure 13 below, this menu includes a single function that renders without any tabs. Although difficult to see in the conceptual diagram, note that the function's prompt displays in the blue bar below the branding.

Figure 13: Conceptual drawing of a page with no tabs or other menu components.



To achieve this effect in your application:

1. Create a function for your page.
2. Create a menu of type "Home Page."
3. Create a menu of type "HTML Tab" and add it to your "Home Page" menu without a prompt (so the tab doesn't display).

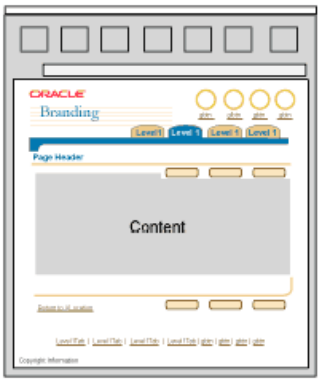
   Note you should not add functions directly to your "Home Page" menu. Remember the rule that you should not skip menu levels, even if they are not displayed.

4. Add your function to the "HTML Tab" menu. Remember to specify a **Prompt** value.
5. Add your "Home Page" menu to your responsibility menu.
6. Make sure your user has access to this responsibility before you try to test the menu.

Tab Bar Only

As illustrated in Figure 14 below, this menu includes four tabs. When the user selects each tab, a page displays. The tabs have no corresponding horizontal or side navigation components.

Figure 14: Conceptual drawing of a page with tabs only.



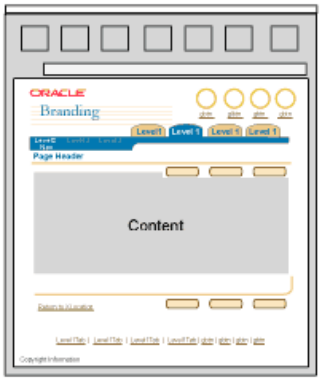
To achieve this effect in your application:

1. Create a function for each page that you want to display.
2. Create a menu of type "HTML Tab" for each tab you want to display.
3. Add each function to its corresponding tab menu. Do not specify any **Prompt** values.
4. Create a menu of type "Home Page."
5. Add the tab submenus to the "Home Page" menu.
    - Add the tab menus in your desired display sequence from left to right. The leftmost tab should be added first.
    - Remember to specify **Prompt** values for each tab submenu. These values display as the tab text.
6. Add your "Home Page" menu to your responsibility menu.
7. Make sure your user has access to this responsibility before you try to test the menu.

Tab Bar and Horizontal Navigation

As illustrated in Figure 15 below, this menu includes four tabs, one of which has a horizontal menu entry. When the user selects the tab, the horizontal menu entry's page automatically displays.

Figure 15: Conceptual drawing of a page with tabs and horizontal navigation.



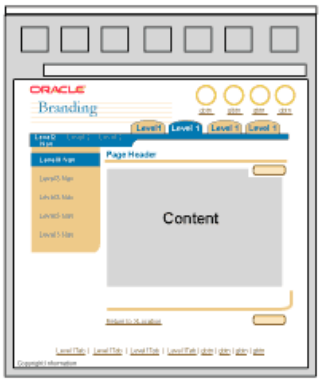
To achieve this effect in your application:

1. Create a function for each page that you want to display.
2. Create a menu of type "HTML Tab" for each tab you want to display.
3. Create a menu of type "HTML Sub Tab" for the horizontal menu entry you want to display in the second tab.
4. Add a function to the "HTML Sub Tab" menu. Do not specify a **Prompt** value.
5. Add the "HTML Sub Tab" submenu to the "HTML Tab" menu. Specify a **Prompt** value to display as the horizontal navigation text.
6. Create a menu of type "Home Page."
7. Add the tab submenus to the "Home Page" menu.
    - Add the tab menus in your desired display sequence from left to right. The leftmost tab should be added first.
    - Remember to specify **Prompt** values for each tab submenu. These values display as the tab text.
8. Add your "Home Page" menu to your responsibility menu.
9. Make sure your user has access to this responsibility before you try to test the menu.

Tab Bar, Horizontal Navigation and Side Navigation

Figure 16 illustrates a menu with four tabs, one of which has both horizontal navigation and side navigation menus. In this case (with only one horizontal menu entry), when a user selects either the tab or the horizontal navigation component, the first function in the side navigation displays.

Figure 16: Conceptual drawing of a page with tabs, horizontal navigation and side navigation.



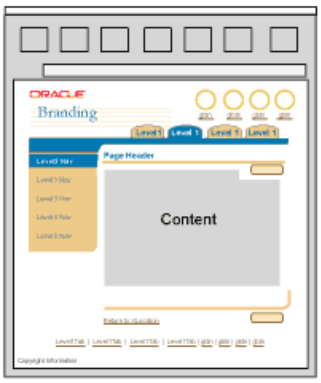
To achieve this effect in your application:

1. Create a function for each page that you want to display.
2. Create a menu of type "HTML Tab" for each tab you want to display.
3. Create a menu of type "HTML Sub Tab" for the horizontal navigation menu entry. Add it to the "HTML Tab" menu that should include the side navigation men. Specify a **Prompt** value.
4. Create menu of type "HTML Sidebar" and add it to the "HTML Sub Tab" menu without a prompt.
5. Add your page functions to the "HTML Sidebar" menu. Specify a **Prompt** value to display for each corresponding link.
6. Create a menu of type "Home Page."
7. Add the "HTML Tab" menus to the "Home Page" menu. Specify a **Prompt** value to display for each corresponding tab.
8. Add your "Home Page" menu to your responsibility menu.
9. Make sure your user has access to this responsibility before you try to test the menu.

Tab Bar and Side Navigation

As illustrated in Figure 17 below, this menu includes four tabs, one of which has a side navigation without a visible horizontal menu entry. When the user selects the tab, the first function in the side navigation menu displays.

Figure 17: Conceptual drawing of a page with tabs and side navigation.



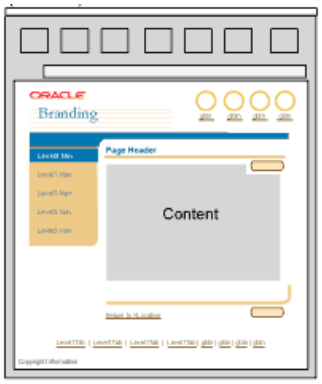
To achieve this effect in your application:

1. Create a function for each page that you want to display.
2. Create a menu of type "HTML Tab" for each tab you want to display.
3. Create a menu of type "HTML Sub Tab" for the invisible horizontal navigation. Add it to the "HTML Tab" menu that should include the side navigation menu.
4. Create menu of type "HTML Sidebar" and add it to the "HTML Sub Tab" menu without a prompt.
5. Add your page functions to the "HTML Sidebar" menu. Specify a **Prompt** value to display for each corresponding link.
6. Create a menu of type "Home Page."
7. Add the "HTML Tab" menus to the "Home Page" menu. Specify a **Prompt** value to display for each corresponding tab.
8. Add your "Home Page" menu to your responsibility menu.
9. Make sure your user has access to this responsibility before you try to test the menu.

Side Navigation Only

In this menu configuration, a side navigation displays without any tabs or horizontal navigation menu entries. The first function in the side navigation renders by default.

Figure 18: Conceptual drawing of a page with side navigation only.



To achieve this effect in your application:

1. Create a function for each page that you want to display.
2. Create a menu of type "HTML Tab" for the invisible tab.
3. Create a menu of type "HTML Sub Tab" for the invisible horizontal navigation. Add it to the "HTML Tab" menu without a prompt.
4. Create menu of type "HTML Sidebar" and add it to the "HTML Sub Tab" menu without a prompt.
5. Add your page functions to the "HTML Sidebar" menu. Specify a **Prompt** value to display for each corresponding link.
6. Create a menu of type "Home Page."
7. Add the "HTML Tab" menu to the "Home Page" menu without specifying a prompt.
8. Add your "Home Page" menu to your responsibility menu.
9. Make sure your user has access to this responsibility before you try to test the menu.

Runtime Control

Setting the Menu Context

In OA Framework applications, users can often navigate freely between related applications. For example, while working in a "Projects" module, a user might want to place a purchase order or initiate a buyer's auction -- two tasks that live within a "Procurement" module. When the user navigates from Projects to Procurement, the corresponding application menus should change to reflect the context switch. To implement this -- and other related behaviors -- simply specify the following URL parameters:

- `OAMC=R` removes the current menu context. You would use this when navigating to a page that should not display a menu, and there is no reason to preserve the menu context (for example, an access error page).
- `OAMC=K` keeps the current menu context. This is the default behavior if the OAMC is not present on the URL.
- `OAMC=N` keeps the current menu context, but without displaying a menu (so if the user selects a button in the page where you don't want the menu to be displayed, the original menu context is still intact). You would use this when navigating to a dialog page in certain situations.
- `OASF=<FunctionName>` Select the given function in the current menu context. If it does not exist move up the menu hierarchy to find a best match for a "Home Page" menu and this function as the selected function.
- `OAHP=<HomePageName>` Use only with OASF to change the menu context to the new "Home Page" and selected function. By setting both OAHP & OASF you can provide your starting point MenuContext, which is something you would commonly do if the user navigates from one application to another.

You can also make achieve the same behavior programmatically by setting appropriate values in one of the `oracle.apps.fnd.framework.webui.OAPageContext setForward*()` methods, or by calling one of the `OAPageContext resetMenuContext()` methods.

Rendering a "Level 1" Menu as a Set of Icons

To render a "Level 1" or top-level menu as icons in Release 12.2.5 and above, you need to perform the following steps.

Step 1. Identify the icons you wish to use.

1. All icons for the "Level 1" menu must reside in the $OA_MEDIA directory. Identify the seeded icons you wish to use from the $OA_MEDIA directory.

Step 2. Assign the seeded icons to the menus and functions of your top-level menu structure.

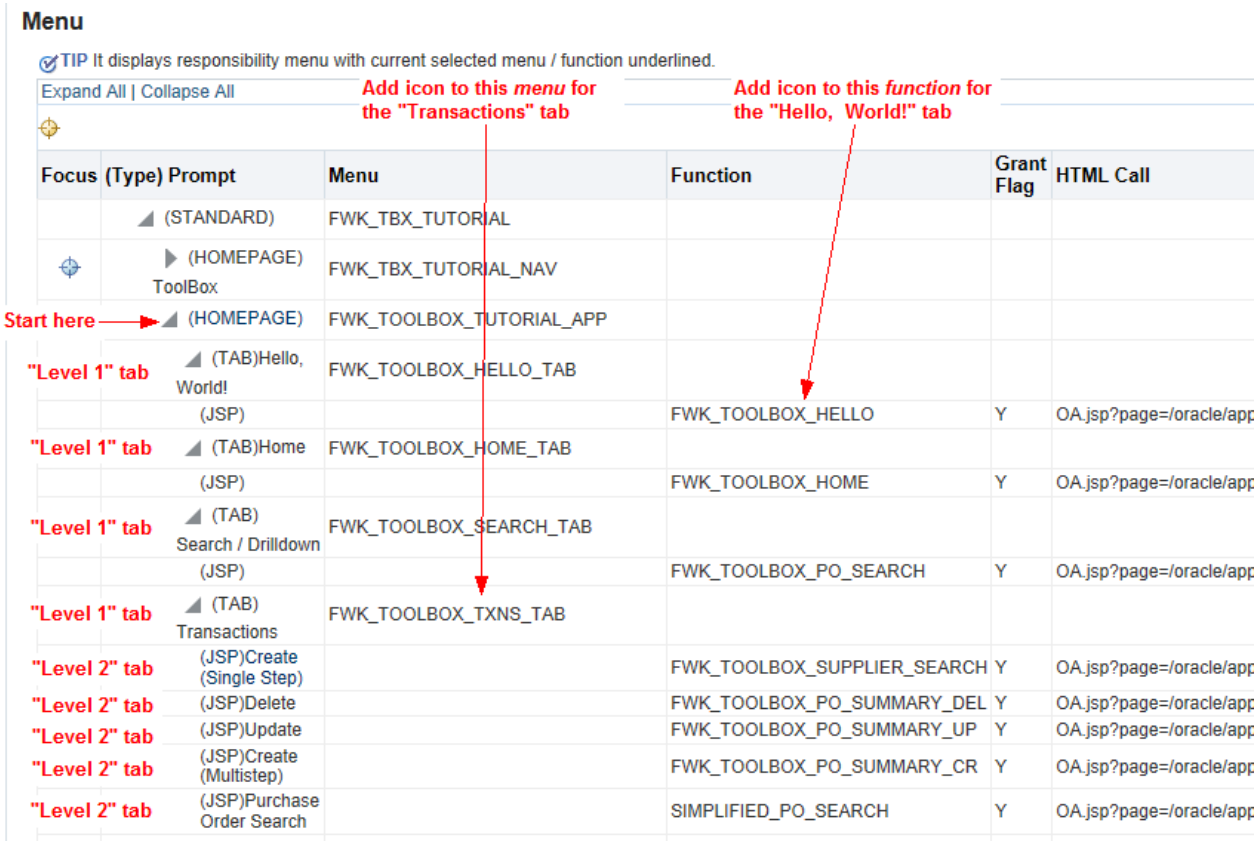
If you are creating a new menu structure, refer to the Declarative Implementation instructions to assign the seeded icons to your functions and menus.

If you are retroactively assigning seeded icons to an existing "Level 1" menu structure then follow these steps below.

1. **Assign a seeded icon to a menu.**
   A. Log in as a user with the "Functional Administrator" responsibility and select the "Functional Administrator" responsibility.
   B. Select **Core Services > Menus** to navigate to the **Navigation Menus** page
   C. In the **Navigation Menus** page, search for the existing menus that appear in "Level 1" of your menu structure and update the **Icon** field for each menu with a seeded icon filename.
2. **Assign a seeded icon to a function.**
   A. Log in as a user with the "Functional Administrator" responsibility and select the "Functional Administrator" responsibility.
   B. Select **Core Services > Functions** to navigate to the **Functions** page.
   C. In the **Function** page, search for the existing functions that appears in "Level 1" of your menu structure and update the **Icon** field of that function with a seeded icon filename.

Tip: To identify the menu and function developer keys or codes of your "Level 1" menu, navigate to any page in your responsibility and from that page, select the **About this page** link. In the resulting "About" page, select the Page Context Subtab and expand the menu structure to view the menu and function keys of your "Level 1" menu. Use these keys to search the **Menus** and **Function** forms or pages for the menus and functions that need to be assigned seeded icons.

Figure 19: "About" page view of the ToolBox menu's expanded menu structure.



Focus	(Type) Prompt	Menu	Function	Grant Flag	HTML Call
	(STANDARD)	FWK_TBX_TUTORIAL			
	(HOMEPAGE) ToolBox	FWK_TBX_TUTORIAL_NAV			
	(HOMEPAGE)	FWK_TOOLBOX_TUTORIAL_APP			
	(TAB)Hello, World!	FWK_TOOLBOX_HELLO_TAB			
	(JSP)		FWK_TOOLBOX_HELLO	Y	OA.jsp?page=/oracle/app
	(TAB)Home	FWK_TOOLBOX_HOME_TAB			
	(JSP)		FWK_TOOLBOX_HOME	Y	OA.jsp?page=/oracle/app
	(TAB) Search / Drilldown	FWK_TOOLBOX_SEARCH_TAB			
	(JSP)		FWK_TOOLBOX_PO_SEARCH	Y	OA.jsp?page=/oracle/app
	(TAB) Transactions	FWK_TOOLBOX_TXNS_TAB			
	(JSP)Create (Single Step)		FWK_TOOLBOX_SUPPLIER_SEARCH	Y	OA.jsp?page=/oracle/app
	(JSP)Delete		FWK_TOOLBOX_PO_SUMMARY_DEL	Y	OA.jsp?page=/oracle/app
	(JSP)Update		FWK_TOOLBOX_PO_SUMMARY_UP	Y	OA.jsp?page=/oracle/app
	(JSP)Create (Multistep)		FWK_TOOLBOX_PO_SUMMARY_CR	Y	OA.jsp?page=/oracle/app
	(JSP)Purchase Order Search		SIMPLIFIED_PO_SEARCH	Y	OA.jsp?page=/oracle/app

Tip: For complicated menu structures, especially menus that have the same prompt in multiple locations, it may be difficult to identify which functions or menus are in the "Level 1" menu. It may help to initially assign different icons to each different menu or function to see which icons appear.

A "Level 1" entry that has no icon specified will appear with a default icon. A "Level 1" entry that has an incorrectly-spelled or non-existent icon name specified will appear as an empty space or with the link name duplicated where the icon should appear.

Tip: If you wish to identify the list the menus and/or functions in a "Level 1" menu, you can search for the menus and functions that have images specified for its **Icon** field by using the Oracle Forms-based **Menus** and **Functions** forms under the "System Administrator" responsibility.

Step 3. Set the profile option FND: Top-Level Menu Display Mode to "Icons and Links" or "Icons and Links on Tablets Only".

Alternatively, you can update the **Top-level Menu Display Style** option from the **Preferences** page to **Icons and Links** or **Icons and Links on Tablets Only**. Setting the **Top-level Menu Display Style** option on the **Preferences** page is equivalent to setting the profile

option FND: Top-Level Menu Display Mode at the user level.

Accessibility Support

You may use the keyboard to navigate the top-level menu as follows:

- **[Tab]** - moves focus to the next page element. Only the active menu item of the "Level 1" menu is in the tab order. With the focus on an active menu in the "Level 1" menu, pressing **[Tab]** again shifts the focus to the next tab-able element in the content region. In the ToolBox Tutorial example shown in Figure 2, if the current focus is on the active menu **Transactions**, then pressing **[Tab]** once shifts the focus to the **Expand/Collapse** element beneath the top-level menu and pressing **[Tab]** again shifts the focus to the **Create (Single Step)** subtab in the "Level 2" menu.
- **[Home]** - with focus on an active menu, pressing **[Home]** shifts the focus to the first menu item of the "Level 1" menu and activates that menu.
- **[End]** - with focus on an active menu, pressing **[End]** shifts focus to the last menu item of the "Level 1" menu and activates that menu.
- **[Control]** + **[Page Up]** - with focus anywhere within a content region of a selected menu, pressing **[Control]** + **[Page Up]** shifts focus from the currently selected menu to the previous menu and activates it; if the currently selected menu is the first menu item in the top-level menu, then focus shifts to the last menu item in the top-level menu and activates it.
- **[Control]** + **[Page Down]** - with focus anywhere within a content region of a selected menu, pressing **[Control]** + **[Page Down]** shifts focus from the currently selected menu to the next menu and activates it; if the currently selected menu is the last item in the top-level menu, then focus shifts to the first menu item in the top-level menu and activates it.
- **[Left Arrow]**/**[Right Arrow]** - with focus on the active menu, pressing **[Left Arrow]**/**[Right Arrow]** shifts focus to the previous/next (or next/previous in a right-to-left session) menu item in the "Level 1" menu and activates it. If the currently selected menu is the last menu item in the top-level menu, pressing **[Right Arrow]** (or **[Left Arrow]** for a right-to-left session) shifts focus to first menu and vice versa.
- **[Control]** + **[Up Arrow]** - with focus anywhere within a content region, pressing **[Control]** + **[Up Arrow]** shifts focus to the currently selected menu.

Personalization Considerations

- None

Known Issues

- None

Related Information

- BLAF UI Guidelines
  - Tabs/Navigation
- Developer's Guide
  - Buttons (Global)
  - Bulleted List
- Javadoc
- OA Framework ToolBox Tutorial / Sample Library

Tree

Overview

As described in the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Tree specification, the Tree control allows users to quickly browse through complex sets of hierarchical objects and displays the relationship between the sets of objects in a hierarchy. You generally implement a Tree for the purpose of allowing users to select objects from a hierarchy or to display the Master/Detail information for a hierarchy of objects.

Contents

- Tree Description
- Defining Business Components
- Defining a Tree Component
  - Declarative Implementation
  - Runtime Control
- Defining a 3-Frame JSP Page for Use with the Tree Component
  - Declarative Implementation
  - Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

Tree Description

The Tree control shares many similarities with the HGrid feature, which displays objects in a hierarchy, but in a tabular format. A Tree is generally used when you want to put emphasis on the hierarchy and the relationship between different sets of objects in a hierarchy. A HGrid is more appropriate when you want to display the hierarchy, but also give more detailed information at each node of the hierarchy. Consider using a Tree control instead of a HGrid if you want your users to either:

- Select an object from the hierarchy, and then continue with a task that does not require display of the hierarchy.
- Display hierarchical Master/Detail content, and select an object in the hierarchy to display object details on the same page (on the right).

A Tree consists of a set of nodes structured in a parent-child hierarchy. Each node consists of three display attributes: text, image, and a URL (rendered as a link). The top level of the Tree is called the set of root nodes. Each node in a Tree can contain the following object:

- Parent Object - a container that also has object details. It may contain other parent objects, child objects, or just containers only. It appears as link text and has an associated hide/show icon that allows you to expand or collapse the tree at that node.
- Container Only - a container (only) that does not have object details. It may contain parent objects, child objects, or other containers (only). It appears as plain text, and has an associated hide/show icon that allows you to expand or collapse the tree at that node.
- Child Object - an object that is not a container, but has object details. It may not contain anything. It appears as link text, but does not have an associated hide/show icon. A node containing a child object is also referred to as a leaf node.

Following is an example of a Tree control containing objects within folder containers that do not have object details.

Figure 1: A Tree component containing objects within folder containers that do not have object details.



Partial Page Rendering of a Tree

The Tree web bean now supports self mode partial page rendering. Whenever a user expands or collapses a node, the Tree web bean adds itsef as a partial target.

Defining Business Components

As with all other beans supported by the OA Framework, the data for the Tree component is derived from BC4J objects. Just as the Tree displays data in hierarchical format, and the structure of the source data is also hierarchical, based on multiple view objects connected via view links. Each instance of a view object is connected to the Tree at a particular level, allowing the Tree to display all the rows in range at that level. The view links that define the parent-child relationship between a row in the master view object and the detail view object, allow you to drill down in the Tree and show details at the next level. When a user selects the hide/show icon to show more details, OA Framework traverses the view link and fetches/displays all the detail records for the master row that is selected. When you define a Tree in OA Extension, you specify the name of the view link instance for each node that is used to drill down to the correct detail view object instance.

Note It is possible to use a different view object for the children rows as long as the parent view object and the child view object share the same attribute names. The reason for this becomes clear when setting up the OA Extension metadata. An unlimited number of view links can be used to define the necessary parent-child relationships at each level of the hierarchy.

Each row in a view object provides data for a corresponding node in the Tree at each level. An initial instance (top most) of the view object should return the root node of the Tree. Note that this top level view object should return *exactly* one row. The Tree component relies heavily on the notion of hierarchical data with a unique root. If the top level view object returns multiple rows of data, those rows are automatically made children of a dummy root node. The automatically generated parent dummy node renders as non-selectable and expanded.

Note You cannot get a handle to this dummy node as it is generated internally and does not map to any row or view object attached to the Tree.

The first step in defining a Tree is to define the business object hierarchy that map to your business requirements.

To illustrate the above, you can build a simple Tree example to display supervisor-employee hierarchy information. (Note that some of the data model is greatly simplified for this example.) The data for each employee comes from the `PER_ALL_PEOPLE_F view`. Each employee is uniquely identified by the `PERSON_ID` column in this view. The `PER_ALL_ASSIGNMENTS_F` view describes the supervisor-employee relationship through the `SUPERVISOR_ID` and `PERSON_ID` columns in this view.

Step 1: Set up a view object definition for the `PER_ALL_PEOPLE_F` view, selecting the data that you want to display in the Tree. You can download `oracle.apps.fnd.framework.persontree.server.PerAllPeopleFVO` as an example. You can also download the corresponding VOImpl class. Note that the `initQuery` method in the VOImpl adds an additional where clause to the view object to fetch the root node.

Step 2: Define the view link used to retrieve subsequent levels of the Tree. In this example, define a view link that links the `PerAllPeopleFVO` to itself.

a. In JDeveloper, select the package to which you want to add the view link. Right click and choose the "Create View Link ..." option to bring up the "View Link Wizard".

b. In the View Link Wizard, **Step 1 of 6: Name**, enter a name for the view link (`PerAllPeopleFVL` in this example).

c. In **Step 2 of 6: View Objects**, choose the source and destination view objects. In this example, `PerAllPeopleFVO` is used as both the source and destination.

d. In **Step 3 of 6: Source Attributes**, select the source attributes. These typically are the primary key attributes (or a subset thereof) of the source view object. You may also want to select other columns that are needed to build the where clause used to fetch the detail row set. The values of these attributes from the master view object are used to determine the detail row set. For this example, use the `PERSON_ID` column as discussed earlier.

e. In **Step 4 of 6: Destination Attributes**, select the destination attributes (as in the previous step).

f. In **Step 5 of 6: View Link SQL**, build the where condition used to get the detail row set. The default where clause simply contains a one-to-one mapping of the source and destination attributes. The bind variables are bound with values of the source attributes in the master row. In this example, use the `PER_ALL_ASSIGNMENTS_F` view to determine all the persons supervised by the person in the master row. In other words, construct a where clause as follows:

```
person_id in (select person_id from per_all_assignments_f where supervisor_id = :1)
```

g. In **Step 6 of 6: View Link Properties**, ensure that the **Generate Accessor in View Object** checkbox is checked for both the Source and Destination view objects. The accessor name is generated automatically but you can change it if desired.

h. You can download the complete definition for PerAllPeopleFVL. You can similarly setup additional view links if the master-detail relationships at each level of the Tree are different.

Step 3: Add the view objects and view links you created to the application module used for the page. Note that adding the view link to the application module using the Application Module Wizard can be tricky. First

add the view objects to the application module. Then to add a view link, select the view link in the left column and select the source view object in the right column. This enables the ">" shuttle control and you can move the view link over to the right.

Defining a Tree Component

Having defined the data sources for the Tree bean, the next step is to define the Tree component in OA Extension. A Tree component can be used in one of two ways, according to the BLAF guidelines:

- To select objects within the hierarchy view (as shown in Figure 1 and in the BLAF guidelines. Instructions to implement this usage are described in the following Declarative Implementation and Runtime Control sections.
- To display the master/detail content for an object hierarchy using frames (as shown in Figure 2 and in the BLAF guidelines for display usage. Instructions to implement this usage are described in the Defining a 3-Frame JSP Page section.

  Note To implement the BLAF guideline for a no-frame page that displays a tree structure in the Side Navigation area and the details of the selected object in the page content area to the right, you should refer to Chapter 4: Tabs/Navigation.

Declarative Implementation

To create a Tree component for the purpose of allowing a user to view sets of records and their relation to one another, you define a region that contains a region item of style **tree**. This Tree Level region item maps to the root of the Tree Bean. The Tree Level region has two nested region items: a Tree Definition and a Tree Child. The Tree Definition region item describes the node of the tree and the Tree Child region item points to a Tree Level nested region. This allows OA Framework to build complex hierarchies in a declarative way.

The definition of a Tree bean is created by specifying the following metadata in OA Extension:

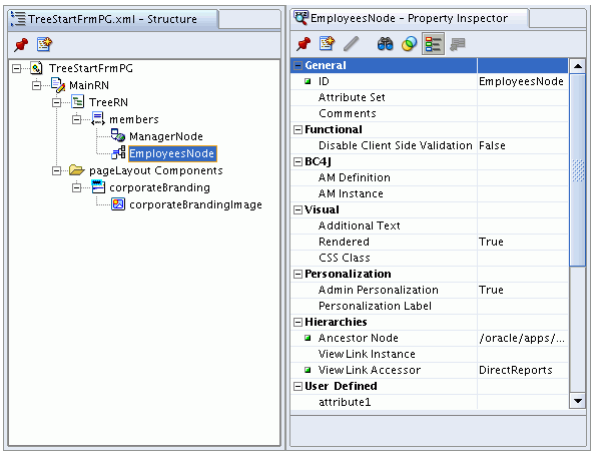
Step 1: Define a top level region and set the `Region Style` property to **tree**. You may specify an optional controller and application module for the tree region, by setting the `Controller Class` and `AM Definition` properties, respectively. You can nest the Tree region within a container region such as Page Layout, Header, Stack Layout, or messageComponentLayout.

Step 2: The Tree region defines the root level in the tree hierarchy. The Tree region can have two types of named children (members):

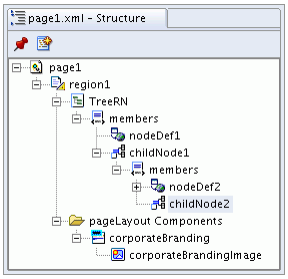
- **nodeDefinition** - The nodeDefinition item automatically appears when you set the Region Style to tree. It defines the appearance of the node in the hierarchy. For the nodeDefinition item, specify:
  - a value for the `View Instance` property to associate the node with a view instance.
  - a value for the `View Attribute` property, to render the view attribute name as the text of the node.
  - a value for the `Icon URI` property if you want to render an image next to the text in the node.
  - a value for the `Destination URI` property to render the node as a hyperlink.
- **childNode** - In the Structure pane, select **members** under the Tree region and display the context menu. Under **New**, select **childNode**. The childNode item holds the view link, which defines the parent-child relationship between tree levels.

  - To achieve a loop in your tree, such as a Manager to Employee recursive relationship, set the `Ancestor Node` property on the childNode item to indicate the region to which you want to loop back. The `Ancestor Node` property can be set to another tree region, to the same tree region (for a recursive relationship), or to no tree region (null, indicating that the node is a leaf level in the hierarchy tree). The ancestor node should be set as a fully-qualified path name such as /oracle/apps/*<applshortname>*/*<module>*/*<pagename>*.*<ancestor region name>* where the ancestor region is whatever region (node)
    you are looping back to.

    Attention: For a recursive relationship, as indicated above, you set the ancestor node to the same tree region. However, the "same tree region" refers to the **parent** of the base recursing node and not the recursing node itself. See the Sample Library for an implementation of a tree region with a recursive relationship, as shown in the figure below.



  - If you need to achieve multiple levels of nodes in your tree, such that the levels do not loop back to an ancestor, you can do so by selecting the last childNode item in the Structure pane, and from the context menu, choose **New** > **Members** to nest another node level until you achieve the multi-level tree structure you need. Note that if you do not require your tree to loop back to some parent node, you do not need to set the `Ancestor Node` property on the childNode items.



  - Set the `View Link Accessor` property on each childNode item to the view link accessor name that should be used to retrieve the child node at that level.

    Note: Previously, the view link instance used to retrieve the child nodes at a particular level was set via the child node's View Link Instance property. This property is now deprecated and is present only for backwards compatibility. You should only use the View Link Accessor property on the child node to specify the view link instance.

Runtime Control

The last step in defining a Tree is to setup a controller. Refer to the sample controller class `PersonTreePageCO` to continue with the person tree example discussed in the Defining Business Components section.

processRequest method

As is the case with other components in OA Framework, use the `processRequest` method for any custom layout code. Initialize the view object for the root node in this method.

Note You **must** execute the query on the root view object. In the earlier original implementation of the Tree, the Tree used to automatically execute the query for the root view object. To ensure backward compatibility, this behavior is still present in earlier versions of OA Framework, however, moving forward, you should consider this behavior deprecated.

processFormRequest method

You can use the `processFormRequest` method to process form events from the page containing the Tree. From a Tree, you should access data only via the business components. Please look at the PersonTreePageCO for code that walks the business component tree. Refer to the `oracle.apps.fnd.framework.webui.OATreeQueriedRowEnumerator` Javadoc for additional information.

Setting Initial Focus for a Tree with Many Nodes

When a page with a tree loads and the number of nodes in the tree is large, a user may find it cumbersome to navigate to a specific node. Now you can indicate the tree node that OA Framework should set initial focus on as soon as the page loads in the browser. Each master and child node in the tree is associated with a BC4J row and these rows have primary keys associated with them. OA Framework takes this into consideration and creates a unique ID for each row of the tree. You then simply indicate on which row to set initial focus.

For example, suppose a page shows Project Task Information in the form of a tree and there is a search page prior to this that allows users to search for a specific project task. In this case, you need to first call the following API on `OACommonUtils` in the `processRequest` method of the tree page to identify the row on which you want to set initial focus:

```
Row row = /* Logic to find the row that needs to be focused - Left to flexibility of the product team.*/
int hashCode = OACommonUtils.getRowKeyHashCode(row);
```

Then you call this `OADefaultTreeBean` API to assign the unique ID for the row:

```
String uniqueNodeID = treeBean.getNodeId(hashCode);
```

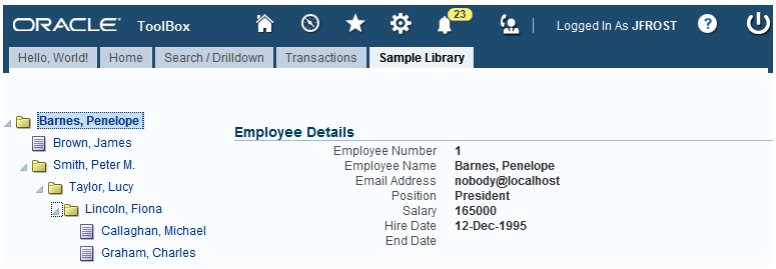
Lastly, you call this API on `OABodyBean` to set the initial focus on that unique ID:

```
((OABodyBean)pageContext.getRootWebBean).setInitialFocusID(uniqueNodeID);
```

Defining a 3-Frame JSP Page for Use with the Tree Component

To use a tree component to display hierarchical master/detail content in a 3 frame page as per the BLAF guidelines, you must call OAFrame.jsp, which renders three HTML frames (herein referred as Top, Start and Center) within a frameset.

Figure 2: Three Frame Example of a Master/Detail Tree



The BLAF guidelines currently certifies the 3-frame page template for the **following use only**, in other words, this is the ONLY frame usage allowed in BLAF applications:

- The Top frame renders the branding image, global icons and/or menu structure. No application content should be displayed in this frame.
- The Start frame renders the summary information in a hierarchical view (in the form of tree). This frame should not contain the page header
- The Center frame renders the content with just the page footer and no header. Selecting a link in the Start frame (a tree node) refreshes the content in the 'Center' frame.

By default, OA Framework currently uses 20% of the size for the width of the Start frame and the remaining for the Center frame. It also defaults the height of the top frame to 175 pixels. You have the option of overriding these default dimensions when you invoke OAFrame.jsp to render the frameset.

Declarative Implementation

Step 1: Create the source(content) for the Top frame. Since the top frame can only render header information such as a branding image, global icons and/or a menu structure, refer to the following topics in Chapter 4 for additional information: Branding, Buttons(Global), and Tabs/Navigation.

Step 2: Create a new page in OA Extension to render the header information specified in Step 1. Be sure to define a pageLayout region in this page.

Step 3: Create the source for the Start frame. The Start frame renders the summary information using a Tree web bean. This frame does not contain a page header.

- In the Structure page, select the page you created in Step 2, and choose **New** > **Region** from the context menu.
- Set the `Region Style` for this new region to a **pageLayout**. Although this region displays a header, it is needed to ensure proper layout and to render the FORM HTML tag, which is needed for the tree to work properly.
- Under the page layout region, choose **New** > **Region** from the context menu. Set the `Region Style` for this new region to **tree**.
- Define the members for the tree region.
- For each tree members' nodeDefinition, enter a URI value for the `Destination URI` property and append to the URI, the value **OAPLRS=PLNH**. This parameter ensures that the page header information does not render in the Start or Center frames. For example:

  ```
  OA.jsp?page=/oracle/apps/fnd/framework/toolbox/tutorial/webui/SupplierDetailsPG&supplierName={@Name}&supplierOnHold={@OnHoldFlag}&supplierId={@Suppli
  ```

- For each tree members' nodeDefinition, enter **OACFr** as the value for the `Target Frame` property. This sets the Center frame as the frame to refresh when a user selects a node in the rendered tree hierarchy.

Step 4: Create the source for the Center frame. The Center frame renders the page content of the selected tree node, with the page footer.

- Define a page in OA Extension with the content required. Be sure to define a pageLayout region in this page so that the page footer can be rendered. Note that the same page can be used for both the Top and Center frame.

Step 5: Create separate form functions for each frame's source. A function is a token that is registered under a unique name for the purpose of assigning it to, or excluding it from, a responsibility. The names of your functions are later passed on the URL using the **OAFUNC** parameter.

Attention: For the form functions that you define for the Top and Center frames' source, the HTML Call that you specify in the Web HTML region of the Oracle E-Business Suite Form Functions screen must include the OAHP and OASF parameters to ensure proper menu rendering. See the Menus Concept section in Chapter 4: Tabs/Navigation information about these parameters.

Step 6: Invoke **OAFrame.jsp** to render the frameset. OAFrame.JSP expects all three function names as a single parameter string delimited by a colon:

```
OAFrame.jsp?OAFunc=<OATFRAME>:<OASFRAME>:<OACFRAME>
```

where:

- *`OATFRAME`* can be replaced with the name of your function used to render the Top frame.
- *`OASFRAME`* can be replaced with the name of your function used to render the Start frame.
- *`OACFRAME`* can be replaced with the name of your function used to render the Center frame.

You may also add the `OAFRDIM` parameter to the **OAFrame.jsp** to override the default dimensions of the frames:

```
OAFrame.jsp?OAFunc=<OATFRAME>:<OASFRAME>:<OACFRAME>&OAFRDIM=<width:height>
```

where:

- `OAFRDIM` is the name of the parameter used to override the default Start and Top frame dimensions. The *`width`* and *`height`* values are delimited with a semicolon (:).
  - *`width`* - a value in pixels or percentage that determines the width of the Start frame. Specify the letter 'p' at the end of the value to denote a percentage value.
  - *`height`* - a value in pixels or percentage that determines the height of the Top frame. Specify the letter 'p' at the end of the value to denote a percentage value.

Example 1

Specify the width of the Start frame as 50 pixels and the height of the Top frame as 15 pixels:

```
OAFrame.jsp?OAFunc=<OATFRAME>:<OASFRAME>:<OACFRAME>&OAFRDIM=50:15
```

Example 2

Specify the width of the Start frame as 50% of the size and the height of the Top frame as 15% of the size:

```
OAFrame.jsp?OAFunc=<OATFRAME>:<OASFRAME>:<OACFRAME>&OAFRDIM=50p:15p
```

Runtime Control

The `Target Frame` property can be programmatically set. Here is a code example of how to set the targetFrame attribute from the controller of the tree:

```
...
// supplierNodeDefRN is the member node of the Tree region.
OAWebBean node = webBean.findChildRecursive("supplierNodeDefRN");

if (node != null)
  {node.setAttributeValue(TARGET_FRAME_ATTR,
     OAWebBeanConstants.CENTERFRAME_NAME);
  }
...
```

Note `OAWebBeanConstants.CENTERFRAME_NAME` is a public constant that you can use. Below are other public constants:

- `TOPFRAME_NAME`
- `STARTFRAME_NAME`
- `OA_PAGELAYOUT_RENDER_STYLE` - Parameter name to specify the page layout render style. Possible values are `PAGELAYOUT_HEADER_ONLY` and `PAGELAYOUT_NO_HEADER`. You can use this when setting the Destination URI attribute programmatically.
- `PAGELAYOUT_HEADER_ONLY`
- `PAGELAYOUT_NO_HEADER`

Menu Tab Content Display

If a user selects a menu tab in the Top frame, you must programmatically get a handle on the menu link, as well as programmatically control how that new page content from the selected tab displays. You can display the page content in one of two ways:

- In the Center frame, such that the Start frame still displays the tree structure.
- In the Top frame, such that the content is displayed in the entire window and the Start and Center frames are both eliminated.

Displaying Content in the Center Frame

To display the new menu content in the Center frame, you need to remove the header shown in the Center frame by passing the parameter `OA_PAGELAYOUT_RENDER_STYLE` in controller code for the Top frame (for the example shown in the Sample Library, the Top frame controller is TreeDetailsCO):

```
OAPageLayoutBean pageLayout =
  pageContext.getPageLayoutBean();
pageLayout.prepareForRendering(pageContext);

UINode tabBar = pageLayout.getTabs();

int childCount = 0;
if( tabBar != null )
{
 childCount =
   tabBar.getIndexedChildCount(
     pageContext.getRenderingContext());
}
for( int i=0; i<childCount; i++)
{
 LinkBean child = (LinkBean)tabBar.getIndexedChild(
   pageContext.getRenderingContext(), i);
 if( child != null )
 {
  child.setTargetFrame(CENTERFRAME_NAME);
 }

 // Need to add the OA_PAGELAYOUT_RENDER_STYLE with the
 // value PAGELAYOUT_NO_HEADER to remove the header rendering.
 // This parameter is exposed as a public constant in
 // OAWebBeanConstants.

 if( child.getDestination() != null )
 {
  child.setDestination((new StringBuffer
    (child.getDestination()).append("&").
    append(OA_PAGELAYOUT_RENDER_STYLE).
    append("=").append(PAGELAYOUT_NO_HEADER).toString()));
 }
}
...
```

Displaying Content in the Top Frame

To display the new menu content in the Top frame, so that the content takes over the entire browser window, you need to add the following to your tree controller code:

```
OAPageLayoutBean pageLayout =
  pageContext.getPageLayoutBean();
pageLayout.prepareForRendering(pageContext);

UINode tabBar = pageLayout.getTabs();

int childCount = 0;
if( tabBar != null )
{
 childCount =
   tabBar.getIndexedChildCount(
     pageContext.getRenderingContext());
}
for( int i=0; i<childCount; i++)
{
 LinkBean child = (LinkBean)tabBar.getIndexedChild(
   pageContext.getRenderingContext(), i);
 if( child != null )
 {
  // Header needs to be displayed here.
  child.setTargetFrame("_top");
 }
}
```

Personalization Considerations

- See a summary of Tree personalization considerations in the Oracle Application Framework Personalization Guide.

Known Issues

- See a summary of key tree issues with suggested workarounds if available

Related Information

- BLAF UI Guideline(s)
  - Tree
- Javadoc File(s)
  - `oracle.cabo.ui.beans.nav.TreeBean`
  - `oracle.apps.fnd.framework.webui.beans.nav.OATreeBean`
  - `oracle.apps.fnd.framework.webui.OATreeQueriedRowEnumerator`
  - `oracle.apps.fnd.framework.webui.beans.nav.OADefaultTreeBean`
  - `oracle.apps.fnd.framework.webui.OAWebBeanConstants`
  - `oracle.apps.fnd.framework.webui.beans.table.OAHGridBean`
- Lesson(s)
- ToolBox Tutorial / Sample Library
  - To view the Tree (in 3-Frame JSP) example implemented in the Sample Library project of the Toolbox workspace, run the `toolbox.jws` -> `SampleLibrary.jpr` -> `SampleBrowserPG.xml` file in OA Extension. Select the *Tree (in 3-Frame JSP)* link in the *SampleBrowserPG* page that displays in your browser to view the implementation.
  - The following package files in the Sample Library display the declarative and runtime implementation of this Tree (in 3-Frame JSP) example:
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.SampleBrowserPG.xml`
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.TreeTopFrmPG.xml`
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.TreeStartFrmPG.xml`
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.TreeCenterFrmPG.xml`
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.TreeDetailsCO.java`
    - `oracle.apps.fnd.framework.toolbox.samplelib.webui.TreeStartFrameCO.java`
- Sample Code
  - PerAllPeopleFVOImpl
  - PersonTreePageCO
  - PerAllPeopleFVL
  - PerAllPeopleFVO

Chapter 5: Implementing Server-Side Features

Java Entity Objects

Overview

- About Entity Objects
- Create
- Update / Validate
- Delete
- Lock
- Commit
- Rollback
- Object Version Number Column
- Standard WHO Columns
- Error Handling
- Entity Experts, Validation View Objects and Validation Application Modules
- Calling PL/SQL Functions and Procedures
- Entity Objects for Translatable (_TL) Tables
- Standard Validation Patterns / Examples

Prerequisite Reading

- Implementing the Model

Related Information

- Chapter 6: Advanced Model Development Topics
- Chapter 8: Applications Java Coding Standards
- Chapter 8: OA Framework File / Package / Directory Structure Standards
- Chapter 8: OA Framework Model Coding Standards

About Entity Objects

Entity objects encapsulate business logic and DML operations for application tables.

Object Model and Key Classes

- `oracle.apps.fnd.framework.server.OAEntityCache`: This cache is used to store queried rows for a particular entity. Multiple view objects that are mapped to the same entity share the same entity cache.
- `<YourEntityName>EOImpl` extends `oracle.apps.fnd.framework.server.OAEntityImpl`: This is the entity object itself. When instantiated, it represents one row of data.
- `oracle.apps.fnd.framework.server.OAEntityDefImpl`: Represents the metadata describing the entity object, including attributes, events, validators, associations, and properties. When instantiated, it describes all instances of the entity object class. The entity definition class is a singleton.
- `<YourEntityName>Expert` extends `oracle.apps.fnd.framework.server.OAEntityExpert`: This is a special singleton helper class that is registered with an entity object.
- `oracle.jbo.Key`: This is an immutable primary, foreign, or composite row identifier.

Create

To create an entity object, you must call `createRow` and then `insertRow` on the corresponding view object as illustrated below.

```
// In the application module; this example from the OA Framework
// ToolBox Tutorial will instantiate a SupplierEOImpl.
public void create()
{
  OAViewObject vo = getSuppliersVO();
  vo.insertRow(vo.createRow());
  // Always call this after you perform a row insert. See the Entity Object
  // New / Initial section below for additional information.
  vo.setNewRowState(Row.STATUS_INITIALIZED);
}
```

The view object `createRow()` method calls the `create()` method on the underlying entity object. Add any defaulting/initialization code to the `create()` method on the entity object as shown for the ToolBox Tutorial `oracle.apps.fnd.framework.toolbox.tutorial.server.SupplierEOImpl` class.

Warning: Do *not* put any initialization logic in the entity object's constructor; it should always be added after the `super.create(attributeList)` method call in the `create()` method).

Tip: If your defaults can be determined at design time, and are appropriate for a specific UI, you can also set default values by setting the `Initial Value` item property in the OA Extension. These values can be personalized by customers; they don't have to subclass your entity object and override the `create()` method to set the defaults. See the Defaulting topic in Implementing the View for additional information.

```
/**
 * In the SupplierEOImpl class; initialize a new supplier.
 */
Public void create(AttributeList attributeList)
{
  super.create(attributeList);
  OADBTransaction transaction = getOADBTransaction();

  // Supplier id is obtained from the table's sequence
  Number supplierId = transaction.getSequenceValue("FWK_TBX_SUPPLIERS_S");
  setSupplierId(supplierId);

  // Start date should be set to sysdate
  setStartDate(transaction.getCurrentDBDate());
} // end create()
```

Tip: When you set values in your entity object, always call `set<AttributeName>(val)` instead of `setAttribute("<AttributeName>", val)` as performance improves when the lookup step is bypassed. To skip any programmatic attribute validation but still perform declarative validations defined for the corresponding attribute, call `setAttributeInternal()` directly. See Entity Object and View Object Attribute Setters for additional information.

Composite Entity Associations

BC4J automatically sets the parent primary key attribute values on child entity objects in a composition association. The parent primary key values are passed in the child's `attributeList` `create()` method parameter and set during the call to `super.create(attributeList)`.

Do not try to populate parent primary key values yourself.

Entity Object Initial / New State

By default, entity objects are created with the row state of `STATUS_NEW`, and BC4J adds them to its validation and post listener lists. In this case, any event that triggers a validation or database post sequence includes these entity objects.

As stated in the OA Framework Model Coding Standards, always circumvent this behavior by explicitly calling the `setNewRowState(STATUS_INITIALIZED)` method on its containing `ViewRowImpl` immediately after you insert the newly created row. See the code example above. This sets the state of any associated entity objects to `STATUS_INITIALIZED`.

When you do this, BC4J removes the corresponding entity objects from the transaction and validation listener lists, so they will not be validated or posted to the database. As soon as the user makes a change (an attribute "setter" is called), the entity object's state changes to `STATUS_NEW`, and BC4J returns it to the validation/post lists. You can also call `setNewRowState(STATUS_NEW)` on the `ViewRowImpl` to change the state manually at any time.

Special "Create" Cases

"Flattened" Master/Detail in a Single Row

In the OA Framework ToolBox Tutorial, we have composite master/detail entities that we display to the user in a single, "flattened" row. A purchase order can include many lines, which can in turn include many shipments, but in our UI, we present the line and shipment in a 1:1 relationship.

Although BC4J can easily create several different entity objects for a single view object row -- where these entity objects are unrelated or are peers -- you need to intervene if one of these objects is a child of the other. In this case, you must add the following custom `create()` method to your view object row implementation to ensure that the correct parent key values are set in the lower-level entity:

```
// The following is required to support the creating the master/detail line
// and shipment entities that have been "flattened" into a single row in
// POLineShipFullVO with a 1:1 join.
//
// If you don't do this, BC4J will automatically treat them like peers and
// try to set the po header id as the parent key for both entities.

protected void create(oracle.jbo.AttributeList nvp)
{
  PurchaseOrderLineEOImpl lineEO = (PurchaseOrderLineEOImpl)getEntity(0);
  PurchaseOrderShipmentEOImpl shipmentEO = (PurchaseOrderShipmentEOImpl)getEntity(1);

  try
  {
    // Create Lines EO
    lineEO.create(nvp);

    // Create Shipments EO
    shipmentEO.create(lineEO);
    // Calling this ensures that any personalization default values are
    // properly set since the OAF normally sets this in the super.create(), but
    // since this is not called in this workaround, we need another method
```

```
    // to ensure customer defaults are applied.
    setDefaultValue();
  }
  catch (Exception ex)
  {
    lineEO.revert();
    shipmentEO.revert();

    if (ex instanceof oracle.jbo.JboException)
    {
      oracle.jbo.JboException jboEx = (oracle.jbo.JboException)ex;
      // Developers have to do the mapping on their own because of the override.
      jboEx.doEntityToVOMapping(getApplicationModule(), new oracle.jbo.ViewObject[]{getViewObject()});
      throw jboEx;
    }
    throw OAException.wrapperException(ex);
  }
} // end create()
```

Entity Object Cache

Once created, BC4J stores entity objects in a special transaction cache for a variety of reasons that are fully described in the JDeveloper BC4J documentation. Two important benefits are:

- Multiple view objects within the same root application module can share the same underlying entity objects. This means that changes made in one view object are immediately visible to any other view objects that reference the changed entities.
- Pending data changes are preserved within this cache even as the view object rowset is refreshed. For example, in a master-detail relationship, entity-derived attribute values on the *detail* view object persist in the cache even as the user navigates from master row to master row. All of your pending changes are preserved intact for the life of the transaction.

Understanding that this cache exists is important because you must explicitly interact with it to perform certain validations such as, when performing a uniqueness check you must look for duplicates in both the entity cache and the database.

There are three primary ways of checking data in both the cache and the database:

1. Call the findByPrimaryKey() method
2. Iterate the cache manually
3. Iterate the cache using an association

findByPrimaryKey( ) Method

The `findByPrimaryKey()` method is guaranteed to first search the cache for a match for the given key and entity object type, and then search the database. This is a very useful method, but not one that you want to use lightly since it instantiates entity objects for any matches that it finds in the database. It also pulls the *entire* entity object into memory, and not just the keys!). However, this method can -- and should -- be used in cases where you don't expect to find a match such as, when validating a sequence-generated primary key . It's also appropriate to use it when you need to call a method on the match so you need middle-tier access.

The following code from the `oracle.apps.fnd.framework.toolbox.schema.sever.SupplierEOImpl` class illustrates the use of this method:

```
public void setSupplierId(Number value)
{
  if (value != null)
  {
    // Supplier id must be unique. To verify this, you must check both the
    // entity cache and the database. In this case, it's appropriate
    // to use findByPrimaryKey( ) because you're unlikely to get a match, and
    // and are therefore unlikely to pull a bunch of large objects into memory.
    // Note that findByPrimaryKey() is guaranteed to check all suppliers.
    // First it checks the entity cache, then it checks the database.
    OADBTransaction transaction = getOADBTransaction();
    Object[] supplierKey = {value};
    EntityDefImpl supplierDefinition = SupplierEOImpl.getDefinitionObject();
    SupplierEOImpl supplier =
      (SupplierEOImpl)supplierDefinition.findByPrimaryKey(transaction, new Key(supplierKey));
    if (supplier != null)
    {
      throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
             getEntityDef().getFullName(), // EO name
             getPrimaryKey(), // EO PK
             "SupplierId", // Attribute Name
             value, // Bad attribute value
             "ICX", // Message application short name
             "FWK_TBX_T_SUP_ID_UNIQUE"); // Message name
    }
  }

  setAttributeInternal(SUPPLIERID, value);
} // end setSupplierId()
```

Manual Cache Iteration

You can perform the same checks that `findByPrimaryKey()` does by manually inspecting the entity object cache. You can then optionally perform the same checks against the database in a separate step. The advantage to this approach is you can avoid instantiating objects unnecessarily.

The following example is also from the ToolBox Tutorial `SupplierEOImpl` class:

```
public void setName(String value)
{
  // Since this value is marked as "mandatory," the BC4J Framework will take
  // care of ensuring that it's a non-null value. However, if it is null, we
  // don't want to proceed with any validation that could result in a NPE.

  if ((value != null) || (!("".equals(value.trim()))))
  {
    // Verify that the name is unique. To do this, we must check both the entity
    // cache and the database. We begin with the entity cache.
    com.sun.java.util.collections.Iterator supplierIterator =
      getEntityDef().getAllEntityInstancesIterator(getDBTransaction());

    Number currentId = getSupplierId();

    while ( supplierIterator.hasNext() )
    {
      SupplierEOImpl cachedSupplier = (SupplierEOImpl)supplierIterator.next();
      String cachedName = cachedSupplier.getName();
      Number cachedId = cachedSupplier.getSupplierId();

      // We found a match for the name we're trying to set, so throw an
      // exception. Note that we need to exclude this EO from our test.
      If (cachedName != null && value.equalsIgnoreCase(cachedName) &&
        cachedId.compareTo(currentId) != 0 )
      {
        throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                              getEntityDef().getFullName(), // EO name
                              getPrimaryKey(), // EO PK
                              "Name", // Attribute Name
                              value, // Attribute value
                              "ICX", // Message product short name
                              "FWK_TBX_T_SUP_DUP_NAME"); // Message name
      }
    }

    // Now we want to check the database for any occurrences of the supplier
    // name. The most efficient way to check this is with a validation view
    // object which we add to a special "Validation" application module.
    OADBTransaction transaction = getOADBTransaction();
    OAApplicationModule vam;
    // Look to see if the VAM has already been created in this transaction. If not,
    // create it.
    vam = (OAApplicationModule)transaction.findApplicationModule("supplierVAM");
    if (vam == null)
    {
      vam =
      (OAApplicationModule)transaction.createApplicationModule("supplierVAM",
         "oracle.apps.fnd.framework.toolbox.schema.server.SupplierVAM");
    }

    // Now, we use a lightweight "validation" view object to see if a supplier exists
    // with the given name.
    SupplierNameVVOImpl valNameVo = (SupplierNameVVOImpl)vam.findViewObject("SupplierNameVVO");
    valNameVo.initQuery(value);
    if (valNameVo.hasNext())
    {
      throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
            getEntityDef().getFullName(), // EO name
            getPrimaryKey(), // EO PK
            "Name", // Attribute Name
            value, // Attribute value
            "ICX", // Message application short name
            "FWK_TBX_T_SUP_DUP_NAME"); // Message name
    }
  }

  setAttributeInternal(NAME, value);
} // end setName()
```

Association Iteration

This is similar to `findByPrimaryKey()` in the sense that it's guaranteed to check both the entity cache and the database. It will also load any of

the entity objects its finds into the database into memory, which is useful if you want to call methods on the entity object. Unlike `findByPrimaryKey()`, which can look for entity objects of any type with any key, this can be used *only* to retrieve entity objects that are related to the current object with an association.

The following example illustrates the use of an association by a root composite entity object to find all of its children.

```
private void checkLineExists()
{
  // A purchase order header must have at least 1 associated line.
  // To check this, we first do a manual check of the entity cache
  // If we find a line for this header, we're done (note that the entity cache won't
  // include EOs that are DELETED or DEAD).
  com.sun.java.util.collections.Iterator fastIterator =
    PurchaseOrderLineEOImpl.getDefinitionObject().getAllEntityInstancesIterator(getDBTransaction());

  Number currentHeaderId = getHeaderId();
  while ( fastIterator.hasNext() )
  {
    PurchaseOrderLineEOImpl cachedLine = (PurchaseOrderLineEOImpl)fastIterator.next();
    Number cachedHeaderId = cachedLine.getHeaderId();
    // If we find a match, we're done. Don't forget to look ONLY for lines
    // for this header... The entity cache can include lines for other headers
    // also.

    If ((cachedHeaderId != null) && (cachedHeaderId.compareTo(currentHeaderId) == 0 ))
    {
      return;
    }
  }
  // We haven't found any matches in the cache yet, so now we need to check
  // the database...

  // In this example, we're illustrating the use of the association between the
  // header and its lines to iterate through all the shipments.  This will
  // check both the cache and the database, and will bring all the rows
  // into the middle tier.
  // Note that this looks only at lines for this
  // header so we don't need to filter our results, so it is convenient.
  RowIterator linesIterator = getPurchaseOrderLineEO();

  if (!(linesIterator.hasNext()))
  {
    throw new OARowValException(OARowValException.TYP_ENTITY_OBJECT,
                getEntityDef().getFullName(),
                getPrimaryKey(),
                "ICX", // Message product short name
                "FWK_TBX_T_PO_NO_LINES"); // Message name
  }
} // end checkLineExists()
```

Entity State

Each entity object has an associated "Entity State" that describes its state relative to the underlying database data and the transaction. You can test for these statuses in your code by calling `getEntityState()`.

Tip: BC4J automatically removes an entity object from the entity cache if its state is `STATUS_DEAD` so you don't need to worry about manually excluding this in your code if you're looking for "good" objects.

- **STATUS_NEW** - the entity object is new in the current transaction.
- **STATUS_DELETED** - the entity object originated in the database and has been deleted in the current transaction.
- **STATUS_MODIFIED** - the entity object originated in the database and has been changed.
- **STATUS_UNMODIFIED** - the entity object originated in the database and has not been changed, or it has been changed and those changes have been committed.
- **STATUS_DEAD** - the entity object is new in the current transaction and it has been deleted.
- **STATUS_INITIALIZED** - the entity object is in a "temporary" state and will not be posted or validated.

Update / Validate

This section describes how to correctly perform attribute-level and entity-level validation.

Attribute-Level Validation

As described in Implementing the View in Chapter 3, whenever an HTTP `POST` request is issued for a page with updateable values, OA Framework writes those values back to the underlying view object, which in turn writes the values to the underlying entity object(s) by calling its setters.

Since each attribute's validation should be added to its setters (see the ToolBox `PurchaseOrderHeaderEOImpl`'s `setHeaderId()` method below as an example), the process of calling the entity object setters executes attribute-level validation.

If you specify any declarative validation (for example, you indicate in the JDeveloper Entity Object Wizard that an attribute cannot be updated after it

is saved), this validation is performed in the `setAttributeInternal()` method that you should call after executing your own validation logic. It is also checked in `validateEntity()`.

```
/**
 * Sets the PO Header Id.
 *
 * Business Rules:
 * Required; cannot be null.
 * Cannot be updated on a committed row.
 */
public void setHeaderId(Number value)
{
  // BC4J validates that this can be updated only on a new line. This
  // adds the additional check of only allowing an update if the value
  // is null to prevent changes while the object is in memory.
  If (getHeaderId() != null)
  {
    throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                 getEntityDef().getFullName(), // EO name
                                 getPrimaryKey(), // EO PK
                                 "HeaderId", // Attribute Name
                                 value, // Attribute value
                                 "ICX", // Message product short name
                                 "DEBUG -- need message name"); // Message name
  }
  if (value != null)
  {
    OADBTransaction transaction = (OADBTransaction)getOADBTransaction();

    // findByPrimaryKey() is guaranteed to first check the entity cache, then check
    // the database. This is an appropriate use of this method because finding a
    // match would be the exception rather than the rule so we're not worried
    // about pulling entities into the middle tier.

    Object[] headerKey = {value};
    EntityDefImpl hdrDef = PurchaseOrderHeaderEOImpl.getDefinitionObject();
    PurchaseOrderHeaderEOImpl hdrEO =
      (PurchaseOrderHeaderEOImpl)hdrDef.findByPrimaryKey(transaction, new Key(headerKey));
    if (hdrEO != null)
    {
      throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                   getEntityDef().getFullName(), // EO name
                                   getPrimaryKey(), // EO PK
                                   "HeaderId", // Attribute Name
                                   value, // Attribute value
                                   "ICX", // Message product short name
                                   "FWK_TBX_T_PO_ID_UNIQUE"); // Message name
    }
  }
  // Executes declarative validation, and finally sets the new value.
  setAttributeInternal(HEADERID, value);
} // end setHeaderId()
```

Different "Set" Methods

There are several different ways of setting values within an entity object. In your code, you most often call `set<AttributeName>()` and `setAttributeInternal()`. See Entity Object and View Object Attribute Setters for additional information about all the possible options.

Cross-Attribute Validation

Any validation involving two or more attribute values on the entity should be included in the `validateEntity()` method; do not include any cross-attribute validation in an individual attribute's setter since attribute values can be set in any (random) order.

You may reference attribute values from referenced entities in your attribute-level validation.

Entity Validation

Whenever OA Framework sets entity object values during an HTTP `POST` processing cycle, it always validates any view object rows that it touches, which in turn calls `validateEntity()` on the underlying entity object(s). Furthermore, entities are validated again prior to posting (up to 10 times in a composition).

Any logic which operates at the row level -- and is not particularly sensitive to being called repeatedly -- should be included in the `validateEntity()` method.

The following `PurchaseOrderHeaderEOImpl` code illustrates typical entity-level validation:

```
/**
 * Performs entity-level validation including cross-attribute validation that
 * is not appropriately performed in a single attribute setter.
 */
```

```
protected void validateEntity()
{
  super.validateEntity();

  // If our supplier value has changed, verify that the order is in an "IN_PROCESS"
  // or "REJECTED" state. Changes to the supplier in any other state are disallowed.
  // Note that these checks for supplier and site are both performed here
  // because they are doing cross-attribute validation.
  String status = getStatusCode();
  if ((("APPROVED")Equals(status)) || ("COMPLETED"Equals(status)))
  {
    // Start by getting the original value and comparing it to the current
    // value. Changes at this point are invalid.

    Number oldSupplierId = (Number)getPostedAttribute(SUPPLIERID);
    Number currentSupplierId = getSupplierId();
    if (oldSupplierId.compareTo(currentSupplierId) != 0)
    {
      throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                   getEntityDef().getFullName(), // EO name
                                   getPrimaryKey(), // EO PK
                                   "SupplierId", // Attribute Name
                                   currentSupplierId, // Attribute value
                                   "ICX", // Message product short name
                                   "FWK_TBX_T_PO_SUPPLIER_NOUPDATE"); // Message name
    }

    // If our supplier site has changed, verify that the order is in an "IN_PROCESS"
    // state. Changes to the supplier site in any other state are disallowed.

    Number oldSiteId = (Number)getPostedAttribute(SUPPLIERSITEID);
    Number currentSiteId = getSupplierSiteId();
    if (oldSiteId.compareTo(currentSiteId) != 0)
    {
      throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                   getEntityDef().getFullName(), // EO name
                                   getPrimaryKey(), // EO PK
                                   "SupplierId", // Attribute Name
                                   currentSiteId, // Attribute value
                                   "ICX", // Message product short name
                                   "FWK_TBX_T_PO_SUPSITE_NOUPDATE"); // Message name
    }
  }
  // Verify that our supplier site is valid for the supplier and make sure it is
  // an active "Purchasing" site.
  SupplierEntityExpert supplierExpert =
    SupplierEOImpl.getSupplierEntityExpert(getOADBTransaction());
  if (!(supplierExpert.isSiteValidForPurchasing(getSupplierId(), getSupplierSiteId())))
  {
    throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                 getEntityDef().getFullName(), // EO name
                                 getPrimaryKey(), // EO PK
                                 "SupplierSiteId", // Attribute Name
                                 getSupplierSiteId(), // Attribute value
                                 "ICX", // Message product short name
                                 "FWK_TBX_T_PO_SUPSITE_INVALID"); // Message name
  }
} // end validateEntity();
```

Cross-Entity Validation

Developers often assume they need to implement "cross-entity" validation whenever one entity object simply calls a method on another during a validation cycle. In OA Framework, "cross-entity validation" means something very specific:

- Entity A and Entity B each reference the other during `validateEntity()` (so Entity A needs to get some attribute values from Entity B, and Entity B needs to get some attribute values from Entity A) and...
- You expect both objects to be "dirty" (require validation) in the same transaction and...
- It's important that the other entity object be valid before the referencing object retrieves attribute values to use in its own validation
  The problem comes down to a question of timing: which entity do you validate first?

Tip: This isn't an issue with master/detail entity objects associated with composition because the children will always be validated before the parent and the BC4J Framework actually validates the hierarchy up to 10 times from bottom to top until all entities are valid.

As you can see from this very precise definition, the circumstances that require OA Framework "cross-entity validation" are quite rare. If you do feel that you have a need for this, the solution involves creating a special "intermediary" object that implements the BC4J `ValidationListener` interface. In simple terms, this object decides who should be validated first and then performs the cross-entity validation.

See the Advanced Java Entity Object topic for an example of this.

Inappropriate Validation Failure Handling

Do not attempt to perform a rollback or clear the BC4J caches by calling `Transaction.rollback()`, `Transaction.clearEntityCache()` or `clearCache()` in your entity-level validation methods (`validateEntity()`, `set<AttribtueName>()` and so on). If you need to take these actions for any reason, you must catch entity validator exceptions at the *application module/transaction level* as shown below, and make whatever calls you need. For example, performing a rollback at the application module level is safe; performing a rollback or clearing the entity cache from within an entity object is not, and can lead to unpredictable behavior.

```
Bad Code:
---------
protected void validateEntity()
{
  ...
  DBTransaction txn = getDBTransaction();
  // Do not issue a rollback from within the EO.
  txn.rollback();
  throw OAException(...);
}
Good Code:
----------
protected void validateEntity()
{
  ...
  throw OAException(...);
}
// The following logic is written at the application-module level.
try
{
  txn.commit();
}
catch (OAException e)
{
  // Cache the exception thrown by the validation logic in the EO,
  // and perform the rollback.
  txn.rollback();
}
```

Delete

To delete an entity object, call the `remove()` method on its corresponding view object, as shown in the following application module code example. This example, iterates through a view object rowset looking for the matching object to delete:

```
/**
 * Deletes a purchase order from the PoSimpleSummaryVO using the
 * poHeaderId parameter.
 */
public Boolean delete(String poHeaderId)
{
  // First, we need to find the selected purchase order in our VO.
  // When we find it, we call remove( ) on the row which in turn
  // calls remove on the associated PurchaseOrderHeaderEOImpl object.
  int poToDelete = Integer.parseInt(poHeaderId);

  OAViewObject vo = getPoSimpleSummaryVO();
  PoSimpleSummaryVORowImpl row = null;
  // This tells us the number of rows that have been fetched in the
  // row set, and will not pull additional rows in like some of the
  // other "get count" methods.
  int fetchedRowCount = vo.getFetchedRowCount();
  boolean rowFound = false;
  // We use a separate iterator -- even though we could step through the
  // rows without it -- because we don't want to affect row currency.
  RowSetIterator deleteIter = vo.createRowSetIterator("deleteIter");
  if (fetchedRowCount > 0)
  {
    deleteIter.setRangeStart(0);
    deleteIter.setRangeSize(fetchedRowCount);
    for (int i = 0; i < fetchedRowCount; i++)
    {
      row = (PoSimpleSummaryVORowImpl)deleteIter.getRowAtRangeIndex(i);
      // For performance reasons, we generate ViewRowImpls for all
      // View Objects. When we need to obtain an attribute value,
      // we use the named accessors instead of a generic String lookup.

      // Number primaryKey = (Number)row.getAttribute("HeaderId");
      Number primaryKey = row.getHeaderId();
      if (primaryKey.compareTo(poToDelete) == 0)
      {
        row.remove();
        rowFound = true;
        getTransaction().commit();
        break; // only one possible selected row in this case
      }
```

```
    }
  }
  // Always close iterators.
  deleteIter.closeRowSetIterator();
  return new Boolean(rowFound);
} // end delete()
```

Validation and Cascade-Delete

The `row.remove()` method in turn calls the `remove()` method on the underlying entity object. To implement any special delete behavior such as, checking to see whether the delete action is allowed, or implementing cascade-delete, add code to your entity's `remove()` method as illustrated below for the ToolBox's `PurchaseOrderHeaderEOImpl`.

Note: Since the coding standards discussed in the "Controlling Window, Block and Region Behavior" chapter of the current *Oracle E-Business Suite Developer's Guide* prohibit the use of the cascade-delete feature in the database, BC4J Framework requires that we manually implement our own cascade delete for the multi-tier purchase order business object. To do this, each entity object deletes all of its immediate children before calling `super.remove()` for itself. This is also illustrated below.

```
/**
 * Marks all the lines for deletion, then mark the header for deletion.
 * You can delete a purchase order only if it is "In Process" or "Rejected."
 */
public void remove()
{
  String status = getStatusCode();

  if (("IN_PROCESS"Equals(status)) || ("REJECTED"Equals(status)))
  {
    // Note this is a good use of the header -> lines association since we
    // want to call remove( ) on each line.
    RowIterator linesIterator = getPurchaseOrderLineEO();
    if (linesIterator != null)
    {
      PurchaseOrderLineEOImpl line = null;

      while (linesIterator.hasNext())
      {
        line = (PurchaseOrderLineEOImpl)linesIterator.next();
        line.remove();
      }
    }
    super.remove(); // Must be called last in this case.
  }
  else
  {
    throw new OARowValException(OARowValException.TYP_ENTITY_OBJECT,
                                getEntityDef().getFullName(),
                                getPrimaryKey(),
                                "ICX", // Message product short name
                                "FWK_TBX_T_PO_NO_DELETE"); // Message name
  }
} // end remove()
```

Lock

BC4J supports the following locking techniques:

- **Pessimistic Locking** - BC4J locks an entity object's database row when any `setAttribute()` method is called (specifically, before any changes are made). If the row is already locked, BC4J throws an `AlreadyLockedException`. This is the default BC4J locking mode.
- **Optimistic Locking** - BC4J locks an entity object's database row during the database post processing logic. If the row is already locked, BC4J throws an `AlreadyLockedException`.

Note: OA Framework uses optimistic locking by default and recommends that you not deviate from this since connection pooling makes traditional pessimistic locking unfeasible. However, for those of you with "Oracle Forms" development experience, Oracle E-Business Suite uses pessimistic locking in Forms-based applications.

If you are certain that you require pessimistic locking, you must change the transaction's behavior as follows:

```
// In the application module...
OADBTransaction txn = getOADBTransaction();
txn.setLockingMode(Transaction.LOCK_PESSIMISTIC);
```

Stale Data Detection

When BC4J locks a row, it tries to determine if the row has been deleted or changed by another user since it was queried for the current user.

- If it the row was deleted, BC4J throws a `RowAlreadyDeletedException`.
- If changes are detected, BC4J throws a `RowInconsistentException`.

To overwrite the default column-by-column comparison behavior for the change check, use the attribute-level **Change Indicator** flag in the entity

object attribute definition wizard. If this indicator is selected for an attribute, BC4J limits its comparison to this attribute. Oracle E-Business Suite PL/SQL API's commonly use an `OBJECT_VERSION_NUMBER` table column to determine the data change, and this column can be leveraged for entity objects also. See the Object Version Number Column section below.

Commit

When you're ready to commit entity object changes, simply call `getTransaction()Commit()` from an application module as illustrated in the rollback example below. When you call this method, your object(s) are validated if needed, posted and committed.

1. The `commit()` method calls the `oracle.apps.fnd.framework.OADBTransaction.validate()` method.
2. The `validate()` method checks the "Validation Listener" for a list of root entity objects that need to be validated. (In a multi-entity composition, only the root entity object is added to the validation list). If you just finished validating an object before committing it, it won't be on this list because when an object validates successfully, BC4J removes it from the validation list).

   Tip: You can also call the `OADBTransaction.validate()` method directly at any point in your code if you want to force a validation cycle. It will function the same way.
3. Assuming an object is on the validation list, the `OADBTransaction validate()` method calls the `final validate()` method on the entity object, which in turn calls `validateEntity()` to execute your validation logic.

   Note: The order in which BC4J validates individual entities on the list should be considered random. However, *within* a composite business object, such as a purchase order with lines and shipments, BC4J always validates any children *before* their parent. To achieve this, BC4J puts only the root entity object of a composition on the validation list (children are not included). When this root entity object calls `super.validateEntity`, BC4J calls `validate` on its children, and so on throughout the hierarchy. For this reason, you should generally add your own validation logic *after* calling `super.validateEntity` to ensure that a parent object validates itself after its children validate themselves.
4. The `commit` method calls the OADBTransaction `postChanges` method.
5. The `postChanges` method checks the "Post Listener" for a list of entity objects whose data must be posted to the database.
6. For any objects on the post list, the OADBTransaction `postChanges` method calls `postChanges` on the entity object. When an object is posted, BC4J removes it from the post list.
7. Assuming no errors are encountered, a database commit is issued and any database locks are released.

Rollback

OA Framework implements an "all or nothing" transactional approach for post and commit actions. Regardless of the error severity, if the database post or commit fails, OA Framework:

- Issues a JDBC rollback to release database locks.

  Note: This does not adversely affect the state of the middle tier.
- Reverts the view object row state so that a second attempt can be made for the transaction

Note: This means that you don't need to explicitly rollback failed entity object transactions; OA Framework automatically displays a user-friendly error message if the post or commit fails. Consider the following example illustrating a commit and subsequent display of a "Confirmation" dialog after the user selects an **Apply** button:

```
// In the root application module
public void apply()
{
  getTransaction()Commit();
}
// In the controller
public void processFormData(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormRequest(webBean);
  // Handle the user pressing the "Apply" button
  if (pageContext.getParameter("Apply") != null)
  {
    OAApplicationModule am = pageContext.getRootApplicationModule();
    // No need for any special exception handling.  You can just display the
    // confirmation message because the OAF won't reach this code if the post/commit
    // fails.
    am.invokeMethod("apply");
    OAException confirmMessage =
      new OAException("ICX", "FWK_TBX_T_SUPPLIER_CREATE_CONF", null,
                      OAException.CONFIRMATION, null);
    pageContext.putDialogMessage(confirmMessage);
  }
}
```

Rollback Methods

To manually clear the middle tier view object and entity object cache, call `getTransaction().rollback()` from an application module. This will also roll back any database changes and clear any values that you cached on the transaction. See Supporting the Browser Back Button to understand how this can be a useful tool when creating entity objects.

If you are executing any PL/SQL procedures and need the ability to explicitly roll back the database without impacting the middle tier, call

`getTransaction().executeCommand("rollback")` from an application module.

Note: As explained in BC4J Native JDBC Statement Management, `Transaction.rollback()` closes the JDBC ResultSets (cursors) associated with the view object query collections by issuing `vo.clearCache()`. Therefore, if you issue the following calls in sequence:

```
vo.executeQuery();
Transaction.rollback();
vo.next();
```

the SQL exception"ORA-01002: fetch out of sequence", which often results from open cursors invalidated by the rollback call in the database, will not occur because `Transaction.rollback()` closes the cursors, forcing `vo.next()` to reopen a new valid cursor by re-executing the view object query.

While `Transaction.rollback()` rolls back the database state as well as the middle-tier business object state, these following direct JDBC calls are not intended to rollback any middle tier business object state and therefore do NOT close open JDBC cursors:

i. `Transaction.executeCommand("rollback")` call or
ii. BC4J's "rollback to savepoint" database call which is internally issued when the `Transaction.postChanges()` or `Transaction.commit()` call fails with validation or post errors. Although the entity object or view object data with its user changes remain in this case, the entity post state is changed back to the modified state so that the user can re-attempt the post/commit.

The BC4J Framework does not compensate for the JDBC or database rollback limitation that results in invalid JDBC and database cursors that occur when cursors are opened after the database savepoint and invalidated due to the rollback call. Therefore, if you need to use `Transaction.executeCommand("rollback")`, please review the M52 model coding standards first.

If you need to override post handlers or `beforeCommit` in `EntityImpl`, be sure to refer to Inappropriate Post Handling first.

Inappropriate Post Handling

Avoid calling `executeQuery()` in either of the following situations:

- On a view object in `EntityImpl's` post handler methods (`postChanges`, `beforePost`, `afterPost`).
- In `beforeCommit` and then later trying to retrieve rows from that same view object using `vo.next()`, `vo.first()`, and so on.

The reason is because when `Transaction.postChanges()` or `Transaction.commit()` fails due to validation or post errors, it issues a rollback to a database savepoint and invalidates any cursors that were opened after the savepoint -- see the note above for details.

Instead, we recommend the following coding practices:

Option 1: If you need to call `executeQuery()` on a view object:

1. Call it in the `EntityImpl.afterCommit()` method after calling the corresponding super method, or after a successfull `Transaction.commit()` call.
2. Call it after a successful `Transaction.postChanges()` call if the view object rows are retrieved and consumed before a `Transaction.commit()` call.

Option 2: If you must call `executeQuery()` in the `EntityImpl's` post handlers, then you must also override `handlePostChangesError` to close the invalid cursors as follows:

```
  protected void handlePostChangesError()
{
  super.handlePostChangesError();
  ViewObject vo = <get the view object where query was issued>;
  vo.clearCache();
}
```

Subsequent row navigation methods like `vo.next()` will force the query to re-execute.

As you can see, Option 1 is easier.

Object Version Number Column

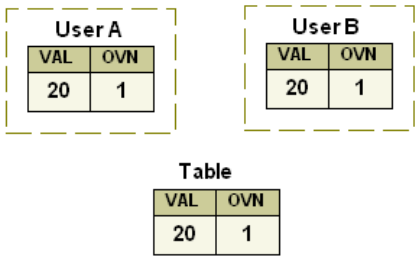
An OBJECT_VERSION_NUMBER (OVN) column (corresponding to the `ObjectVersionNumber` EO attribute) is used for locking rows in the database. Every table that can be updated or deleted by users should have an OVN column on it. When a new row is inserted into a table, the OVN column value for that row is set to a nominal value, typically one. The OVN value is then incremented whenever the row is updated. This value is preserved until the next update or delete and is never decremented or reset to a previous value.

Whenever a client queries a row the current OVN value of the row is always returned with the other attributes. If the object (row) is modified by the client and posted to the database, the OVN value on the database is compared with the OVN value that is posted:

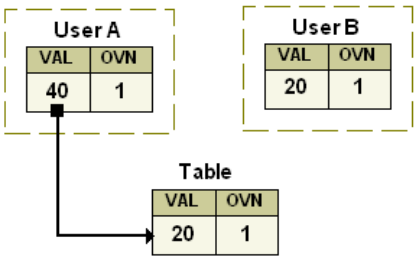
- If the values are the same, then the posted server row state is equivalent to the state when the client queried the row. As no changes have occurred, the row is posted and the OVN number is incremented on the server.
- If the values are different (database value higher than the posted value), it implies that another user has already changed and committed another row on the database. Therefore, the current change post is not allowed since the modifications made by the other user may be overwritten and lost.

The following is an example of this concept:

Step 1: Two users, User A and User B, read values from a server. The OVN attribute value is read with all other values.

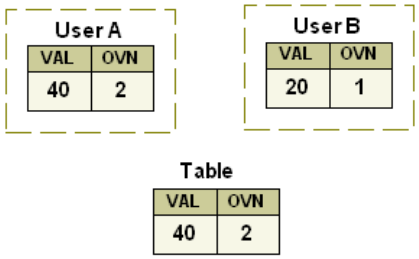


Step 2: User A changes a value and attempts to post this to the server.

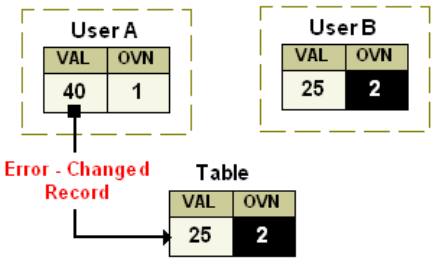


Step 3: The posted OVN value is compared against the server OVN value:

Case1: If the OVN values are identical the post succeeds and the OVN value is incremented.



Case 2: User B updates and posts a record before User A posts it. User A then posts his record. The OVN values do not match, the post fails with a "record changed by another user" error.



Note: The OVN value is not unique and in no way replaces the row primary key. Many rows in the same table can have the same OVN value. An OVN value indicates a particular version of a primary key row. Also, the database locking is *not* overridden by the deferred locking method. This prevents uncommitted changes being overwritten by another user.

Standard Rules

Use the following rules while creating the `ObjectVersionNumber` attribute in your entity object:

Rule 1: If your database table has an OBJECT_VERSION_NUMBER column, then map it to the `ObjectVersionNumber` attribute. This attribute is optional.

Rule 2: If your database column name is *not* OBJECT_VERSION_NUMBER and if you have an equivalent column with a different column name, then map the database column to the attribute name `ObjectVersionNumber` through the BC4J entity object wizard.

You can also use other columns like LAST_UPDATE_DATE for implementing your locking mechanisms. However, even if you update the LAST_UPDATE_DATE with the current datetime, it may not be safe because two users might update the same record at the same time and end up with the same value for the LAST_UPDATE_DATE column. Its better to use a column like OBJECT_VERSION_NUMBER for this purpose.

Rule 3: Set the `ObjectVersionNumber` attribute as "Change Indicator" through the entity object wizard and include the column in your view

object SQL. The value of the attribute set as the Change Indicator will be compared with the database column value upon locking. If no attribute is marked as the "Change Indicator", then all attribute values are compared with the respective column values for locking.

OA Framework updates the `ObjectVersionNumber` attribute in the generic OAEntityImpl Java class. That is, the `createObjectVersionNumber()`, and `updateObjectVersionNumber()` methods perform automatic initialization and update of these values. The BC4J Framework takes care of the locking, which includes comparing the original attribute values against the database column values to detect data staleness.

Change Indicator with Entity Associations

If set as the "Change Indicator", OA Framework also uses this property to make the master EO's Change Indicator dirty in an entity association whenever a child is modified or removed. For example, assume that you have some logic on your master EO that calculates the sum of an amount column from its children EOs. If one of the children EOs was modified by another user, then in order to always calculate the sum correctly, any changes to any of the child EOs should ensure that the master EO is also marked dirty and posted. A change on the "Change Indicator" property on the child EO ensures that the "Change Indicator" of the master is also modified, and is therefore marked dirty.

Note: If you do not want to lock the parent of a composite association when the children are changed, then you must not check the change indicator checkbox for the child. In this case, the child EO will not be able to take advantage of the default BC4J locking mechanism using the change indicator attribute.

Entity Objects that subclass OATLEntityImpl

The PL/SQL calls to the table handlers are hard coded in the OATLEntityImpl class. You should not modify the hard coded implementation as the `ObjectVersionNumber` is automatically managed by the entity object.

The table handlers generated for _TL tables implement their own locking by calling the appropriate table handler's `lock_row()` PL/SQL procedure. This procedure compares the originally queried values with the current database values. Therefore, the entity objects subclassing OATLEntityImpl are not required to have the `ObjectVersionNumber` attribute. However, if you do have the OBJECT_VERSION_NUMBER column in your TL table, you can override the default implementation of your table handler's `lock_row()` PL/SQL procedure to just compare the OVN column instead of comparing all columns.

Note: If you do include the `ObjectVersionNumber` attribute in your TL entity object, set it as "Change Indicator" to take advantage of the standard OA Framework behavior as described above.

Standard WHO Columns

OA Framework provides standard Oracle E-Business Suite WHO column support. All entity objects with WHO columns should include the following attributes:

- CreatedBy
- CreationDate
- LastUpdatedBy
- LastUpdateDate
- LastUpdateLogin

If your entity object does not include the standard WHO attributes, simply provide a no-op implementation for the standard WHO attribute setter methods.

Error Handling

All OA Framework application error handling is described in Chapter 3: Error Handling.

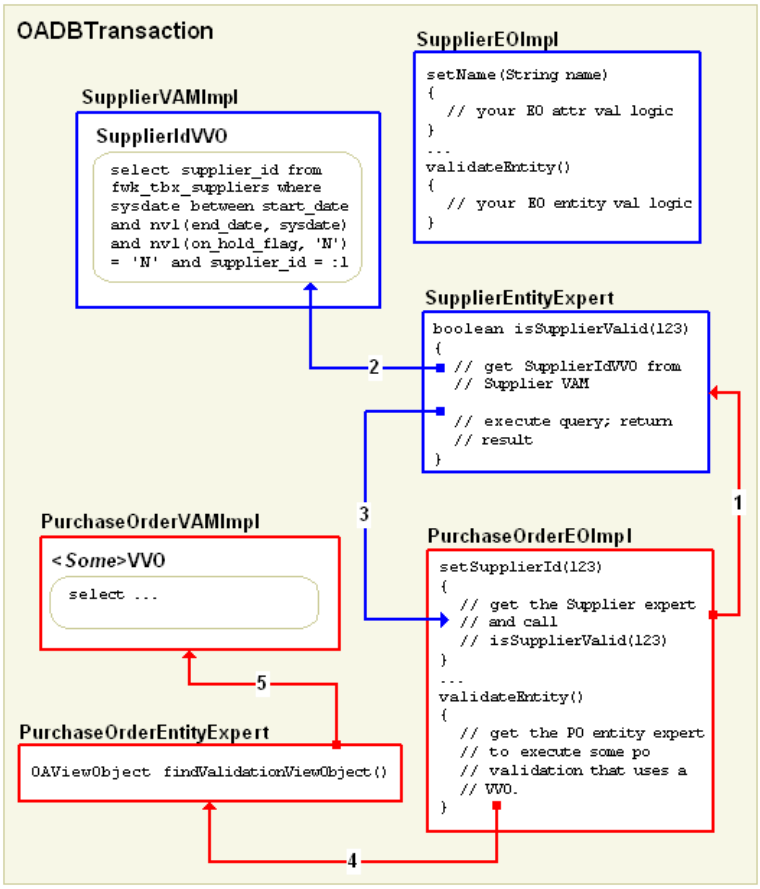
Entity Experts, Validation Applications Modules and Validation View Objects

As described in Implementing the Model,

The following diagram illustrates the runtime relationship between two entity objects and their experts, VAMs and validation view objects. For the PurchaseOrderEntityImpl, it illustrates the use of another entity object's expert to perform foreign key validation. It also shows how the Purchase Order entity object can use its own expert, VAM and VVOs to perform simple, internal validations.

The specific steps shown on the diagram are described below:

Figure 1: relationship between VAMs, VVOs, EOs and Entity Experts



Step 1: The supplierId value is set on the PurchaseOrderEOImpl. The "setter" makes sure that this is a valid supplierId for use on a purchase order. Rather than instantiate a SupplierEOImpl to perform this validation or add the code directly to the purchase order which breaks encapsulation, the purchase order EO gets a reference to the SupplierEO's expert singleton and calls its `isSupplierValid()` method.

Step 2: Within the `isSupplierValid()` method, the SupplierEntityExpert finds the validation view object that it needs to perform the SQL test for the given ID (SupplierIdVVO) in the SupplierEO's validation application module.

Step 3: The `isSupplierValid()` method returns **true** or **false** to the purchase order.

Step 4: Later, in the purchase order's `validateEntity()` method, it needs to perform some simple SQL. To do this, it asks its associated `PurchaseOrderEntityExpert` for access to one of its validation view objects.

Step 5: The entity expert finds the validation view object in the PurchaseOrderEO's validation application module.

What Code Goes Where?

When should code be added to the expert, and when should it stay in the entity object? We recommend that you follow these guidelines:

1. Always consider putting the logic in the entity object first.
2. If you have logic that you want to share (and we're about to define 2 distinct types of "sharing") then you have a few additional options:
   - If the logic is to be shared by multiple entity objects WITHIN a business object (so it's really internal logic) then you should add it to the entity expert, or use a simple helper class if the shared logic involves extensive processing that should be modularized.
   - If the logic is to be called by entity objects within another business object, you should add it to the entity expert. If you add it on the entity object only, the calling classes have no choice but to instantiate your entity object to execute the logic. For example, you have a `SupplierEO`, a `SupplierEntityExpert` and a `PurchaseOrderHeaderEO`. The `SupplierEntityExpert` includes the following method, which was added specifically to be called by the `PurchaseOrderHeaderEO` and other classes that reference suppliers. This is so they don't have to instantiate a `SupplierEO` to perform this lightweight validation:public Boolean `isSupplierValid(Number supplierId)` This method performs a simple SQL check to verify that the supplierId exists in the database and the supplier's END_DATE is null. (It uses a "validation" view object and the supplier's "validation" application module to do this).The `SupplierEOImpl` class also includes a `setSupplierId(Number value)` method, which verifies that the given Id value is unique and not null. It doesn't check the END_DATE because the `SupplierEOImpl` itself never needs to perform this kind of validation.

One of the most common validations you'll perform with entity objects involves verifying that a foreign key is valid. Sometimes this is a simple existence check but frequently the validation is more elaborate.

The use of entity objects presents a few interesting challenges and considerations:

- Given that "real-world" Oracle E-Business Suite entity objects aren't small, you don't want to pull these objects into the middle tier unless you really need substantial processing assistance from them. BC4J will instantiate a foreign key entity object when you call an accessor to

reference its key from the main entity object.
- Even if you're willing to accept the cost of validating foreign keys using fully populated association objects, you might find that the team that "owns" the object you want to reference hasn't written a fully-featured Entity Object for it yet and doesn't plan to within your release horizon.
- The natural tendency here is to write concise, lightweight SQL statements to perform these validations.

Tip: Why bother with the entity expert methods like `isSupplierValid()` when you can just access the underlying validation application module and its view objects directly? While the supplier entity object can certainly work directly with these resources, referencing objects should respect that they are actually part of the supplier's implementation. When the purchase order agrees to call a method to find out if the supplier is valid instead of using the view object directly, the supplier implementer is free to:

- Optimize performance by checking the cache first .
- Leverage any future BC4J enhancements that might make the use of an entity instance preferable to the validation application module solution .
- Add validation logic that imposes additional constraints on the query.

Note: Refer to Figure 7: "BC4J Validation Flow Chart" of the *Business Rules in BC4J* technical reference paper (page 48), available on OTN, for an illustration of the BC4J validation processing that occurs when a transaction is committed.

Calling PL/SQL Functions and Procedures

Even when writing Java entity objects, you might need to call PL/SQL functions or procedures.

Note: Do *not* use JDBC to perform simple SQL statements. Always leverage view objects for this purpose. If possible, you should define the view object declaratively.

In general, to invoke a stored procedure from within an entity object or an application module, you need to:

1. Create a JDBC CallableStatement with the PL/SQL block containing the stored procedure invocation
2. Bind any variables.
3. Execute the statement.
4. Optionally retrieve the values of any OUT parameters.
5. Close the statement.

The following application module example shows how to create and use a CallableStatement in an entity object.

```
import java.sql.CallableStatement;
import java.sql.SQLException;
import java.sql.Types;
...
OADBTransaction txn = getDBTransaction();
CallableStatement cs =
  txn.createCallableStatement("begin dbms_application_info.set_module(:1, :2); end;");
try
{
  cs.setString(1, module);
  cs.setString(2, action);
  cs.execute();
  cs.close();
}
catch  (SQLException sqle)
{
  try { cs.close } catch (Exception(e) {}
  throw OAException.wrapperException(sqle);
}
```

This example illustrates an OUT parameter:

```
import java.sql.CallableStatement;
import java.sql.SQLException;
import java.sql.Types;
import oracle.jdbc.driver.OracleCallableStatement;
...
DBTransaction txn = getDBTransaction();
String sql = "BEGIN :1 := FND_MESSAGE.GET; END;";
CallableStatement cs = txn.createCallableStatement(sql, 1);
String messageBuffer = "";

try
{
  ((OracleCallableStatement)cs.registerOutParameter(1, Types.VARCHAR, 0, 2000);
  cs.execute();
  messageBuffer = cs.getString(1);
  cs.close();
}
catch (SQLException sqle)
{
  try { cs.close } catch (Exception(e) {}
  throw OAException.wrapperException(sqle);
}
```

Entity Objects for Translatable (_TL) Tables

Important: The use of `OATlEntityImpl` has been deprecated. To migrate `OATLEntityImpl` to the new TL entities model you must complete the following steps to create the _TL entity and association, making sure that you change the current entity object to extend `OAEntityImpl`. This entity object should have already been built on top of the database _VL view and therefore, you also need to add a custom property to indicate the base table name: OA_BASE_TABLE = {base table name}.

To create entity objects for your translatable (_TL tables), follow these instructions:

Step 1: Use these rules to create an entity object for your _TL table. (*Not* the _VL view) :

- Name it **<*Entity*>TLEO**. For example, for the table FWK_TBX_LOOKUP_CODES_TL in the ToolBox Tutorial, we created an entity object named **LookupCodeTLEO**.
- Include all 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**.

  Note: For Release 12.2, make sure the entity object for your _TL table omits the `ZdEditionName` attribute, as it is only for internal use by Online Patching.

- Identify the table's primary keys including the `LANGUAGE` column.
- Verify that this extends `OAEntityImpl` like any other OAF entity object.
- Add whatever validation logic you need for this entity and its attributes. The translatable values are unlikely to need any special validation.

By default, all the attributes in the _TL table will be considered translatable if they are:

- Not a primary key attribute and
- Not an entity accessor and
- One of the following types: VARCHAR, CHAR, FIXED_CHAR, LONGVARCHAR, CLOB

If you need to override this default behavior, create a custom BC4J attribute-level property named **OA_TRANSLATABLE** and set its value as follows:

- If the property value is set to **true**, the attribute will be considered translatable attribute.
- If the property value is set to **false**, the attribute will not be considered translatable.

Step 2: Create an entity object for your _VL view following these rules.

- Use the regular entity object naming convention. For example, for the FWK_TBX_LOOKUP_CODES_VL view in the ToolBox Tutorial, the corresponding entity object is named **LookupCodeEO**.
- Do not include the `RowId` pseudo-column. Include all the other columns in the view.
- Identify your primary keys as you normally would.
- Create an entity-level custom BC4J property named **OA_BASE_TABLE** with a value naming the true base table of your translatable entity. In this example, this value would be set to FWK_TBX_LOOKUP_CODES_B.

OA Framework will automatically override the entity's `doDML` method to ensure that all inserts, updates and deletes are actually performed on the base table identified this property. All reads will be done against the _VL view.

Step 3: Create an association between the _VL (source) and _TL (destination) entity objects following these rules:

- Follow the standard association object naming convention that describes the entity relationships. For example: **LookupCodeToTLAO**.
- The association should be designated as a composition with **1:*** cardinality. Be sure to *deselect* the **Cascade Delete** checkbox that is enabled when you choose the composition option. The base entity should be selected as the source, and the _TL entity as the destination.
- Change the **Accessor Name** in the **destination entity** to OA_TL_ENTITIES.

Step 4: Create views objects that use your translatable entities.

When creating view objects to access your translatable entities, always use the entity object created for the _VL view. For example, `LookupCodeEO`. Do *not* use the _TL entity object `LookupCodeTLEO` directly. For the purpose of any code that needs to access your translatable entity, you should treat the base EO as the *only* entity object. Coordination between the base and _TL entities is handled automatically and the _TL entity should remain "invisible". Otherwise, you can treat your base EO like any other EO.

_TL Table with no Corresponding _B Table

For those rare cases where you have a _TL table and _VL view and no _B table, because all of the attributes are translatable; you can still define the base entity on the database view **_VL**, set the **OA_BASE_TABLE** property to be the **_VL** view name, and then override `doDML()` for the base entity to do nothing as this is going to be a virtual entity that does not have an underlying database table.

Standard Validation Patterns and Examples

This section provides standard implementations for the following common tasks:

- Enforce Non-Null Value
- Enforce Unique Value (Primary Key)
- Enforce Unique Value (Other Value)
- Validate a Foreign Key (Lightweight)
- Enforce Update Only When Entity is New
- Conditionally Disallow Update

- Check Existence of Child Entity
- Perform "Last-Minute" Processing
- Implement Cascade Delete
- Coordinate with Parent/Child Entities
- Maintain Transient Member Variables
- Cross-Attribute Validation
- Cross-Entity Validation

Enforce Non-Null Value

Check the **Mandatory** check box in the JDeveloper entity object wizard for all required attributes.

When this is checked, the BC4J Framework ensures that the value is non-null, and it will perform this test both in `validateEntity()` and in the attribute's setter in the `setAttributeInternal()` method.

To check whether a value is null, always check it in both the setter and in `validateEntity()`, unless it's only conditionally required. In this case, by definition as a cross-attribute test, it should only be performed in the entity validation.

We also recommend adding a further restriction for system-generated primary key values: disallow a value change once it has been set -- even if this row is still new. This avoids the problem of having children in memory with a different key value.

Tip: If you code additional validation for these values you should still verify that they are non-null before working with them to avoid a `NullPointerException`. This is because BC4J won't perform its validation until you call `setAttributeInternal()` -- *after* any validation you write.

Enforce Unique Value (Primary Key)

Whenever you're verifying that a system-generated primary key value is unique, you can safely use the `findByPrimaryKey()` method to quickly look for matches since it's guaranteed to check both the cache and the database. Even though it will pull any database matches into the middle tier, finding a match is unexpected.

Enforce Unique Value (Other Value)

Whenever you need to verify that a value is unique, and you don't want to pay the price of pulling objects into memory for testing purposes, you must perform a 2-part manual check:

1. First, iterate through the entity cache looking for a match.

   Tip: Whenever you're looking for objects in the entity cache, remember that it potentially includes instances that aren't interesting to you. For example, if you're trying to verify that a given purchase order line's "user key" line number is unique, you must filter the cache using it's parent's PK. Furthermore, the cache includes the current entity object that you're validating, so you would most likely need to filter that out also.

2. If you don't find a match in the entity cache, then you want to execute a lightweight database query. To do this, use a validation view object designed for this test.

Validate a Foreign Key (Lightweight)

Assuming you have no reason to pull the foreign key's complete entity object into memory (in other words, you're not going to call any methods on it), you typically want to perform this validation in a "lightweight" manner. To address this, obtain the foreign key's associated entity expert and call a method asking if the value is valid.

Tip: If a foreign key doesn't have an associated entity expert to support this kind of validation, the next best strategy is to write your own validation view object (VVO) and include it in your entity's validation application module (VAM). You can create multiple validation application modules if necessary to help keep your code more modular.

Enforce Update Only When Entity is New

Select the **Update When New** radio option in the JDeveloper entity object wizard for all attributes that cannot be changed on a committed row. When this is checked, the BC4J Framework ensures that the entity object is in a `STATUS_NEW` state before allowing the update. BC4J performs this test in the `setAttributeInternal` method.

Conditionally Disallow Update

In this case, allow updates to an attribute only when certain conditions, beyond the entity transaction status, are satisfied.

For example, in a purchase order header you cannot update the SupplierId or the SupplierSiteId if the purchase order's StatusCode is not "IN_PROCESS." Although the purchase order header's state is set to "IN_PROCESS" when initialized, you still cannot reliably reference this value in the supplier/site setters because it could change at the same time that the supplier/site values change.

For this reason, this validation falls into the category of "cross-attribute" validation and should always be written in `validateEntity`.

Check Existence of Child Entity

It's not uncommon for composite business objects to require that one or more children exist before a parent entity considers itself to be valid. For example, a purchase order header can't be saved without any lines, and the lines can't be saved without any shipments.

To implement this, you need to do a 2-part check (similar to the process of verifying that a value is unique): first you iterate through the entity cache

looking for any children belonging to the current parent, and if you don't find any, you check the database. As soon as you find a match, you're done.

Note: The timing of this particular check. This isn't something you want to do early in the entity object's life cycle because you could easily be validating a parent before you've even had a chance to create children.

Perform "Last-Minute" Processing

Sometimes you need to wait until just before posting the data before setting some attribute values or performing explicit validation. For example, financial products that generate "user key" document identifiers don't want to waste sequence numbers on abandoned transactions (customers often require that all such identifiers be accounted for to avoid auditing issues).

To address this, you can add this logic by overriding the `postChanges()` method.

Note: Since both the `validateEntity()` and `postChanges()` methods may be called multiple times before the transaction is committed (particularly if you manually call `postChanges()` at some point during your transaction so you can execute a SQL query that will include new/modified rows), it's not an appropriate place for processing code that absolutely must be called only once. For example, starting an associated workflow process. In this case, we recommend that you consider adding your logic to `beforeCommit()`, or `afterCommit()`. Obviously, `beforeCommit()` it's too late for any code that sets values on the entity object, in which case you should use `postChanges()` only if you can't use `validateEntity()`.

Tip: Given that `postChanges()` can be called multiple times, and it's also possible -- although fairly rare -- for `beforeCommit()` to be called multiple times if the initial database commit fails, we recommend that you add a transient attribute to your entity object to keep track of whether the key method has been called.

Implement Cascade Delete

The coding standards discussed in the "Controlling Window, Block and Region Behavior" chapter of the current *Oracle E-Business Suite Developer's Guide* prohibit the use of a declarative Cascade Delete Constraint in the database. BC4J currently stipulates that this is a prerequisite for automated cascade delete. Since Applications tables do not satisfy this prerequisite, we must implement cascade delete manually.

Coordinate with Parent/Child Entities

It's fairly common to encounter cases where a child needs to "notify" its parent of a change within itself so the parent can react accordingly. For example, a `PurchaseOrderLineEOImpl` implements logic to apprise the `PurchaseOrderHeaderEOImpl` of any changes to its line total (price * quantity) so the header can update the transient `OrderTotal` value that it maintains.

Tip: Although BC4J supports a "publish/subscribe" model as a means of letting entity objects notify one another of important events, this is most appropriately used when you don't know who might be interested in the news. In this case, where the line knows that its header is the only interested party, it's best to simply call a method directly on the parent.

A parent entity object can also act as the "single source of truth" for its children whenever they need to maintain a value that must be validated across the children. For example, the purchase order header maintains current maximum line number for all of its children. When a line needs a new line number, it asks the header for the next number in the sequence.

If you need to set values in a child entity object, you simply use the association iterator to find the instance(s) you're interested in and call whatever method you need.

Maintain Transient Member Variables

It is appropriate to create transient member variables in your entity objects if you need them to facilitate calculations (performance enhancers). By definition, these values won't be serialized with the object, unlike entity object attributes, so you should always initialize them with a check value that you can use to trigger referencing logic to "start over" if necessary.

For example, a `PurchaseOrderLineEOImpl` maintains a transient member variable (`mMaxShipNum`) for tracking the current maximum shipment number. If that variable hasn't been initialized when someone calls `getMaxShipNum()`, the code steps through the process of checking both the entity cache and the database to get the current maximum shipment number. The next time `getMaxShipNum()` is called, this expensive initialization is skipped.

Cross-Attribute Validation

Whenever an attribute's validation depends on the value of another attribute in the same entity object, you *must* write the code for this in your `validateEntity()` method (or a delegate). You *cannot* reliably perform this validation in the attribute setter.

Tip: You can use attribute values in reference entities during attribute setter validation. This rule applies only to attributes in the same entity object.

Cross-Entity Validation

Developers often assume they need to implement "cross-entity" validation whenever one entity object simply calls a method on another during a validation cycle.

In OA Framework, "cross-entity validation" means something very specific:

- Entity A and Entity B each reference the other during `validateEntity()` so Entity A needs to get some attribute values from Entity B, and Entity B needs to get some attribute values from Entity A, and...
- You expect both objects to be "dirty" (require validation) in the same transaction, and...
- It's important that the *other* entity object be valid before the referencing object retrieves attribute values to use in its own validation

The problem is a question of timing: which entity do you validate first?

Tip: This isn't an issue with master/detail entity objects associated with composition. This is because the children will always be validated before the parent and the BC4J Framework actually validates the hierarchy up to 10 times from bottom to top until all entities are valid.

As you can see from this very precise definition, the circumstances that require OA Framework "cross-entity validation" are quite rare. If you feel that you have a need for this, the solution involves creating a special "intermediary" object that implements the BC4J `oracle.jbo.server.ValidationListener` interface. In simple terms, this object decides who should be validated first and then performs the cross-entity validation.

PL/SQL Entity Object Design and Development

Overview

A PL/SQL entity object provides an object representation of the data from a table or view and routes the DML (update, insert,delete, and lock) operations to stored procedures in the database. It must extend the abstract class `oracle.apps.fnd.framework.server.OAPlsqlEntityImpl` and can then override any of the following methods for its DML operations and provide JDBC calls when applicable (make sure not to call super in your implementation).

```
void insertRow();
void updateRow();
void deleteRow();
void lockRow();
```

You should use the PL/SQL EOs **only** if you have legacy PL/SQL code that maintains all your transactions and validations. If you are a new product and/or do not have a lot of PL/SQL legacy code, the OA Framework recommends the use of Java EOs over PL/SQL EOs.

You should create one PL/SQL EO per business function that is associated with one PL/SQL package unlike the Java EOs where you create one EO per database object. Note that though the approach of one PL/SQL EO per business function is very similar to PL/SQL VOs (VOs that extend `OAPlsqlViewObjectImpl`), you must use PL/SQL EOs and not PL/SQL VOs. PL/SQL VOs have been deprecated, and if you have any PL/SQL VOs in your product, you should plan on converting them to PL/SQL EOs instead.

This document uses examples to illustrate how you would handle different aspects of your PL/SQL EO manually.

Contents

- Create
- Insert
- Lock
- Update
- Delete
- Rollback
- WHO Column Support
- Error Handling
- PL/SQL Entity Objects for _TL Tables

Prerequisite Reading

- Chapter 5: Implementing Server-Side Features - Java Entity Objects
- Chapter 6: Advanced OA Framework Application Topics - Advanced Java Entity Object Development Topics

Create

You can create a PL/SQL EO just like you would create a java entity object. The only difference between creating a Java EO and a PL/SQL EO is making sure that your PL/SQL EOs extends `oracle.apps.fnd.framework.server.OAPlsqlEntityImpl`. You can find details on how to create a Java EO in Chapter 5: Implementing Entity Objects.

Creating Primary Keys

You should use the view object wizard to set the Primary Key in the designer, or set it programmatically in your VORowImpl. In the programmatic case, if your primary key column will be null at the time `insertRow()` is called, then you can create it either in the Java layer or in the PL/SQL layer as shown below.

Case 1: If you want to create your primary key(s) in the Java layer.

You should use logic similar to the following to set your primary key in the Java layer:

```
public void create(AttributeList attributeList)
{
   setAttribute(EMPLOYEEID,
                yourJavaMethodToReturnPrimaryKey()); //EMPLOYEEID is your PK.
}
```

Case 2 : If you want to create your primary key(s) in PL/SQL, and if your primary key(s) should be populated on the EO after your row insertion in the database.

For this case, you need to perform the refresh-on-insert operation manually as shown below (marking the attribute as "refresh-on-

insert" has no effect).

Step 1 : You need to set temporary, unique primary key values in your `create()` method using the `OATemporaryKeyFactory` as shown below (the convenience method `getTemporaryKey()` on the `OAExtentityExpert` makes use of this factory).

Note: Do not use `System.currentTimeMillis` to get a unique number.

```
public void create(AttributeList attributeList)
{
   super.create(attributeList);

   // Set a temporary primary key
   OAEntityExpert expert = trxn.getExpert(this.getEntityDef());
   AttributeList tempKey = expert.getTemporaryKey();

   Object[] tempKeyAttrNames = tempKey.getAttributeNames();

   for(int i = 0; i < tempKeyAttrNames.length; i++)
   {
     String keyAttrName = tempKeyAttrNames[i];
     setAttribute(keyAttrName, tempKey.getAttribute(keyAttrName));
   }
   ...
}
```

Step 2 : During the `doDML()` cycle, the OA Framework calls the `insertRow()` method. You should call your PL/SQL procedure to insert in this method, and your PL/SQL procedure should return the actual primary key.

You should then populate it on the EO using:

```
protected void insertRow()
{
   ...
   populateAttribute(empIdIndex,
                     empId,
                     false, // don't send notification
                     false, // don't mark as changed
                     false); // don't save original value
   ...
}
```

For composite associations, BC4J takes care of propagating the updated primary key values to any children that reference them as foreign keys.

Insert

You should call your PL/SQL insert procedure in your `insertRow()` method **without** calling the `super()`. You should create your callable statement each time in your `insertRow(), updateRow() and deleteRow()` method. The OA Framework has deprecated the use of the `getCachedStatement()` method.

You should call the `checkErrors()` method after executing your callable statement to handle exceptions. This will be discussed in length in the section on Error Handling.

Here's an example of how to call a PL/SQL procedure INSERT_RECORD in your `insertRow()` method.

Warning: Do not bypass the insert PL/SQL call within your `insertRow()` method since BC4J doesn't have any way to know that you didn't actually perform the insert, and might try to lock the row at some point since it expects it to exist.

```
protected void insertRow()
{
   try
   {
     String insertStmt = "BEGIN INSERT_RECORD( " +
                                 "PARAM1 => :1, " +
                                 "PARAM2 => :2, " +
                                 "PARAM3 => :3, " +
                                 "PARAM4 => :4); END;";

     DBTransaction trxn = getDBTransaction();
     CallableStatement insertStmt = trxn.createCallableStatement(insertStmt, 1);

     // Rebind parameters
     insertStmt.setString(1, getFirstName());
```

```
      insertStmt.setString(2, getLastName());
      insertStmt.setString(3, getAddress1());
      insertStmt.setString(4, getAddress2());

      // Execute the statement
      insertStmt.executeUpdate();

      // OAExceptionUtils.checkErrors as per PLSQL API standards
      OAExceptionUtils.checkErrors (txn);
    }
    catch(SQLException sqlE)
    {
       ...
    }
  }
```

Synchronizing values from PL/SQL on the EO

If you have any calculations for any of your EO columns in your PL/SQL procedure, and if you would like to reflect those values in your EO after row insertion in the database, then you should include those columns in your view object SQL and do the following:

```
txn.commit();
vo.executeQuery();
```

If you used methods like `populateAttribute()` and `revert()` to do the above synchronization, then you should change your code to issue an `executeQuery()` after `commit()` instead. The code change is required to make post and commit actions work correctly after previous post failures. There is no good way to revert the attribute values for already posted entities when post fails on some other entity if populateAttribute or revert is used.

Note that the above logic holds good for synchronizing all values but the primary keys. To perform a refresh-on-insert operation manually for your primary keys, you should use `populateAttribute()` for the primary keys as stated above in the Creating Primary Keys section, because BC4J uses the primary keys for bringing in rows while you execute the query.

If you do not want to execute the query, but would like to just clear the middle tier entity cache, then you can do so using the `clearEntityCache()` method on your EO. You may want to do this if you do not want to re execute the query to avoid the db roundtrip and you simply want to clean your middle tier state.

Lock

The OA Framework provides automatic locking of your entity objects using the `ObjectVersionNumber` attribute (corresponding to the OBJECT_VERSION_NUMBER column) and the "**Change Indicator**" property in your entity object. Please read the section on "Object Version Number" from Chapter 5: Implementing Entity Objects before continuing further.

As stated in the chapter above, you can rely on the super class OAEntityImpl to handle locking by relying on the default BC4J default optimistic locking mechanism where the table on which the EO is based is queried with the primary key to compare the OBJECT_VERSION_NUMBER column values. You can do so if your PLSQL EO Java class updates the OBJECT_VERSION_NUMBER column or if your PL/SQL EO is just based on one table only.

Also note that, previously, if your application code managed the `ObjectVersionNumber` attribute in your PL/SQL EO Java class similar to the generic OAEntityImpl's `createObjectVersionNumber()`, and `updateObjectVersionNumber()` methods, then you can remove your Java logic to rely on the OA Framework to manage these attributes.

Please read on if you perform locking in your PL/SQL procedure instead. You may want to do so when your PL/SQL EO is based on a complex view that joins multiple tables.

You should do the following if your PL/SQL procedure updates the OBJECT_VERSION_NUMBER column

- Override the following OAEntityImpl methods as no op in your subclass:
  - `createObjectVersionNumber()` -- Populates the `ObjectVersionNumber` attribute value upon entity creation.
  - `updateObjectVersionNumber()` -- Updates the `ObjectVersionNumber` attribute value upon update DML operation.

  Note that even if you do not override these methods as noops, your application will continue to work. However there will be redundant operations and gaps in the OBJECT_VERSION_NUMBER column values.

- As a part of your DML operation (`insert()`, `update()` or `delete()`) pass the value of the `ObjectVersionNumber` attribute to your PL/SQL procedure.

  If you have modified the `ObjectVersionNumber` attribute for some reason in your EO, you should remember to pass its value as retrieved from the database and not pass the changed value. You can do this by using the method `getPostedAttribute(attrIndex)` instead of `getAttribute()`.

- You can call your PL/SQL locking procedure in either of the following two hook points:
  - By overriding the `lockRow()` method on your EO and calling your own PL/SQL locking procedure to compare the OVNs.
  - By overriding `lockRow()` method on your EO as a no-op and calling your PL/SQL locking procedure from your PL/SQL insert, update or delete PL/SQL procedure.
- Your DML operation should change the value of the OBJECT_VERSION_NUMBER in the database in your PL/SQL procedure.
- You should finally synchronize the ObjectVersionNumber EO attribute values with the incremented OBJECT_VERSION_NUMBER database column values after the `insertRow()` and `updateRow()` operations. You can do this by calling `executeQuery()` on the view object after committing your transaction.

  ```
  txn.commit();
  vo.executeQuery();
  ```

  As stated above, this is required to ensure that post and commit actions work correctly after previous post failures.
- If you like to use the `updateRow()` method for comparing instead of the `lockRow()` method, you can use Approach 1 detailed in following section on Using the FND_FWK_COMPATIBILITY_MODE to do so.

Even if you lock and increment the `ObjectVersionNumber` attribute through PL/SQL procedures, you should still set it as the **Change Indicator** in your entity object in the designer. When set, the OA Framework uses this property to make the master EO's Change Indicator dirty in an entity association as described in Chapter 5: Implementing Entity Objects. It is explained further in the paragraphs below.

In BC4J, the parent entity of a composite association will be locked if "Lock top-level container" is checked in the entity association at design time. The OA Framework uses the **Change Indicator** flag of a composite association's child as an indicator to dirty the parent. When the parent that is dirty is posted, its OVN will be incremented to indicate to others that the parent or one of its children has been changed. The above mechanism will increment the parent's OVN whenever a child is added, removed, or modified.

If you increment the parent's OVN in your PL/SQL procedure whenever a child is added, removed, or modified, then you don't have to set the change indicator flag.

Exception Handling with locking

You should return meaningful exceptions indicating that the record has been modified or deleted by another user when locking. You can use the message dictionary and throw an OAException for your error messages. Here are some exceptions and messages that you can use:

Exception Condition	Message Key	Message Text
Record is already locked in the database.	FND/FND_LOCK_RECORD_ERROR	Unable to lock the record. Cause: The record is being modified by another user.
Inconsistent row - record already changed in the database	FND/FND_RECORD_CHANGED_ERROR	Cause: The record contains stale data. The record has been modified by another user. Action: Re-query the record to get the new data.
Row Already Deleted in the database	FND/FND_RECORD_DELETED_ERROR	Unable to perform transaction on the record. Cause: The record has been deleted by another user. Action: Re-query the records to get the new data.

Using the FND_FWK_COMPATIBILITY_MODE

If you had custom code in 11.5.9 to increment and compare the `ObjectVersionNumber` attribute for locking in your based OAPlsqlEntityImpl subclass (PL/SQL EO), and if you implemented the comparison logic in the `updateRow()` method AND if you did not no-op your `lockRow()` method, then your comparison logic will incorrectly raise an exception when it is executed using OA Framework Release 12. You should handle this using the compatibility mode profile option, **FND_FWK_COMPATIBILITY_MODE**.

If you set the mode to 11.5.10 -- the OA Framework will perform automatic incrementation of the `objectVersionNumber` attribute if present.

If you set the mode to 11.5.9 -- the OA Framework will suppress automatic incrementation of the `ObjectVersionNumber` attribute.

If you already have custom logic in your 11.5.9 based PL/SQL EOs to handle the `ObjectVersionNumber` attribute's incrementation and comparison (for locking), and if you want to ensure that it works with **both** "11.5.9" and "11.5.10" compatibility modes, then change your code using one of the two approaches below.

Approach 1

Suppress the automatic initialization and incrementation of the `ObjectVersionNumber` attribute by inserting the following no-op methods into your `OAPlsqlEntityImpl` subclass:

```
protected void createObjectVersionNumber()
{
}

protected void updateObjectVersionNumber()
{
}
```

(No-op'ing `lockRow()` avoids a redundant select statement call from BC4J.)

```
protected void lockRow()
{
}
```

Approach 2

Move your comparison logic into the `lockRow()` method.

Update / Validate

You should call your PL/SQL update procedure in your `updateRow()` method **without** calling the `super()`. You should create your callable statement each time as mentioned above in this case as well.

You should call the `checkErrors()` method after executing your callable statement to handle exceptions. See the Error Handling section below for additional information.

Here's an example of how you can use the `updateRow()` method:

```
protected void updateRow()
{
  try
  {

    String updateStmt = "BEGIN UPDATE_RECORD( " +
                        "PARAM1 => :1, " +
                        "PARAM2 => :2, " +
                        "PARAM3 => :3, " +
                        "PARAM4 => :4); END;";

   DBTransaction trxn = getDBTransaction();
   CallableStatement updateStmt = trxn.createCallableStatement(updateStmt, 1);

   //Rebind parameters
   updateStmt.setString(1, getFirstName());
   updateStmt.setString(2, getLastName());
   updateStmt.setString(3, getAddress1());
   updateStmt.setString(4, getAddress2());

   //Execute the statement
   updateStmt.executeUpdate();

   //OAExceptionUtils.checkErrors as per PLSQL API standards
```

```
   OAExceptionUtils.checkErrors (txn);
  }
  catch(SQLException sqlE)
  {
    ...
  }
}
```

Validation

You can perform your validation of your attributes in either of the two places:

- In your `insertRow()` or `updateRow()` methods: You can perform your validation in java in either of these two methods, or in PL/SQL stored procedures that is called from the above two methods.
- In your `validateEntity()` method: If validations are done in a separate stored procedure in PL/SQL, then you can call that stored procedure in your `validateEntity()` method.

The exceptions raised as a result of validation in your PL/SQL stored procedures will be handled in the `OAExceptionUtils.checkErrors(txn)` method. See the Error Handling section below for additional information.

Delete

You should call your PL/SQL update procedure in your `deleteRow()` method **without** calling the `super()`. You should create your callable statement each time as mentioned above in this case as well.

You should call the `checkErrors()` method after executing your callable statement to handle exceptions. See the Error Handling section below for additional information.

Here's an example of how you can use the `deleteRow()` method:

```
protected void deleteRow()
{
  try

  {

     String deleteStmt = "BEGIN DELETE_RECORD( " +
                                "PARAM1 => :1, " +
                                "PARAM2 => :2, " +
                                "PARAM3 => :3, " +
                                "PARAM4 => :4); END;";

     DBTransaction trxn = getDBTransaction();
     CallableStatement updateStmt = trxn.createCallableStatement(deleteStmt, 1);

     // Rebind parameters
     deleteStmt.setString(1, getFirstName());
     deleteStmt.setString(2, getLastName());
     deleteStmt.setString(3, getAddress1());
     deleteStmt.setString(4, getAddress2());

     // Execute the statement
     deleteStmt.executeUpdate();

     // OAExceptionUtils.checkErrors as per PLSQL API standards
     OAExceptionUtils.checkErrors(txn);
   }
   catch(SQLException sqle)
   {
   ...
   }
 }
```

Delete Validation Implementation

If you have any validation logic in your delete stored procedure, you must explicitly call it *before* the deletion is actually performed. You would do this by overriding the `remove()` method on your `*ViewRowImpl` as follows:

```
public void remove()
{

  // Step 1: Call your delete validation stored procedure.  If
```

```
    // you want to use the same procedure that you use to perform
    // the delete operation, pass a flag so it can function in

    // "validate only" mode.

    ...

    // Step 2: If your validation succeeds, call super.remove()
    // to mark the entity for deletion.  Otherwise, throw an exception.

    super.remove();

}
```

If you don't do this, your row may appear to be successfully deleted when it actually remains in the database. Consider the following example:

1. The user selects a Delete icon in a table.
2. The view row is deleted from the query collection, and the PL/SQL entity object is marked as "deleted."
3. The user commits the transaction, and the commit triggers the `doDML()` for the entity.
4. Your delete logic called during `doDML()` throws a validation exception, so BC4J rolls back the state of the transaction to the save point at the beginning of the post cycle, and resets the entity's state.
5. BC4J doesn't do anything to the corresponding view row, so it is still missing from the collection. Since the user can't see it in the UI, she things it was deleted correctly.

For deletions, it's important to force validation *before* the point where BC4J changes the query collection and entity object states. Once the `doDML()` cycle is underway, it is too late.

Rollback

As with all entity objects, any exceptions thrown during the `doDML()` cycle causes a rollback of your data (till the previous save point).

If you need the ability to explicitly rollback the database without impacting the middle tier, you can call `getTransaction().executeCommand("rollback")` from an application module.

WHO Column Support

You should use null implementations for the `createRowWho()`, `updateRowWho()` methods. Your PL/SQL APIs in the `insertRow()` and the `updateRow()` methods should populate the WHO columns.

Error Handling

You should handle errors and exceptions in your PL/SQL procedure. You should then include the `OAExceptionUtils.checkErrors(txn)` method to throw these exceptions as shown in the insert, update and delete examples above.

This method is used to raise bundled exceptions registered by your PL/SQL stored procedures from the `FND_MESSAGE` PL/SQL package. It relies on the `FND_MSG_PUB.GET_DETAIL PL/SQL` procedure to get the error details. This method uses the flag that you set on the transaction to determine if and how to bundle the exceptions.

You should bundle all your validation exceptions and display them together. So for example, while posting a set of 10 rows, if the validation fails for the first row, then you do not want to stop the processing the other rows because the exception raised may be a validation exception and not a severe exception. In this case you want to bundle all validation exceptions raised while processing all the 10 rows and display them together on your page.

You can bundle your exceptions using the steps below:

Step 1 : Set the `postChanges` flag correctly in your transaction. You may want to do this soon after you create your application module.

```
txn.setPostChangesFlag(POST_ALL_RESET_ON_EXCEPTION);
```

This means that BC4J will attempt process all your entities, bundle exceptions raised during the posting cycle, rollback the transaction, and throw the bundled errors together.

Step 2 : Call `OAExceptionUtils.checkErrors()` to bundle your exceptions after your insert, update, or delete operation. You can use the following `checkErrors()` signatures:

- `checkErrors(OADBTransaction pTx)` -- Raises a bundled `OAException` of **all** errors registered by PL/SQL APIs.

- `checkErrors(OADBTransaction pTx, int pMessageCount)` -- Raises a bundled `OAException` of a **specified number** of errors registered by PL/SQL APIs.
- `checkErrors(OADBTransaction pTx, int pMessageCount, String pReturnStatus, String pMessageData)` --
  The `pReturnStatus` here is the status returned by your PL/SQL procedure. The possible values for `pReturnStatus` are:
  - "S" for success
  - "E" for error
  - "U" for unexpected error

  The OA Framework will displays messages only if the status code is E (signifying Errors) or U (Unexpected Errors). The `pMessageData` parameter here is the actual message in an encoded format as returned by the PL/SQL API call.

PL/SQL Entity Objects for _TL Tables

You should not use `OATLEntityImpl` to populate the MLS entities. Instead, you should code your public PL/SQL APIs to call the MLS table handlers.

Business Service Objects Architecture

Overview

Prerequisite reading

- Chapter 2: The Model
- Chapter 3: Implementing the Model

Contents

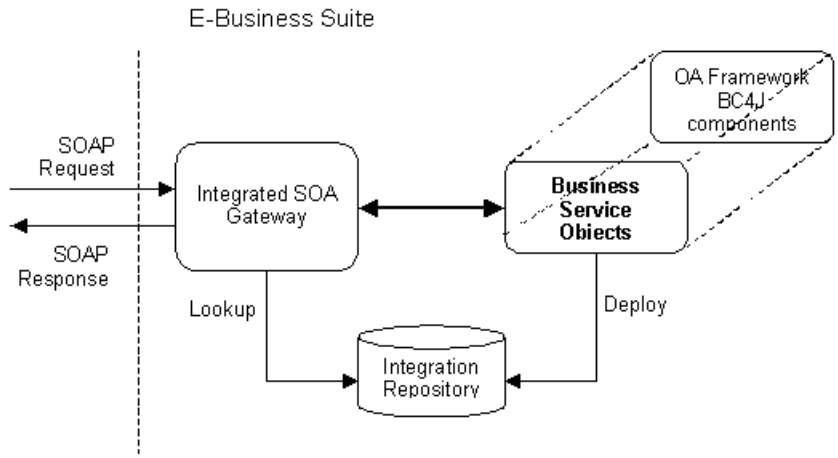
- Business Service Objects Architecture
  - Service Data Object
  - Service View Object
  - Service Application Module
  - Service Interface and Implementation
- Features and Benefits
- Design, Development and Deployment
- Lifecycle
- Authentication and Authorization
- Diagnostics
- Related Information

Note: After you read this document, see Building Business Service Objects for implementation instructions.

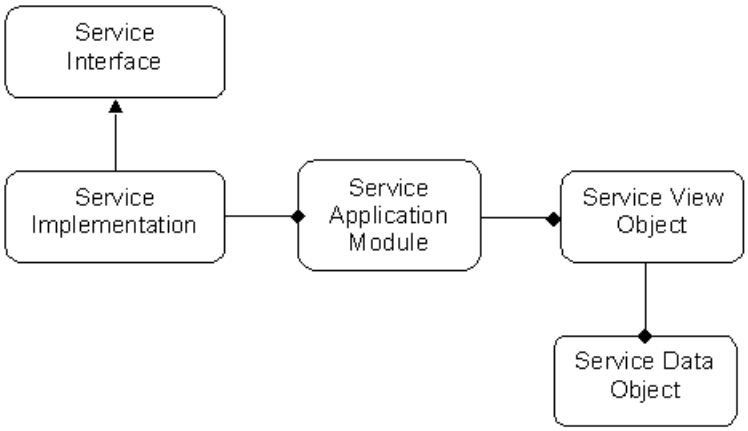
The Business Service Objects framework provides a function-rich, highly scalable and reliable SOA implementation framework for E-Business Suite business logic in the middle-tier. A Business Service Object (BSO) is a Java and XML-based, service-oriented implementation on top of an OA Framework business component. Business Service Objects are layered on top of OA Framework BC4J components such as the application modules, view objects and entity objects, and allow service-enablement of these middle-tier, application model artifacts.

Business Service Objects are exposed along with the rest of the E-Business Suite interfaces through the Integration Repository. Invocations of BSOs are mediated through the Integrated SOA Gateway (ISG) that interprets the incoming SOAP service request, looks up the required BSO service in the Integration Repository, instantiates the appropriate BSO and invokes the service.

Note: The OA Framework Release 12.1 Business Service Objects framework is an extension to the Service Bean framework first released in OA Framework Release 12.0. Similarly, the 12.1 Integrated SOA Gateway is an extension of the 12.0 Web Services Provider. Unless otherwise specified, these terms are used interchangeably.



The term Business Service Objects refers to five objects that participate together in service-enabling the OA Framework BC4J components - Service Interface, Service Implementation, Service Application Module, Service View Object and Service Data Object.



Service Data Object

A Service Data Object (SDO) is a Serializable value class that represents a row of data. SDOs represent business entities that get exchanged in service requests and service responses.

Service View Object

A service view object (SVO) is a regular BC4J view object augmented with metadata to service-enable the model. Each service view object has one service data object, and optional filter data objects and domain value set view objects. SVOs are contained in service application modules (see below).

- A **filter data object** is similar to an SDO except that it includes only Queryable attributes.
- A domain value set view object defines valid values for a view attribute in the SVO.

Service Application Module

A service application module (SAM) is a regular BC4J application module augmented with metadata to describe service-oriented methods that can be published in the Integration Repository. These service-oriented methods are defined in the service implementation class (see below).

Service Interface and Implementation

A service interface defines the public contract of an application and contains definitions of service-oriented methods. Each service interface has a corresponding service implementation class that implements the contract. The service implementation class holds a reference to the SAM, and can reach into SAM and SVO methods.

Features and Benefits

The key features and benefits of the Business Service Objects framework are listed below.

1. **Generic service-enablement framework** for the Java-based, middle-tier business logic.
2. **Minimal learning curve**, as its an incremental layer on top of OA Framework business components.
3. **Easy, step-by-step design time wizards** to guide through design and development.
4. **Integrated Security**, conforming to the E-Business Suite authentication and authorization mechanisms.
5. Support for EBS-specific artifacts like Flexfields and Attachments.
6. Comprehensive unit testing framework.
7. Integrated with ARU and AD to allow automatic deployment of services on patching.
8. **Highly scalable**, through the use of Application Module and Connection pooling.
9. **Full reusability** of existing business component code.
10. Consistent exception handling.
11. Support for batch operations.
12. Diagnostics, integrated with E-Business Suite logging and diagnostics framework.
13. **Standards based**, leveraging XML and SDO specifications.

Design, Development and Deployment

The Business Service Objects design, development, testing and deployment are covered in detail in separate chapters.

- Development [Lauren - to add link]
- Testing [Lauren - to add link]
- Deployment and Usage [Lauren - to add link]

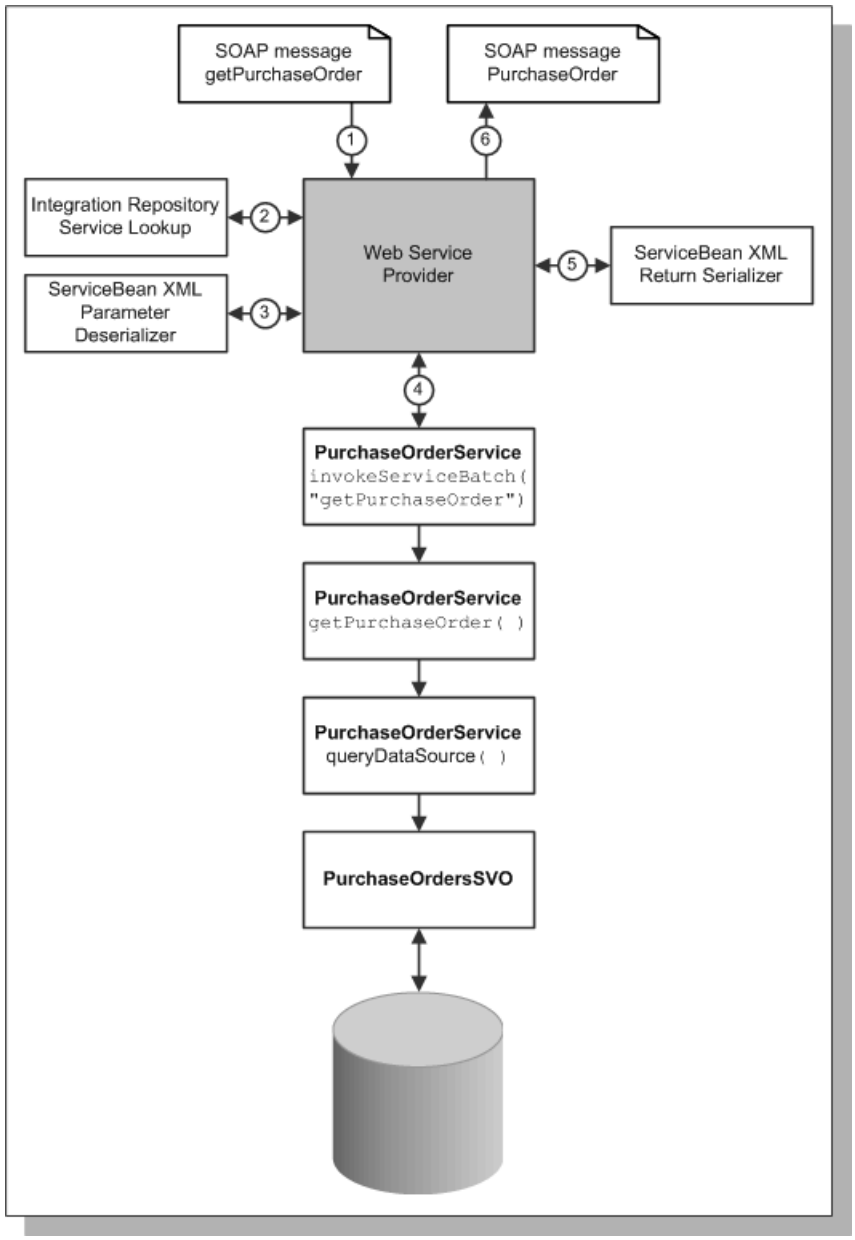
Lifecycle

This section describes how business service objects are accessed as web services from the client.

Method Invocation

In the following example, a client program makes a call to the `PurchaseOrderService`'s `getPurchaseOrder()` method which returns a `PurchaseOrder` service data object (the user wants to query a single purchase order). The numbered steps are described below.

Figure 1: Web service method invocation flow



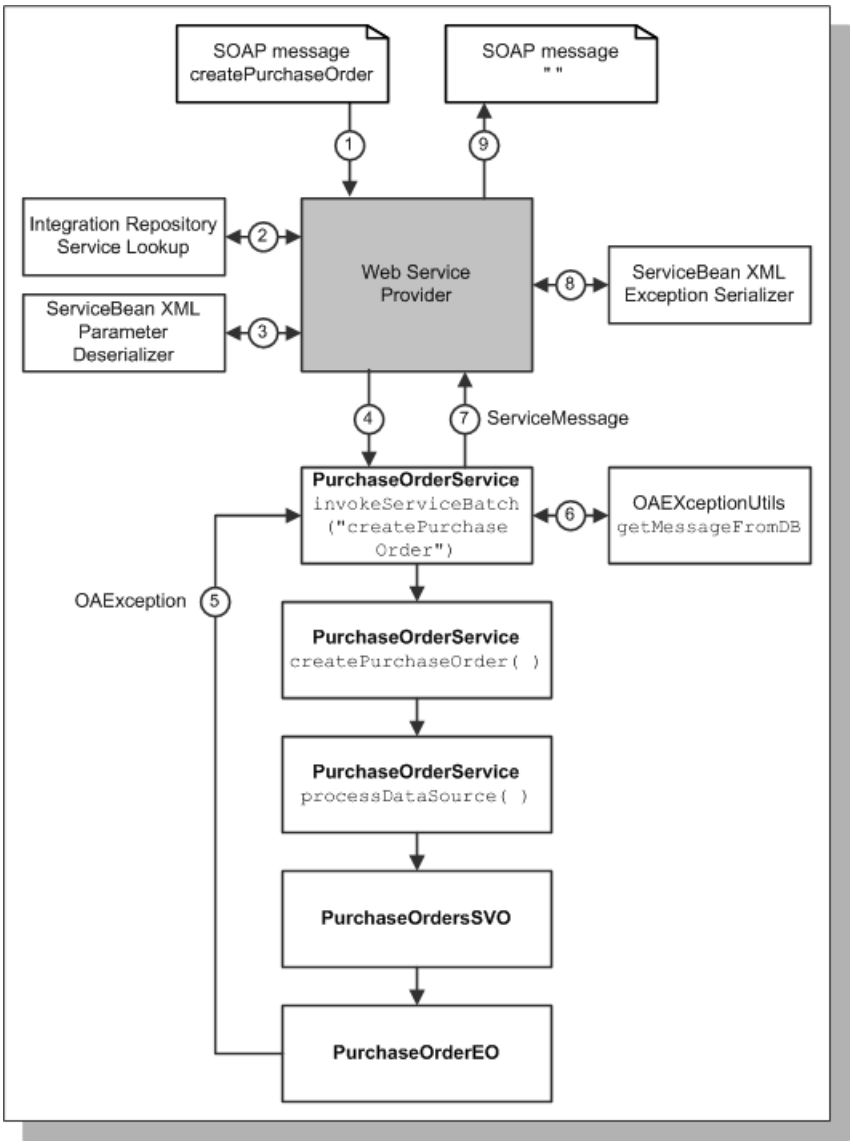
1. A SOAP message is sent to the web service provider telling it to invoke the `getPurchaseOrder()` method on the `PurchaseOrderService`.
2. The Oracle E-Business Suite web service provider (a servlet listening on a port for SOAP messages) looks up the service and the method in the Oracle E-Business Suite Integration Repository.
3. The web service provider calls the XML serializer to convert the inbound document from XML into a list of Java parameters.
4. The web service provider invokes the `getPurchaseOrder()` method using the service interface `invokeServiceBatch` method. This calls `getPurchaseOrder`, which internally delegates to `queryDataSource`. The `queryDataSource`

method executes a query against the `PurchaseOrdersSVO` view object, and returns a `PurchaseOrder` service data object document for the selected purchase order.
5. The web service provider calls the XML serializer to create an XML from the Java response.
6. A SOAP message is sent back to the client including the `PurchaseOrder` data.

Exception Handling

In the following example, a client program makes a call to the `PurchaseOrderService`'s `createPurchaseOrder()` method, which throws an `OAException` in the underlying entity object validation. The numbered steps are described below.

Figure 2: Web service method invocation flow with exception



1. A SOAP message is sent to the web service provider telling it to invoke the `createPurchaseOrder()` method on the `PurchaseOrderService`.
2. The Oracle E-Business Suite web service provider looks up the service and the method in the Oracle E-Business Suite Integration Repository.
3. The web service provider calls the XML serializer to convert the inbound document from XML into a list of Java parameters.
4. The web service provider invokes the `createPurchaseOrder()` method using the service interface `invokeServiceBatch` method. This calls `createPurchaseOrder`, which internally delegates to `processDataSource`. The `processDataSource` method inserts a row into the `PurchaseOrdersSVO` view object, which is based on the `PurchaseOrderEO` entity object.
5. The `PurchaseOrderEO` throws an `OAException` during the `create` operation.
6. The translated message for the exception is retrieved from the database.
7. A `ServiceMessage` is created with the error message information.
8. The web service provider calls the XML serializer to create an XML from the Java response.
9. A SOAP message is sent back to the client with a null data response.

Note: In the event of an application exception, the web service provider sends a standard SOAP message, not a SOAP Fault message. SOAP Faults are sent only in the event of system error.

Authentication and Authorization

The Integration Repository entries are stored in tables used by the Oracle E-Business Suite security model, which insures that all service interfaces and methods can be handled as secured objects. Specifically, OA Framework automatically creates a form function in FND_FORM_FUNCTIONS for each public method on the application's service interface, and as a convenience.

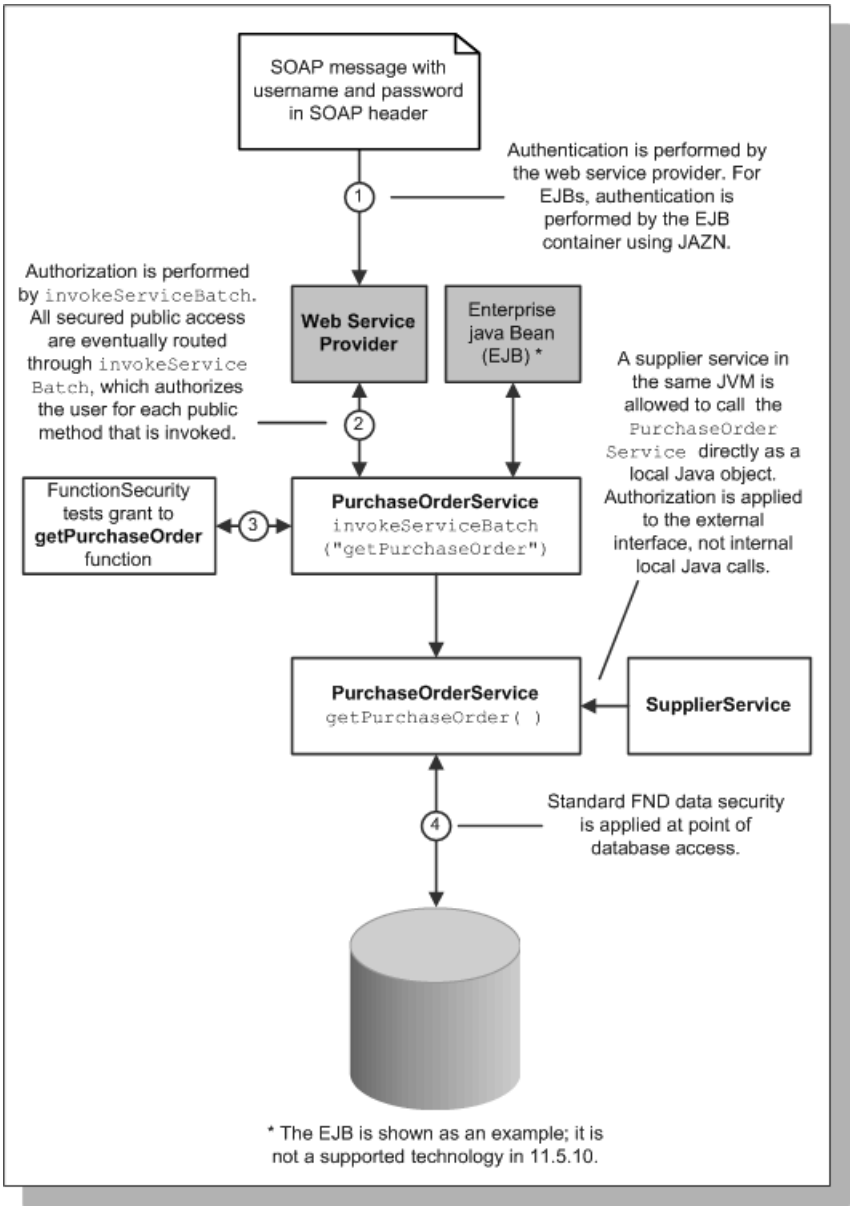
Note: OA Framework also seeds form functions for the generic methods (`processDataSource` and `queryDataSource`) if the corresponding interfaces are extended by the service interface. Functions are named according to the logical operations as shown below, and the assumption is that any corresponding custom methods on the service interface has the same name so a single function can secure both methods.

- `create<Object>` (for example, `createPurchaseOrder`)
- `delete<Object>` (for example, `deletePurchaseOrder`)
- `update<Object>` (for example, `updatePurchaseOrder`)
- `merge<Object>` (for example, `mergePurchaseOrder`)
- `get<Object>` (for example, `getPurchaseOrder`)

Developers and customers can create additional permission sets to assign different levels of access as required by different business functions.

The following diagram illustrates when and how authentication and authorization is performed for a `getPurchaseOrder()` method call on a `PurchaseOrderService`. Specifics for each step are provided below.

Figure 3: Security and authorization test points in a sample method web service or EJB method call



1. For a web service, the incoming SOAP message includes a username and password in the SOAP header.
2. User authentication is performed by the web service provider. For Enterprise Java Beans (EJBs), authentication is performed by the EJB container using Oracle's Java Authentication and Authorization Service (JAAS) provider (also known as "JAZN" for Java Authorization).
3. All service method calls are routed internally through the `invokeServiceBatch()` method if the client calls a custom service method. Alternatively, the client can call `invokeServiceBatch()` directly to invoke multiple method calls in a single message. In either case, `invokeServiceBatch()` performs a function security check to ensure that the user can invoke the method(s) at hand (in this case, the `getPurchaseOrder()` method).
4. If data security rules are defined, they are applied at the point of database access.

As noted in the diagram, if one service invokes methods in another, the "co-located" model function security checks are not performed for the individual method invocations.

Logging

Diagnostic messages can be printed from the Service layer using the writeDiagnostics method available in oracle.apps.fnd.framework.server.OADBTransaction. Since the integrated SOA gateway runs on the oafm instance of the application server, the log files will be present in oafmstd.out.

Related Information

- Business Service Objects in Detail
- Development
- Using Services

Business Service Objects in Detail

Overview

This document discusses:

- Service Classes and Interfaces
  - Core Interfaces and Method Invocation Classes
  - Service Implementation
  - Service Data Object
  - Request (Query) Objects
  - Domain Value Sets
  - Dynamic Attribute Groups
  - Description Objects

Service Classes and Interfaces

This section provides an introduction to service classes and interfaces including brief descriptions of key methods. For additional information about individual objects, see the associated API documentation. Unless otherwise stated, all classes and interfaces are included in the `oracle.svc` package.

Core Interfaces and Method Invocation Classes

The Service interface extends one of the three interfaces - `Service`, `DataProcessorService` or `DataSourceService`. The Service Data Object implements the `DataObject` interface.

The implementation of these four interfaces provides the following capabilities:

- Retrieve data using rich query criteria and other instructions to optimize performance.
- Insert, update and delete data.
- Bundle multiple operations into a single request/response message pair.

In the Oracle E-Business Suite, the default service implementation is an `oracle.apps.fnd.framework.service.ServiceImpl`. The application module's view objects (and associated entity objects) provide the data and corresponding validation logic for the service. Developers create an application-specific service by defining a custom interface that implements the standard interfaces. They then subclass `ServiceImpl` to implement their service interface. The development steps are described in detail in Business Service Objects Devlopment. [TODO: Fix Link]

Interface	Description
Service (interface)	Defines a service. Provides access to the service description and publishes a method that enables bulk method invocation. All services must extend this interface. Key methods include: BatchReponse `invokeServiceBatch(`BatchRequest `requests)` -- Batches multiple method invocations in a single message.

DataObject (interface)	Defines a `Serializable` row of data that can be used remotely or locally. For hierarchical business objects, a `DataObject` may include nested `DataObjects`. For example, a `PurchaseOrderDataObject` includes its lines, which in turn includes its shipments. Key methods include: • `Object get(String name)` -- Gets an attribute value. • `void set(String name, Object value)` -- Sets an attribute value. All service data objects must extend this base implementation. In local (co-located) mode, this class is a thin wrapper over an underlying view object row. See Service Data Object for more information about this interface. **Note:** This is a lightweight "value" object that is used to simply pass data back and forth between the service and a client program. As such, subclasses should never include references to other external objects, or include any validation / business rules. Data changes are local to this object until they are processed by the service.
DataSourceService (interface)	Getting instances of business objects from services using flexible criteria. All service interfaces that support query operations must extend this interface. The single method on this interface is: `List queryDataSource(String dataSourceName, DataCriteria dataCriteria, QueryControl queryControl)` -- Returns a root DataObject, which contains one attribute with type of List. See the Request (Query) Objects section below for additional information.
DataProcessorService (interface)	Enables posting (insert, update, delete) of DataObjects. All service interfaces that support insert, update, and/or delete operations must extend this interface. The single method on this interface is: `List processDataSource(String dataSourceName, String operation, ProcessControl control, DataObject rootDataObject)` -- Performs a data post operation. The user passes the name of the data source to update (the list of possible data sources is obtained by calling `ServiceDescription.getDataSourceDescriptions()`), the operation to perform (OPERATION_CREATE, OPERATION_UPDATE, OPERATION_DELETE or OPERATION_MERGE), ProcessControl instructions on how the methods should be invoked / responses returned, and a `List` of DataObjects. **Note:** A "merge" operation lets you combine inserts and updates in a single method call. **Tip:** List is a collection class defined in the`oracle.svc.util` package.

The following control and method invocation classes support the core interfaces described above.

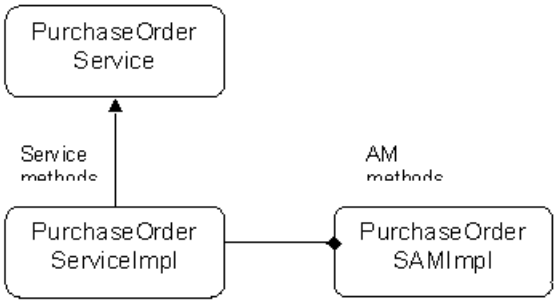
Class	Description

ProcessControl	Encapsulates instructions for how the results of all process data operations should be processed and returned. **Return Mode** • **RETURN_KEYS** -- Returns the keys for each posted data object in a sparsely populated document. (Includes both external and internally generated "surrogate" keys). This is the default for create and merge operations: • **RETURN_FULL** -- Returns a complete object for each posted data object by making a call to `queryDataSource()` at the end of the posting process. Note that the client must have function security access to `queryDataSource` for this to succeed. • **RETURN_NONE** -- Does *not* return any data objects. This is the default for update and delete operations **Exception Return Mode** • **RETURN_KEY** -- Returns the key of a failed data object. • **RETURN_FULL** -- Returns the entire failed data object. **Key Sets** A list of key set names that can be used to order the alternate key sets used to locate the row.
MethodRequest	A single method invocation to be used with `Service.invokeServiceBatch()`.
MethodResponse	A single method invocation response associated with `Service.invokeServiceBatch()`.
BatchRequest	Defines multiple method invocations for a call to `Service.invokeServiceBatch`. Key methods include: • `void setMethodRequest(List requests)` -- A List of MethodRequest objects. • `void setTransactionMode(String mode)` -- Specifies the transaction processing behavior for the batch method invocation. Mode options include: • **SINGLE** -- Single transaction for the entire batch and accumulate errors until done. The batch will be committed if no errors are detected, otherwise rolled back. • **MULTIPLE** -- A new transaction per method invocation and accumulate errors until done. Each method invocation is either committed or rolled backed if errors are detected. • **SINGLE_ABORT_ON_FIRST_ERROR** -- Single transaction for the entire batch and abort on the first error. The batch will be committed if no errors are detected, otherwise rolled back on the first error. • **BULK_PROCESSING** -- Used to import and export large amounts of data.
BatchResponse	Defines multiple method invocation responses for a call to `Service.invokeServiceBatch()`. Key methods include: `List getMethodResponse()` -- Returns a list of MethodResponse objects. `List getMessage()` or `List getErrorMessage` -- Returns any informational messages or errors raised during processing.
ServiceMessage	Container for any informational and error messages registered on the transaction by the service implementation. For example, if the implementation throws an `OAException`, it is represented as an error message on this object.
QueryControl	Defines the control switches to be used when retrieving data from a service.

See the API documentation for the core interfaces for examples of how to use these classes.

Service Implementation

The Service implementation class encapsulates the implementaion of all service-oriented methods that the Service interface defines. The Service implementation is an abstraction over the application module that allows segregation of service-oriented methods from other non-service business methods or UI-oriented methods that may be present in the application module. All Service implementation classes extend oracle.apps.fnd.framework.service.ServiceImpl. The ServiceImpl holds a reference to the application module instance it abstracts, and can invoke application module methods using this reference.



Service Data Object

Service Data Objects encapsulate data in document form passed to or retrieved from a Service. As documented in the chapter Business Service Object Development, OA Framework generates a data object for each service view object.

The following code example illustrates a simple `DataObject` with child rows:

Note: DataObjects are normally commented as described in the implementation section below. However, to facilitate a quick review of the content, comments are *not* included in this example.

Tip: Since `DataObjects` are simple objects with no validation, they can be sparsely populated. For example, if you define a service method that takes the following `PurchaseOrderdata object` as a parameter -- but only the purchase order header number is required for the given operation -- the client can populate this single value and ignore the others.

```
public class PurchaseOrder extends DataObject
{
  public PurchaseOrder()
  {
    super();
    mAttributeNames = ATTRIBUTE_NAMES;
  }

  /**
   * Gets Po Number.
   * @return Po Number
   */
  public Number getPoNumber()
  {
    return (Number)get(PO_NUMBER);
  }

  /**
   * Sets Po Number.
   * @param value Po Number. The Purchase Order Number
   */
  public void setPoNumber(Number value)
  {
    set(PO_NUMBER, value);
  }

  /**
   * Gets Supplier Id.
   * @return Supplier Id
```

```
   */
  public Number getSupplierId()
  {
    return (Number)get(SUPPLIER_ID);
  }

  /**
   * Sets Supplier Id.
   * @param value Supplier Id
   */
  public void setSupplierId(Number value)
  {
    set(SUPPLIER_ID, value);
  }


  ... // More attribute accessors


  /**
   * Gets Purchase Order Lines.
   * @return Purchase Order Lines
   */
  public List getPurchaseOrderLines()
  {
    return (List)get(PURCHASE_ORDER_LINES);
  }

  /**
   * Sets Purchase Order Lines.
   * @param value Purchase Order Lines
   */
  public void setPurchaseOrderLines(List value)
  {
    set(PURCHASE_ORDER_LINES, value);
  }
}
```

Request (Query) Objects

The `DataCriteria` object combined with filter and sorting objects allows clients to construct fairly complex query instructions for the data they are requesting.

Class	Description
DataCriteria	Encapsulates a request for one or more rows of data. The `DataCriteria` gives fine-grained control over query execution. Specify things such as sort order, fetch size, fetch starting point, filter criteria to restrict the result set, and more. Also selectively identify which attributes to return in the response if you don't want the entire object. • To specify a sort order, create and associate one or more `DataSortAttribute` objects with the `DataCriteria`. • To specify query filter criteria, create and associate a `Filter` object with the `DataCriteria`.
SortAttribute	Defines the sort properties of a single `DataServiceObject` attribute.

Filter	Encapsulates query criteria for a `DataCriteria` object. For example, create a `Filter` to achieve the following: `USERNAME = "LXLYONS"`. There are two types of filters: • **Expression Filter** -- Allows a client program to construct a simple or complex expression, including nested expressions using ANDs, ORs and expression operators EQUALS and GREATER_THAN. The service framework uses this expression to build a `WHERE` clause for the VO SQL statement. This is the recommended filter to create. There are three types of expression filters: • **Value Expression** -- Compares an attribute to a value. • **Attribute Expression** -- Compares two attributes. • **Child Expression** -- Filters the parent row based on child attribute values. • **Fixed Filter** -- A simple list of attributes. This is used when the view object must do custom processing of filter attributes and the client program should not be allowed to build nested and complex filter expressions.

As seen in Business Service Object Development, developers can generate one or more filter objects for each service view object. See the API documentation for additional examples that show how to use the query classes.

Domain Value Sets

An attribute of a data object might have a list of possible values. A domain value set is simply a view object that defines all the valid values for an attribute. For example, valid buyers for a purchase order is any active employee who has a procurement department job code. Former employees and people in the accounting department are outside the purchase order buyer domain, and therefore, invalid values. The buyer domain value set is a view object that contains all currently active employees who have a procurement job code.

Domain value sets are used to validate attributes while posting data objects. They can also be used to derive other attribute values based on some attribute values. For instance, when a foreign key is an internally generated surrogate key (buyer id), clients might need to use other alternate attributes (buyer name or e-mail address) to identify a foreign key, and domain value sets can be used to derive foreign key ids based on the alternate attributes, or vice versa.

As seen in Business Service Object Development, developers can generate domain value sets and map the attributes to the domain value sets by defining domain value set usages.

Dynamic Attribute Groups (DAGs)

There are many examples of business objects in Oracle E-Business Suite where the definition of a business object can vary based on attribute values supplied by each instance of a business object. These cases are referred to as business objects, which contain dynamic attributes. For example:

Business Object = Person

Person contains the following attributes: Name, Age, Country, Address.

If the attribute Country = USA, then the Address will include the City, State, and Zip Code attributes. However, if the attribute Country = Canada, then the Address will include the Province and Postal Code attributes.

In this example, the Address is referred to as a Dynamic Attribute Group (DAG) because it contains a set of attributes that vary based on the runtime values of the Country attribute. The attributes of the Address DAG are referred to as dynamic attributes. The service-oriented architecture (SOA) supports these dynamic attributes on a per row basis.

OA flexfields are used as the source of this dynamic definition information. There are two types of flexfields, key flexfields and descriptive flexfields. Key flexfields provide a flexible way for the Oracle E-Business Suite to represent objects such as accounting codes, part numbers, and so on. Descriptive flexfields provide a flexible way for Oracle E-Business Suite to provide customizable "expansion space" as well as a way to implement context-sensitive fields that appear only when needed.

If your table includes flexfields, make sure of the following:

* Key flexfield - It's corresponding entity object (EO) includes attributes for the CCID and Structure Number, as identified in the key flexfield's combinations table.

* Descriptive flexfield - It's corresponding EO includes attributes for all of the flexfield columns (AttributeCategory, Attribute1-n and so on).

You should then define a Dynamic Attribute Group for each flexfield defined in the table on the entity object.

Note: See Step 2: Define Entity Objects and Editing Service View Objects in the Business Service Objects Development document for more information about how to define flexfield dynamic attribute groups.

Description Objects

The description objects collectively let clients obtain detailed information about the service and goes beyond standard Java introspection. This feature enables automatic generation of XML schema, web services, and EJBs.

The description is available from the interface `oracle.svc.description.ServiceDescriptionProvider`, which defines the following two methods:

- `getDataObjectDescription(String qualifiedName)` -- Provides the ability to request descriptive information about the *data* managed by the service. (Provides more information than standard Java introspection).
- `getServiceDescription(String qualifiedName)` -- Provides the ability to request information about the *service itself*. (Provides more information than standard Java introspection).

The `ServiceDescriptionProvider` interface is implemented by the following classes:

- `oracle.jbo.server.ApplicationModuleServiceImpl`: The two `ServiceDescriptionProvider` methods are protected for use within the application module.
- `oracle.apps.fnd.rep.ws.IntegrationRepositoryService`: A public service used to describe integration repository metadata, including services.

See the API documentation for examples that show how to use these classes.

Note: The description objects are simple data documents that all subclass DataObject.

Class	Description
Description	The base description class that contains generic object information, such as its name and user-friendly display name. All other description objects subclass `Description`.
ServiceDescription	Describes the service including its methods, data objects, and data source descriptions. Getters return `MethodDescription`, `DataObjectDescription` and `DataSourceDescription` objects.
DataSourceDescription	Describes a data source including whether it is queryable, insertable, deleteable, and updateable when passed as a parameter to processDataSource and queryDataSource.
CriteriaDescription	Describes what kind of criteria can be applied to a data source, including the applicable filters, sortable attributes, and so on.
MethodDescription	Describes a method, including its parameters and return object.
DataObjectDescription	Describes a data object's attributes. Getters return `DataAttributeDescription` objects.
AttributeDescription	Describes any kind of attributes that might be referenced in the service framework, such as method parameters, filter/request criteria attributes, data attributes and so on.
KeySetDescription	Represents a key set, which is used to identify a record using a set of alternate key attributes when its internal primary key is *not* provided.
DomainValueSetDescription	Describes a domain value set. A domain value set defined on a data object (with its children) is uniquely identified by the domain value set name.
DynamicAttributeGroupDescription	Describes a dynamic attribute group and provides the metadata necessary for clients to interact with the dynamic attribute group. The metadata includes: the name of this dynamic attribute group through `Description.getName()`, the type of this dynamic attribute group through `getType()`, what attributes are contained in this dynamic attribute group through `getAttributeDescription()`, and the string description of this dynamic attribute group through `DataObjectDescription.getDocumentation()`.

Related Information

- Creating and Maintaining Services
- Javadoc:
  - oracle.svc.Service
  - oracle.svc.DataObject
  - oracle.svc.DataProcessorService
  - oracle.svc.DataSourceService
  - oracle.jbo.server.ApplicationModuleImpl
  - oracle.svc.ProcessControl
  - oracle.svc.MethodRequest
  - oracle.svc.MethodResponse
  - oracle.svc.BatchRequest
  - oracle.svc.BatchResponse
  - oracle.svc.QueryControl
  - oracle.svc.description.ServiceDescriptionProvider
  - oracle.svc.description.Description
  - oracle.svc.description.ServiceDescription
  - oracle.svc.description.DataSourceDescription
  - oracle.svc.description.CriteriaDescription
  - oracle.svc.description.MethodDescription
  - oracle.svc.description.DataObjectDescription
  - oracle.svc.description.AttributeDescription
  - oracle.svc.description.KeySetDescription
  - oracle.svc.description.DomainValueSetDescription
  - oracle.svc.description.DynamicAttributeGroupDescription
  - oracle.svc.DataCriteria
  - oracle.svc.SortAttribute
  - oracle.svc.ServiceMessage
  - oracle.svc.expression.ExpressionFilter

Business Service Object Methods in Detail

Overview

Contents

- queryDataSource
- processDataSource
- Various ServiceVO and AM Methods You can Override
- Supporting Attachments in a Service
- Retrieving Information from ServiceMessage
- Related Information

queryDataSource

`queryDataSource` is used to download data from a data source. This method can be used to implement custom methods that require data retrieval.

The queryDataSource() method uses SQL/X to fetch the records from the database and serializes the entire XML stream back to the client. In this process, any type conversions at the BC4J levels will be skipped. For example, if you have a table with VARCHAR columns, and corresponding VO attribute is of type Boolean, the response you get will contain the actual strings, instead of the type converted value.

Client Usage

Refer to the Using Services document for information on how to call the `queryDataSource` method.

Variations of the Method

None.

Overriding This Method

There are several hook points where developers can override the default behavior of querying from a data source, including:

```
oracle.jbo.server.ApplicationModuleServiceImpl.queryDataSource
  (String dataSourceName, DataCriteria dataCriteria, QueryControl queryControl)
```

Override this method in your ServiceImpl class if you need to change the overall behavior of querying data sources. The default behavior is to find the view object for the inbound data source, and call `getDataList` on the view object.

See `oracle.jbo.server.ApplicationModuleServiceImpl.queryDataSource` Javadoc for more information.

```
oracle.jbo.server.ViewObjectServiceImpl.getDataList
  (DataCriteria dataCriteria, QueryControl queryControl)
```

Override this method in your VOImpl class if you need to change how a view object or data source is queried. The default behavior is to execute a query based on the inbound data criteria, if it is not fetch only, and then construct a data object for each retrieved view row.

Note: The detail attributes are handled by:

```
oracle.jbo.server.ViewRowServiceImpl.getAttribute
  (int index, DataCriteria dataCriteria, QueryControl queryControl)
```

See `oracle.jbo.server.ViewObjectServiceImpl.getDataList` Javadoc for more information.

```
oracle.jbo.server.ViewRowServiceImpl.getAttribute
  (int index, DataCriteria dataCriteria, QueryControl queryControl)
```

Override this method in your VORowImpl class if you need to change how a detail view object is queried. The default behavior is to get the child's data criteria and recursively call `getDataList` on the child view object.

See `oracle.jbo.server.ViewRowServiceImpl.getAttribute` Javadoc for more information.

Custom Properties

None.

Filtering Parent Service View Objects Using Child Criteria

You can query a parent service view object (SVO) based on search criteria of its children. For example, suppose you want to get all purchase orders that have a line price greater than 100 US dollars. The query should return only those purchase orders that have a purchase order line with a line price greater than 100.

Declarative Implementation

To query a parent service view object using child criteria, you need to construct a filter on the parent service view object that references a filter on the child.

Step 1: Use the **Create Service View Object Editor** to create a new filter or update an existing filter on your **child SVO**.

Step 2: Use the **Create Service View Object Editor** to create a new filter or update an existing filter on your parent SVO. In the **Create Filter** panel of the **Create Service View Object Editor**, make sure the **Filter Type** is **Expression**.

Step 3: In the **Create Filter** panel of the **Create Service View Object Editor**, select the child attribute (either a view link attribute, a data object-type attribute, or a list-type attribute) from the **Available** list, and move it to the **Selected** list.

Step 4: In the **Child Filter Query** field, choose the child SVO filter that you created in Step 1.

Once you create the parent SVO expression filter, a client can then use that filter at runtime. Refer to the Using Services document for an example of how to use the filter.

Runtime Control

No runtime control steps are necessary to create a parent SVO expression filter.

processDataSource

`processDataSource` is used to upload data from a data source. This method can be used to implement custom methods that need to post data.

Client Usage

Refer to the Using Services document for information on how to call the `processDataSource` method.

Variations of the Method

None.

Overriding This Method

There are several hook points that developers can use to override the default behavior of uploading data to a data source, including:

```
oracle.jbo.server.ApplicationModuleServiceImpl.processDataSource
  (String dataSourceName, String operation, ProcessControl processControl,
  DataObject rootDataObject)
```

Override this method in your ServiceImpl class if you need to change the overall behavior of uploading data sources. The default behavior is to find the view object for the inbound data source and then call `processDataList` on the view object.

See `oracle.jbo.server.ApplicationModuleServiceImpl.processDataSource` Javadoc for more information.

```
oracle.jbo.server.ViewObjectServiceImpl.processDataList
  (RowSetIterator rsi, String oper, ProcessControl processControl, List dataList)
```

Override this method in your VOImpl class if you need to change how a list of data objects are uploaded to a view object or data source. The default logic is to go through each data object in the list and then call `processRow`.

See `oracle.jbo.server.ViewObjectServiceImpl.processDataList` Javadoc for more information.

```
oracle.jbo.server.ViewObjectServiceImpl.processRow
  (RowSetIterator rsi, ViewRowImpl currentRow, String oper, ProcessControl processControl,
  DataObject dataObject, DataObject retDataObject)
```

Override this method in your VOImpl class if you need to change how a data object is uploaded to a view object or data source. The default behavior is:

- If it is not a create operation, then call `findRow` to locate the server row.
- If it is a create operation, then create a new row.
- If it is a merge operation and `findRow` doesn't return a view row, then create a new row.
- After getting server view row, then:
  - If it is not a delete operation, call `ViewRow.setAttributes(Map attrs)` and then recursively handle each child.
  - If it is a delete operation, remove the child first and then remove itself.

See `oracle.jbo.server.ViewObjectServiceImpl.processRow` Javadoc for more information.

```
oracle.jbo.server.ViewObjectServiceImpl.findRow
    (ProcessControl control, RowSetIterator rsi, DataObject clientRow)
```

This method is used to locate a view row for an inbound data object. Override this method in your VOImpl class if you need to override the following default behavior:

- If the primary key is provided, the primary key values will be considered and the alternate key values will be ignored.
- If the primary key is not provided, then use the alternate key to locate the view row. The order of the key set is based on:
  - If the inbound process control contains `KeySet`, then use this order provided by the client.

    See `oracle.svc.ProcessControl.getKeySet` Javadoc for more information.

  - If the client doesn't provide the key set order, then use the pre-defined alternate key set order as described in Service View Object Wizard, Step 7: Alternate Keys.

```
oracle.jbo.server.ViewRowServiceImpl.setAttributes(Map attrs)
```

Override this method in your VORowImpl class if you need to change how the attributes are applied on a view row.

See `oracle.jbo.server.ViewRowServiceImpl.setAttributes` Javadoc for more information.

Custom Properties

As shown in the Extra Properties panel of Service Wizards.

See Viewing Custom Service View Properties and Viewing Custom Application Module Properties.

Enabling Partial Failures During Bulk Processing

Services can support partial failure during bulk processing of data. This is very useful when you load a large amount of data using batch load applications, as the occurrence of one or more failures would not prevent the continued posting of other unrelated data.

This feature is implemented by calling the `invokeServiceBatch` method on `oracle.svc.Service,` which has the inbound batch request's transaction mode set to `BULK_PROCESSING`, and uses the invoking method `processDataSource` on `oracle.svc.DataProcessorService`. The `invokeServiceBatch` method posts and commits successful data objects, and returns a batch response containing service messages for any failures that occur.

Logically, each top level data object in the inbound data list is processed separately and independently of each other. If a failure occurs, the following behavior results:

- In the case where the data processing involves a "remove" and an error occurs, then the current top level service data object (SDO) with its children is skipped, while the other top-level SDOs continue to be processed. For example, if the request is to remove five purchase orders, and one of the lines in the purchase order fails, then its corresponding purchase order is not removed, while the other 4 purchase orders are still removed if no other errors occur.
- In the case where the data processing involves an "insert", "update", or "merge", and an error occurs with a detail SDO, and its corresponding SVO or EO enables partial failure, then that detail SDO and its children are skipped, while the other valid detail SDOs are committed if no other errors occur. For example, if the request is to insert 3 purchase orders, and one shipment fails, and the shipment supports partial failure, then that shipment is not inserted, while the other objects are still inserted if no other errors occur.
- In the case where the data processing involves an "insert", "update", or "merge", and any other error occurs, then the current top level SDO with its children are skipped, while the other top-level SDOs are processed as long as no other errors occur. Consider the prior example of inserting three purchase orders where one shipment fails. If the shipment does not support partial failure, then its corresponding purchase order fails, while the other two purchase orders are inserted if no other errors occur.

Declarative Implementation

Follow these steps to enable partial failure support:

Step1: For each service view object (SVO) that is an insertable, updateable, mergeable, and/or deletable data source, or is a direct or indirect child of such a data source, you should decide whether it will support partial failure. While making your decision, take into consideration the following:

- From a functional perspective, does it make sense to simply skip that row if the row fails?
- A SVO should not have partial failure enabled if it does not meet the M72 and M73 model coding standards for partial failure.

Step 2: For each SVO that is to support partial failure, select the SVO in the **Navigator**, and using the context menu, open the **Service View Object Editor**. Select the **Extra Service Properties** node of the **Create Service View Object Editor**. Check the **Support Partial Failure** checkbox. Choose **Apply** and then **OK**.

Note: You can set this service property on either the service view object or on its entity object. The SVO, however, must have only one updateable EO if the SVO is to support partial failure. The custom property on the SVO overrides the property on the EO if it is set on both.

Various Service VO and AM Methods That You Can Override

This section provides information about the various service view object and application module methods that you can override. See the Javadoc for more information.

ApplicationModuleServiceImpl

```
protected BatchResponse invokeServiceBatchInternal(BatchRequest batchRequest)
```

This function is called by `invokeServiceBatch`, and all service method calls are routed internally through the `invokeServiceBatch()` method. You may override this method to customize the way invocation is handled.

ViewObjectServiceImpl

You can override the following functions to customize the query behavior:

```
protected void initQuery(DataCriteria dataCriteria, RowSet rs)
```

This function is called by `getDataList` when a query needs to be executed. It sets the `WHERE` clause, `order by`, and `bind` parameters on the row set according to `DataCriteria`. You can override this method to provide custom functionality to certain filters, or set and bind additional parameters on the row set before calling super to initialize the query.

```
protected void applyFilter(DataObject filter, StringBuffer predicate, List bindParams)
```

This function is called by `initQuery`. It constructs the `WHERE` clause based on the inbound filter and adds the bind parameters. You can override this function to construct a different `WHERE` clause and bind parameters.

```
protected SQLFilterExpression createSQLFilterExpression
  (DataObject filter, Expression expr, int startingBindParamIndex)
```

This function is called by `applyFilter`. For each nested/child expression filter, this function will be invoked once. It creates the SQL expression (sql fragment plus bind parameters) for an expression filter. You should override this method to create your own SQL expressions for proxy attributes. Each expression type can be handled separately.

```
protected void applyChildFilter
  (ViewAttributeDefImpl attrDef,DataObject filter, StringBuffer predicate, List bindParams)
```

This function is called by `applyFilter` if the inbound filter requires criteria on the children. For example, getting purchase orders that have lines with order quantity greater than 100. It constructs the predicate SQL statement out of the child filter object. You should override this method if a different behavior is required.

ViewRowServiceImpl

```
protected void initChildQuery
  (String childAccessorName, DataCriteria dataCriteria, RowSet rs)
```

This function is called by `getDataList` to retrieve the children and, by default, it delegates to `ViewObjectServiceImpl.initQuery`. You can override this method to bind parameters on the `RowSet` where these bind parameters are dependent on attributes on the parent row.

Supporting Attachments in a Service

Services can support Simple Object Access Protocol (SOAP) attachments and FND attachments. Keep in mind that these are different technologies designed for different purposes. A SOAP attachment defines how a document is attached to a web service message. The alternative to a SOAP attachment is to send a document inline with the web service body. In contrast, an FND attachment defines how the document is represented in the business object and how it is accessed from the database or other

document storage repository.

Supporting SOAP Attachments

A SOAP attachment is a compound document structure that consists of a primary SOAP message and zero or more secondary related documents. The primary message provides the process context for the compound SOAP structure as a whole, as well as for its secondary parts. SOAP attachments are used within web services and are *not* used for pure Java interfaces such as collocated Java interfaces or remote EJB interfaces.

By using a SOAP attachment, the web service SOAP message body is much smaller because it contains only a reference to the data and not the data itself. Using attachments can be more efficient because smaller SOAP message bodies are processed and validated by the XML parser more quickly than very large message bodies. For more detail about SOAP Attachments, refer to the W3C Group Note on the topic.

Declarative Implementation

Step 1: Go to the **Attributes** panel of the **Service View Object Wizard** or **Service View Object Editor** for the SVO that you want to enable for SOAP attachments. Select **New** to define a new attribute in the **Attribute Settings** panel. For the new attribute, set the **Type** poplist to either **CLOB** or **BLOB**.

Step 2: If you check the **SOAP Attachment** checkbox, you must also specify a **MIME Type**. You may enter a static value (for example, `text/html`) or an expression if the MIME type changes for each service data object instance (`${svc.current.mimeAttributeName}` where `mimeAttributeName` is the name of the SDO attribute containing the MIME type for this instance).

If you choose not to check the **SOAP Attachment** checkbox, the BLOB or CLOB attribute is handled inline within the SDO. Although this is fine for small documents, for very large documents, the **SOAP Attachment** checkbox should be checked.

Step 3: For information on other standard properties that you can specify in the Attribute Settings panel, refer to the discussion of the Attribute Settings panel in the Creating and Maintaining Services document.

Supporting FND Attachments

You can enable FND attachments in a service to allow service clients to retrieve and upload attached documents.

There are five types of FND attachments:

- URL attachment - a URL string, stored in the FND document catalog.
- Text attachment - short text (no longer than 200 characters), stored in the FND document catalog.
- Folder attachment - folder in Oracle Files.
- Blob document attachment - binary document, stored in the FND document catalog.
- Oracle File attachment - a file stored in Oracle File.

Note: Folder attachments and Oracle File attachments are not currently supported.

Note: SOAP attachments are always implicitly used for FND Attachments of type BLOB. FND Attachments of type URL and Text are always handled inline in the SDO document as strings.

Attachments are retrieved and posted as `oracle.apps.fnd.framework.svc.AttachedDocument` data objects. An attached document is treated as a detail of its parent view object. Services use the method `queryDataSource("SvcServiceDescriptions")` on `oracle.svc.DataSourceService` to return the description that includes the attachments. On a high level, the method `queryDataSource` returns the attached documents along with its parent service data objects (SDOs) and the method `processDataSource` on `oracle.svc.DataProcessorService` uploads the attachments along with its parent SDOs. Attachments can be filtered via the `queryDataSource` method using the `oracle.apps.fnd.framework.svc.AttachedDocumentFilter`.

Each `AttachedDocument` data object has the following attributes:

Attribute Name	Data Type	Description
AttachmentId	Number	Attachment ID, which is the primary key.
DocumentId	Number	Identifier of the document in the FND document catalog, or the identifier of the file in Oracle Files. See Note.
RepositoryName	String	Name of the file repository. Used only if the attached document is a file in Oracle Files. See Note.
RepositoryDocumentPath	String	Path of the file in the Oracle Files repository, excluding the file name. Used only if the attached document is a file in Oracle Files. See Note.

Name	String	Name of the attached document. Not used if the attachment is a URL.
Version	String	Version of the file in Oracle Files. Used only if the attachment is a file in Oracle Files. Read-only. See Note.
Description	String	Description of the attachment.
SeqNum	Number	Sequence number to determine the order of the attachment.
CategoryName	String	Category to which the document belongs.
UsageType	String	Standard, Template, or Onetime. Read-only.
ContentType	String	URL, TXT, BLOB, or FOLDER. See Note.
ContentURL	String	URL string that identifies the content for a URL attachment. Valid only if ContentType=URL.
ContentBlob	Blob	Content for a blob attachment. Valid only if ContentType=BLOB.
ContentText	String	Content for a text attachment. Valid only if ContentType=TXT.
MimeType	String	Mime type for blob attachment. Valid only if ContentType=BLOB.
LastUpdateDate	Date	Date when the attachment was last updated. Read-only.
LastUpdateBy	String	Name of the last user who updated the attachment. Read-only.

Declarative Implementation

Before you decide whether a service will support FND attachments, you should first understand the concept of an entity, as each attached document has to be linked to an entity. An entity is an object within Oracle E-Business Suite data, such as an item, a purchase order, or an order line. The attachments feature must be enabled for an entity before users can link attachments to the entity.

Important: Entities are not related to BC4J Entity Objects.

An entity consists of a set of view attribute names, which are the primary keys for the view object related to your region. When an attachment is stored, the values of these primary keys and the Entity ID are stored so the attachments can be uniquely retrieved. Refer to the Attachments topic in Chapter 4: Implementing Specific UI Features, for additional information about attachments and entities.

Once you decide which service view object will support attachments, follow the steps below to enable attachments for that service:

Step 1: Go to the **Attributes** panel of the **Service View Object Wizard** or **Service View Object Editor** for the SVO that you want to enable for attachments. Select **New** to define a new attribute in the **Attribute Settings** panel. For the new attribute, set the **Type** poplist to **Attachment**.

Step 2: Select **Add** under the **EntityId** poplist to link the attachment attribute to an entity. An attachment attribute can link to multiple entities, but each entity must have its own unique name. For example, a purchase order can have attachments for buyers and suppliers. You can choose to define a single attachment attribute that links to two separate entities, buyers and suppliers, or define two separate attachment attributes, where one links to the buyer entity and the other links to the supplier entity.

Note: As a standard, if you define only one attachment attribute, name it "Attachments". If you define more than one, append "Attachments" to their names, such as "BuyerAttachments" and "SupplierAttachments".

You can link to an entity already defined in Oracle E-Business Suite or link to an entity that you define in the service view object itself.

- To link to an entity already defined in Oracle E-Business Suite, enter the **Entity ID**, as defined in Oracle E-Business Suite. The wizard uses the specified Entity ID to retrieve the primary key of the entity from the database and displays it in the **Selected** list.
- To link to a new entity in the service view object, enter a unique name in the **Entity ID** field. In the **Entity Primary Keys Available** list, select a primary key and shuttle it to the **Selected** list.

Step 3: Specify the category of documents that you want to associate with the attachment. Document categories provide security by restricting the documents that service clients can view, add, or delete as an attachment. Service clients can only add, edit, or view documents within the specified categories. Use the **Categories** shuttle to select the categories you want to associate with your attachment. Select **OK** when you are done.

Note: A "Miscellaneous" category is seeded to provide easy visibility of a document across products.

Step 4: You can choose to secure the attachment attribute that you are defining in the service view object, by a particular context. If you specify a security context for an attachment attribute, then any attachment that a user adds is automatically secured to that specified context. For example, if you set the security context for the attachment attribute of this service to Organization and the document was created by a user in Organization ABC, then only users in Organization ABC will be able to access the attachments using this service.

To set the security context, select the **Edit** button to the right of the **Security** field. Specify a **Security Type,** then specify the attribute that contains the security value using the **Security Attr** poplist. Select **OK** when you are done.

Note: If you do not provide a value for **Security Attr** and **Security Type** is not **None**, then the corresponding value in the user's current context for the organization id, set of books id, or business group id is used.

Step 5: For information on other standard properties that you can specify in the Attribute Settings panel, refer to the discussion of the Attribute Settings panel in the Creating and Maintaining Services document.

Retrieving Information from ServiceMessage

The `oracle.svc.ServiceMessage` data object is used by the Service Interface engine to return information about errors and warnings when a client calls `Service.invokeServiceBatch` to invoke methods on a service.

When an exception is thrown in a service, the Service Interface engine converts it into a `ServiceMessage`. If the thrown `oracle.jbo.JboException` or `oracle.apps.fnd.framework.OAException` is a bundled exception, the Service Interface engine creates a list of `ServiceMessages` that correspond to the list of peer exceptions in the bundle. The details of a `JboException` or `OAException` are also converted into detail `ServiceMessages`.

The Service Interface engine returns a `ServiceMessage` data object at the method invocation level as part of `oracle.svc.MethodResponse`, and at the batch level as part of `oracle.svc.BatchResponse`. The convenience methods `findAllMessage` and `findAllErrorMessage` on `BatchResponse` allows you to get all messages returned at the method invocation level as well as at the batch level.

The following table summarizes a few of the key methods on `ServiceMessage` that you can use to retrieve information about Service error message. Refer to the `ServiceMessage` Javadoc for additional information.

Method	Description
`getCode`	This method returns a string message code that identifies the error or warning. This corresponds to the message code of the thrown `OAException`.
`getType`	This method returns one of the following message types that corresponds to the type of the `OAException` thrown: `TYPE_SEVERE`, `TYPE_ERROR`, `TYPE_WARNING`, and `TYPE_CONFIRMATION`.
`getText`	This method returns the translated text of the message that corresponds to the translated message text of the thrown `OAException`.
`getDetail`	If a `ServiceMessage` contains other detail `ServiceMessages`, this method returns those detail messages to provide further information to help explain the problem.
`getDataSourceName`	If a `ServiceMessage` is thrown while processing or getting a certain data source, this method returns that data source name. This method returns `null` if the message does not belong to a data object.
`getDataObject`	If a `ServiceMessage` represents a message for a certain data object, this method may return the actual fully populated data object or an instance of the data object with just the primary key set. This method returns `null` if the message does not belong to a data object.
`getDataObjectQualifiedName`	If a `ServiceMessage` represents a message for a certain data object, this method returns the data object qualified name. This method returns `null` if the message does not belong to a data object.
`getDataObjectAttributeName`	If a `ServiceMessage` represents a message for a certain data object, where specifically, it represents an error for a certain attribute on the data object, this method returns the attribute name. This method returns `null` if the message does not belong to a data object.
`getDataObjectAttributeValue`	If a `ServiceMessage` represents a message for a certain data object, where specifically, it represents an error for a certain attribute on the data object, this method returns the value of that attribute that caused the error. This method returns `null` if the message does not belong to a data object.
`getPath`	If a `ServiceMessage` represents a message for a certain data object, this method returns an XPath (xml path) string representation of the path to the data object that caused the error. For example, if an error occurred in the second Purchase Order Line of the first Purchase Order record, the following XPath string would be returned: `/PurchaseOrders.0/PurchaseOrderLines.1`

`getPathElement`	If a `ServiceMessage` represents a message for a certain data object, this method returns a list of `oracle.svc.ServiceMessage.PathElement` that represents the path to the data object that caused the error. The PathElement data object is an object representation of the XPath string returned by `getPath` and provides the element name and index for each segment in the path. For example, if an error occurred in the second Purchase Order Line of the first Purchase Order record, this method would return a list of two PathElements: the element name PurchaseOrder, whose index is 0 and the element name PurchaseOrderLines whose index is 1.

Related Information

- Business Service Object Development
- Using Services
- OA Framework Model Coding Standards
- Javadoc
  - `oracle.jbo.server.ApplicationModuleServiceImpl`
  - `oracle.jbo.server.ViewObjectServiceImpl`
  - `oracle.jbo.server.ViewRowServiceImpl`
  - `oracle.svc.DataSourceService`
  - `oracle.svc.DataProcessorService`
  - `oracle.svc.Service`
  - `oracle.svc.xml.XMLUtil`
  - `oracle.svc.BatchRequest`
  - `oracle.svc.DataCriteria`
  - `oracle.svc.QueryControl`
  - `oracle.svc.ProcessControl`
  - `oracle.apps.fnd.framework.svc.AttachedDocument`
  - `oracle.apps.fnd.framework.svc.AttachedDocumentFilter`
  - `oracle.svc.ServiceMessage`
  - `oracle.jbo.JboException`
  - `oracle.apps.fnd.framework.OAException`
  - `oracle.svc.MethodResponse`
  - `oracle.svc.BatchResponse`
  - `oracle.svc.ServiceMessage.PathElement`

Oracle Application Web Services

Overview

This document discusses the various services that are related to Oracle Application Framework (OAF):

Contents:

- EbizHomepageService
- OAAttachmentsService

EbizHomepageService

The EbizHomepage Service, which exposes the data on the E-Business Suite homepage, is developed using the 12.1 Business Service Object infrastructure. Please refer the "Introduction to SOA and Services" section for an overview of the SOA concepts.

Service Description

Using this service you can retrieve E - Business suite user's functions and favorites information as well as encrypted function URL by providing the function name. This service is intended for exposing the E – Business suite homepage content as a service.

Methods:

EbizHomepageService consist of the following services:

- getFunctions( )
- getFavorites( )
- getFunctionUrl( )

Service Data Objects

EbizHomepageService consist of the following Service Data Objects:

- Function
- Favorite

Methods

- List getFunctions( )

  This method returns the E-Business Suite user's function information for the user invoking a service.

  Parameters

  None.

  Return Type

  List of "Function" Service Data Objects.

- List getFavorites( )

  This method returns E-Business Suite user's favorites information for the user invoking a service.

  Parameters

  None.

  Return Type

  List of "Favorite" Service Data Objects.

- String getFunctionUrl (String funcName)

  This method returns encrypted function URL provided by the valid E-Business Suite form function name.

  Parameters

Parameter Name	Data Type	Description
FuncName	java.lang.String	E-Business Suite form function name.

Return Type

Encrypted function url of data type java.lang.String. returns the encrypted url of the function only if the user has access to it, otherwise it will throw an API OAException.

Service Data Objects

- Function - This service data object contains E-Business suite form function information with 4 String attributes.

  Attributes

Attribute Name	Data Type	Description
FunctionName	java.lang.String	Name of the function.
FunctionUrl	java.lang.String	Encrypted function url.
RespName	java.lang.String	Name of the Responsibility corresponds to FunctionName.
FunctionType	java.lang.String	Type of the function e.g SSWA jsp function.

- Favorite - This service data object contains E-Business suite form function information with 3 String attributes

  Attributes

Attribute Name	Data Type	Description
FavoriteName	java.lang.String	Name of the favorite.
FavoriteUrl	java.lang.String	Encrypted favorite url.
FavoriteType	java.lang.String	Type of the favorites function e.g SSWA jsp function.

OAAttachmentsService

The OAAttachments Service, developed using the 12.1 Business Service Object infrastructure, allows users to upload FND attachments in bulk. Please refer the attachments section for additional information on this service.

Related Documents

- Invoking service from BPEL
- Business Service Objects (BSO)

Related Information

- Creating and Maintaining Services
- Javadoc:
  - `oracle.svc.Service`
  - `oracle.svc.DataObject`
  - `oracle.svc.DataProcessorService`
  - `oracle.svc.DataSourceService`

- oracle.jbo.server.ApplicationModuleImpl
- oracle.svc.ProcessControl
- oracle.svc.MethodRequest
- oracle.svc.MethodResponse
- oracle.svc.BatchRequest
- oracle.svc.BatchResponse
- oracle.svc.QueryControl
- oracle.svc.description.ServiceDescriptionProvider
- oracle.svc.description.Description
- oracle.svc.description.ServiceDescription
- oracle.svc.description.DataSourceDescription
- oracle.svc.description.CriteriaDescription
- oracle.svc.description.MethodDescription
- oracle.svc.description.DataObjectDescription
- oracle.svc.description.AttributeDescription
- oracle.svc.description.KeySetDescription
- oracle.svc.description.DomainValueSetDescription
- oracle.svc.description.DynamicAttributeGroupDescription
- oracle.svc.DataCriteria
- oracle.svc.SortAttribute
- oracle.svc.ServiceMessage
- oracle.svc.expression.ExpressionFilter

View Objects in Detail

Overview

This document discusses view objects in detail. For more advanced view object discussions, you should also refer to Advanced View Object Development Topics.

- View Object Attribute Types and Caching
- Initialization Guidelines
  - Preparation for Inserts
    - View Objects with No Database Query
    - View Objects that Perform Database Queries
  - Query Execution
    - Avoiding Unconditional Initialization
    - Preventing Redundant Queries with Tables
  - Examples
  - Known Issues
- View Object Query Criteria
  - Where Clause and Where Clause Parameters

View Object Attribute Types and Caching

In general, a view object row is maintained by a view object cache, whereas an entity object is maintained by an entity cache and is shared by view objects using the same entity object definition. Each view object can have multiple row sets. Usually, when you use methods in `oracle.jbo.server.ViewObjectImpl`, such as `getRowAtRangeIndex`, BC4J uses the default row set. You can optionally create a user-defined row set, although we do not encourage this.

In a master/detail case, where you use a view link to manage the master/detail relationship, BC4J exposes an accessor method on the master view object row that allows you to get a detail view object row set. These row sets are owned by an internal instance of a detail view object by BC4J. Each row set can have multiple row set iterators. Once again , when you navigate between rows, BC4J uses a default row set iterator, but you can always create one with the `oracle.jbo.server.ViewObjectImpl.createRowSetIterator` method.

View objects may include four different types of attributes as outlined in the following table:

View Attribute Type	Description / Example	Mapped To	Changes Persist After Query?
Entity-Derived Persistent	Any attribute based on a persistent entity object attribute.	Entity object attribute which is, in turn, mapped to a database column	Yes
Entity-Derived Transient	Any attribute based on a transient entity object attribute. For example, a calculated purchase order total value maintained on the entity object, but not stored in the database.	Transient entity object attribute	Yes
SQL-Derived	Any queried (calculated) value not based on an entity object attribute.	SQL SELECT	No
Transient / Dynamic	Attributes without any datasource. Example: a "SelectFlag" attribute added to view object for single or multiple selection in a corresponding table.	Nothing (values assigned programmatically)	No

View object attribute data is stored based on this mapping:

- Entity-derived view object attribute values are stored in a special entity object cache that persists for the life of the *transaction*.
- SQL-derived and transient / dynamic attribute values are stored on the view object row, which is cached for the life of the *rowset*.

This storage scheme has important implications for runtime behavior. For example, in a master-detail relationship, SQL-derived and transient / dynamic attribute values on the detail view object persist *only* while its master row is current. As the user navigates from master row to master row, and the rowset is refreshed, pending changes to SQL-derived and

transient / dynamic attributes values on the associated detail rows are lost. Changes to entity-derived attributes, however, are preserved.

See Java Entity Objects for additional information about the entity cache implications for the different view object attribute types. See the Entity Object and View Object Attribute Setters for detailed information about setting values for these different attribute types.

Initialization Guidelines

Note: Supporting the Browser Back Button, the OA Framework Model Coding Standards and Controller Coding Standards reference these consolidated instructions to avoid a fragmented picture of view object-specific initialization rules and behavior.

In order to comply with various standards outlined in these referencing documents, it is important to take care when initializing view objects. For the purposes of this document, "initialization operations" include:

- Preparing the view object for inserts (see the Preparation for Inserts section for details).
- Executing queries (see the Query Execution section for details).

Preparation for Inserts

To programmatically insert rows into your view objects, follow these instructions to properly prepare them based on whether they do -- or do not -- query from the database.

View Objects with No Database Query

Before inserting rows into a view object that does *not* query data from the database, you *must* initialize it with a call to `setMaxFetchSize(0)` to suppress any efforts by BC4J to execute a query on your behalf.

Note: Always verify that the view object doesn't have any rows before you perform this initialization.

The following application module logic shows how to add a single row to a view object to which additional rows might be added later. For example, your `processFormRequest()` "Add Another Row" table button handler code might delegate to this to properly insert the first of (potentially) many rows in the table's view object.

```
/*
** This logic should be added to an application module method which you
** would then invoke from your controller.
*/

// Check to see that the view object doesn't have any rows. We only want
// to perform this initialization before we insert the first row.
if (vo.getFetchedRowCount() == 0)
{
  // We want to initialize the VO with the call to setMaxFetchSize()
  // only before we add the first row.  If we add subsequent rows,
  // we don't want to call this again, which is way it's enclosed
  // in the IF check.
  vo.setMaxFetchSize(0);
}

// Perform insert
Row row = vo.createRow();
vo.insertRow(row);

// If this view object is transactional, you must also comply with
// Model Coding Standard M69 by setting the new row state to
// STATUS_INITIALIZED.  Note that there is no harm in including
// this line of code in non-transactional view objects if you
// find it easier to follow a single coding pattern for all your
// view objects.
row.setNewRowState(Row.STATUS_INITIALIZED);
```

Note: For the **Add Another Row** table button to respond to the form submit event when the button is clicked, the `if` clause before the `createRow()` and `insertRow()` calls above *must* also be present in your controller's `processRequest()` method. This ensures that the view object is prepared to carry out the add operation for the table.

To insert all of your rows in a single call, such as to manually populate the contents of a poplist, modify the above code to look like the following. In this case, all of your inserts are handled at once during `processRequest()`, so you don't need to worry about checking this initialization logic again to make additional inserts later.

```
/*
```

```
** This logic should be added to an application module method which you
** would then invoke from your controller.
*/

if (vo.getFetchedRowCount() == 0)
{
  vo.setMaxFetchSize(0);

  // Insert as many rows as you want.
  // You could insert multiple rows with a FOR loop here.
  row = vo.createRow();
  vo.insertRow(row);
}
```

Note: `getFetchedRowCount()` returns the number of rows cached in the view object, including programmatically inserted rows. The `getRowCount()`, `getEstimatedRowCount()`, and `hasNext()` methods are *not* appropriate alternatives because they implicitly execute a query against the database (if a query has not been executed yet). You also cannot use the `isPreparedForExecution()` method to check the state of the view object because the flag it checks is not reset after a rollback. However, you still need to reinitialize the view object after a rollback because BC4J clears all the rows that you inserted.

If this transient view object includes mutable data, and you might issue a commit in the transaction in which you are maintaining the data for this view object (an application properties VO is a good use case example of this), you need to add a call to `vo.executeQuery()` after you call `setMaxFetchSize(0)`. This small workaround for a known BC4J issue ensures that your rows are not inadvertently cleared after a commit.

For example:

```
/*
** This logic should be added to an application module method which you
** would then invoke from your controller.
*/

if (vo.getFetchedRowCount() == 0)
{
  vo.setMaxFetchSize(0);
  vo.executeQuery();

  ... // Insert rows into the view object.
}
```

As a rule, do *not* use the false SQL predicate `WHERE 1=0` to try to suppress query execution as this still issues a redundant database query. However, there is one exception to this rule: for detail view objects in a master - detail relationship, use the false SQL predicate `WHERE 1=0` and call `executeQuery()` before inserting new rows into the detail view object. Calling `setMaxFetchSize(0)` has an adverse effect in this case. For example:

```
/*
** This logic should be added to an application module method which you
** would then invoke from your controller.
*/

if (!detailVO.isPreparedForExecution())
{
  // The detail VO query has not been executed yet, so
  // set the where clause to 1=0 if you do not want any
  // detail rows to be queried from the database.
  detailVO.setWhereClause("1=0");

  // Execute the detail VO query before inserting rows.
  detailVO.executeQuery();
}

Row row = detailVO.createRow();
detailVO.insertRow(row);

// If this view object is transactional, you must also comply with
// Model Coding Standard M69 by setting the new row state to
// STATUS_INITIALIZED.  Note that there is no harm in including
// this line of code in non-transactional view objects if you
// find it easier to follow a single coding pattern for all your
// view objects.
row.setNewRowState(Row.STATUS_INITIALIZED);
```

```
...
```

Note: For the **Add Another Row** table button to respond to the form submit event when the button is clicked, the `if` clause before the `createRow()` and `insertRow()` calls above *must* also be present in your controller's `processRequest()` method. This ensures that the view object is prepared to carry out the add operation for the table.

View Objects that Perform Database Queries

In any view object that is used both for queries and inserts, you *must* initialize your view object with an `executeQuery()` call *before* you call `insertRow(row)`.

Note: You can execute the query directly with a call to `executeQuery()` or you can use the `queryData()` method for tables as described in the Preventing Redundant Queries w/ Tables section below. These calls *must* be made before you insert a row.

Query Execution

When executing view object queries, there are two primary considerations for you address: unconditional initialization and redundant queries with `OATableBean` / `OAAdvancedTableBean` regions.

Avoiding Unconditional Initialization

As described in Supporting the Browser Back Button, it is important that you avoid unconditional view object query execution in the context of a `processRequest()` method since the OA Framework calls this method when it needs to synchronize the web bean hierarchy after Back button navigation, and after display a dialog page.

When the `processRequest()` method is reentered, unconditional initialization could reset an undesirable loss of transaction state. For example, user updates made to transient view object attributes will be lost, and the the view object current row and range will be reset.

To avoid this, add an `OAViewObject.isPreparedForExecution()` check before executing your query. For example:

```
/*
** This logic should be added to an application module method, which you would then
** invoke in your controller's processRequest() method.
*/

if (!vo.isPreparedForExecution())
{
  vo.executeQuery();

  // Or call vo.initQuery(...) if you have an initQuery method
  // you want to use instead to set a new WHERE clause or bind
  // query criteria.
}
```

Tip: If, for some reason, it is essential that you re-query a "view-only" page every time it is rendered, you don't need to perform this check. This is required only when the subsequent querying is unnecessary, and you are not concerned about the loss of view object state. For example, if you have a product Home page with small "At a Glance" summary tables and always want to show the latest data when the page renders, you can skip this check. Just remember that all view object state will be reset.

These instructions also do *not* apply to view object query execution in the context of a `processFormRequest()` method. For queries initiated within `processFormRequest()` in response to form submit events, such as the user's selection of the **Go** button in a Search page, you should explicitly refresh the data from the database by making the `executeQuery()` call directly.

Note: The `isPreparedForExecution()` method checks whether a view object is properly prepared for query execution. "Properly prepared for query execution" means that the view object's WHERE clause parameters are bound to appropriate values. As long as the WHERE clause parameters are properly bound, the OA Framework can safely execute a fetch. Calls to `executeQuery()`, `setMaxFetchSize(0)`, and `setPreparedForExecution(true)` all mark a view object as being "prepared" for query execution.

Warning: In some corner cases, you cannot reliably use the `isPreparedForExecution()` check as described above.Consider the following scenario that commonly occurs with an HGrid view object: your controller invokes an application module method that issues a `rollback()` or `clearCache()` to reset the state of a queried view object. Then within the same request -- and before the next page is processed for JSP forward case -- you immediately insert one or more rows into the view object. In this case, to avoid an unintended loss of data, substitute the `isPreparedForExecution()` check with an `OAViewObject.isExecuted()` call.

Special Case: Drilldown Target Page

When you navigate to a page that you expect to query each time you explicitly visit it, it is not sufficient to rely on the `isPreparedForExecution()` test since this could easily return `true` when you really should be executing a fresh query.

If your drilldown target page displays a single row of data -- and you will be querying using a primary key value -- you may use the view object `findByKey()` method as shown below. When you use this method, BC4J first checks its cache for a row match and if it fails to find one, it executes a query from the database.

Tip: If you want to use this method, the drilldown target page should use a different view object instance from the search page so you don't disturb the state of the search results region.

For example:

```
/*
*******************************************************************************
* Initializes the detail employee query.
*******************************************************************************
*/
public void initDetails(Number employeeNumber)
{
  EmployeeFullVOImpl vo = getEmployeeFullVO1();

  // Since we're querying a single row with a primary key value, we
  // can use the findByKey method which first checks the BC4J cache,
  // and if it doesn't find a matching row, it will formulate a query using
  // the given key and retrieve it from the database.
  Number[] keys = { employeeNumber };
  Row[] rows = vo.findByKey(new Key(keys), 1);

  // You must set the current row, or the Details page won't display any
  // data. When you explicitly query data, you don't have to do this.
  if ((rows != null) && (rows.length > 0))
  {
    vo.setCurrentRow(rows[0]);
  }
} // end initDetails()
```

If your drilldown target page involves querying a table instead of a single row (this applies only to the top-level entity in the page; you can still use `findByKey()` to query a master-detail where the details are displayed in a table), you should explicitly compare your incoming search parameters with those used in the previous query to ascertain whether a new query is actually required.

For example, create a utility method like the one shown below to check the value of the **ename** parameter when your view object select statement is `select empno, ename, sal, deptno from emp where ename = :1`.

```
// The following test is in your view object's initQuery method
if (isSearchCriteriaNew(enamParam))
{
  // Do whatever WHERE clause preparation you need to do...

  executeQuery();
}

private boolean isSearchCriteriaNew(String ename)
{
  boolean isNew = true;

  // Get the ename parameter currently stored in the VO.
  Object[] parameters = getWhereClauseParams();

  // If there are no parameters, this is the first
  // query so we want to return true.
  if (parameters.length > 0)
  {
    // BC4J always returns these as Strings
    String enameParam = (String)parameters[0].toString();

    if (ename.compareTo(enameParam) == 0))
    {
      isNew = false;
    }
```

```
  }

  return isNew;
}
```

Preventing Redundant Queries w/ Tables

Note: The following instructions apply regardless of whether your table's query is executed in the scope of `processRequest()` or `processFormRequest()`.

OA Framework requires an event point for applying personalization changes to a view object and as a consequence, *always* executes the query against a table's view object when it gains control after your code executes. To avoid executing the same `OATableBean(table)` and `OAAdvancedTableBean(advancedTable)` query twice (once yourself, and later by OA Framework), designate the view object for querying and let OA Framework perform the query.

Note: You do not need to worry about this for tables queried automatically via a view link, or for tables queried automatically in an `OAQueryBean` region.

To achieve this, follow these instructions:

Step 1: For view objects used with tables, implement your `OAViewObjectImpl's initQuery()` method to include a `Boolean` query indicator flag (`executeQuery`). When this value is set to **true**, explicitly execute the query. When the value is set to **false**, allow OA Framework to do this on your behalf.

Tip: Implementing your view object's `initQuery()` method with this flag allows you use this view object outside the context of a table/advancedTable since you won't have OA Framework to implement the query for you in these other scenarios.

For example:

```
public void initQuery(..., Boolean executeQuery)
{
  // Set the WHERE clause and/or bind parameters as needed.
  ...

  // Check the executeQuery parameter and explicitly execute
  // the query if needed.
  if ((executeQuery != null) && (executeQuery.booleanValue()))
  {
    executeQuery();
  }
}
```

You also need a corresponding method to invoke in the application module that delegates to the view object's `initQuery()` method.

For example:

```
public void initXXXVOQuery(..., Boolean executeQuery)
{
  ViewObject vo = getXXXVO();

  // The preparedForExecution check is needed when invoked from processRequest()
  // to avoid unconditional view object initialization -- should be skipped
  // when invoked from processFormRequest() for a user-initiated search.
  if (vo.isPreparedForExecution())
  {
    vo.initQuery(..., executeQuery);
  }
}
```

When the `initQuery()` method is used for both initialization in `processRequest()` and a user-initiated search in `processFormRequest()`, conditionally perform the `preparedForExecution` check using an additional flag, `checkForExecuted` as in Step 2 below, in the shared view object `initQuery()` method as follows. Have the application module delegate to the view object's `initQuery()` method without performing the `preparedForExecution` check, or create two separate methods in the application module instead.

```
public void initQuery(..., Boolean checkForExecuted, Boolean executeQuery)
{
  if ((checkForExecuted != null) && (checkForExecuted.booleanValue()) &&
      (isPreparedForExecution()))
  {
    return;
```

```
  }

  // Set the WHERE clause and/or bind parameters as needed.
  // Per Model Coding Standard M37, we should set up the query criteria only if
  // the query really needs to be executed -- observing this coding standard
  // promotes view object state consistency and avoids redundant query criteria
  // setup and undesired side effects.
  ...

  // Check the executeQuery parameter and explicitly execute
  // the query if needed.
  if ((executeQuery != null) && (executeQuery.booleanValue()))
  {
    executeQuery();
  }
}
```

Step 2: Implement controller logic to call the `initQuery()` method followed by a call to your table's `queryData()` method. In addition to a `pageContext` object, the `queryData()` method takes a `boolean` parameter, `checkForExecuted`, indicating whether you want OA Framework to check the `preparedForExecution` flag before actually executing the query.

- Pass `true` to ensure that the query is *not* re-executed if it has already been performed. This is important if you don't want to inadvertently reset view object state after it is initialized. For example, you might use this if you have an updateable table that queries once when the page renders, but not again after this point.
- Pass `false` to always re-execute the query regardless of the view object's current state.

For example, the following code illustrates the correct way to invoke a user-initiated query related to a table. In this case, we don't check the `preparedForExecution` flag because we *always* want to execute a fresh query when the user initiates a search, such as pressing a **Go** button.

```
import oracle.bali.share.util.BooleanUtils;
...

public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormRequest(pageContext, webBean);

  Boolean executeQuery = BooleanUtils.getBoolean(false);
  Serializable[] methodParams = { ..., executeQuery };
  Class[] methodParamTypes = { ..., executeQuery.getClass() };

  // We call an initXXXVOQuery() method on the application module that
  // delegates to the view object's initQuery().  You should not access
  // view objects directly in your controller.
  OAApplicationModule am = ...;
  am.invokeMethod("initXXXVOQuery", methodParams, methodParamTypes);

  // This tells the OA Framework to execute the query for the table.
  // Note that we pass "false" to this call because we don't want the OA
  // Framework to check the prepared for execution state; we always want
  // to perform the query in response to a user initiated search.

  // OATableBean tableBean = ...
  // OAAdvancedTableBean tableBean = ...
  tableBean.queryData(pageContext, false);
}
```

To reuse the same `initQuery()` method in non-table contexts, set the `executeQuery` parameter to **true** so your view object actually queries itself. For example:

```
import oracle.bali.share.util.BooleanUtils;

public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  Boolean executeQuery = BooleanUtils.getBoolean(true);
  Serializable[] methodParams = { ..., executeQuery };
  Class[] methodParamTypes = { ..., executeQuery.getClass() };

  // We call an initXXXVOQuery() method on the application module that
```

```
  // delegates to the view object's initQuery().  You should not access
  // view objects directly in your controller.
  OAApplicationModule am = ...;
  am.invokeMethod("initXXXVOQuery", methodParams, methodParamTypes);
}
```

The following matrix shows you the general guideline on setting the `checkForExecuted` and `executeQuery` flags mentioned in the above steps. There are exception cases like a product Home page with small view-only "At a Glance" summary tables where we may *always* want to show the latest data when the page renders.

Note: `checkForExecuted` would be `false` in `processRequest()` for this exception case.

	Query in processFormRequest() (in response to a user-initiated search)	**Query in processRequest() (to initialize data)**
Non-Table-Bean View Object	checkForExecuted = false executeQuery = true	checkForExecuted = true executeQuery = true
Table Bean View Object	checkForExecuted = false executeQuery = false	checkForExecuted = true executeQuery = false

Examples

The following examples illustrates the practical and combined use of the rules that have so far been discussed.

Use Case #1: Search Page w/ Drilldown to Details

This example has a search page with a results table. The user enters search criteria and then selects a **Go** button to execute the table's query. Once results are queried, the user can select a link to drill down to a details page that queries using the primary key from the selected row in the Search page.

Search Page

Question	**Answer**
Are we inserting any rows?	No
Are we querying data?	Yes
Do we perform any `processRequest()` initializations?	No. We query only in response to a user-initiated event.
Do we query data for a table?	Yes

In this case, we need to be concerned only with avoiding a redundant query on the table as described above.

Drilldown to Details Page

Question	**Answer**
Are we inserting any rows?	No
Are we querying data?	Yes
Do we perform any `processRequest()` initializations?	Yes
Do we query a single row using a primary key value?	Yes
Do we query data for a table?	No

In this case, we need to query a single row of a data for a primary key value. Using the findByKey() strategy ensures that we do not execute the query needlessly.

Use Case #2: Insert-Only Table w/ Add a Row Button

Question	**Answer**
Are we inserting any rows?	Yes
Do we need to query the view object in which we are inserting rows?	No

In this case, we need to initialize the table to prepare it for inserts as described above for view objects that do not query the database.

Use Case #3: Shared Component w/ a Table

Question	Answer
Are we inserting any rows?	No
Are we querying data?	Yes
Do we perform any `processRequest()` initializations?	Yes. The region is designed to query when the page renders.
Do we query data for a table?	Yes

In this case, we have no search criteria to test and there is no potential to lose pending user data. Therefore, we have no choice but to assume that the data should be queried each time the page renders. The only remaining concern involves avoiding the redundant data query as described above.

Known Issues

- Detail objects involved in master-detail relationships should never use `setMaxFetchSize(0)` as it indicates to the BC4J Framework that the query should be suppressed against the database. When you perform operations such as following, we expect that `executeQuery()` will have no effect on the state of the VO:

```
...
vo.setMaxfetchSize(0);
vo.insertRow(...);
vo.insertRow(...);
vo.executeQuery();
vo.reset();
// Iterate the rows
...
```

  When you iterate the rows, you should be able to get all the rows inserted in the VO. When BC4J Framework sees the "Max Fetch Size=0" on a view object it bypasses initializing a query collection object for the VO when `extecuteQueryCollection` is called on the VO to achieve the desired effect.

  This causes a problem with master-detail relationships. When the BC4J Framework fetches detail rows from the current master row it sometimes needs to swap the query collection. For example, the query collection of the detail row set in the internal accessor VO contains the correct data but when the query collection of the public detail VO had to be swapped to match the new current master row it did *not* occur because the "Max Fetch Size=0".

View Object Query Criteria

Where Clause and Where Clause Parameters

The rules and behaviors of the WHERE clause and WHERE clause parameters are described below for your information. As a general coding guideline, you should do the following to avoid having mixed (declarative and programmatic) WHERE and ORDER BY clauses:

- If a SQL statement can have a *finite* number of different WHERE or ORDER BY clauses, you should create distinct view objects at design time for distinct WHERE clauses and ORDER BY clauses. For WHERE clause parameters, use bind parameters (:1, :2, and so on) to bind in different values. This allows BC4J to bypass the need to re-prepare JDBC statements.
- If a SQL statement needs to be used for a search region and can be bound to an *infinite* number of WHERE clauses or if the search result can be sorted by multiple columns which you can not define ahead of time, then create one view object at design time and reuse the view object. Instead of specifying the WHERE clause and ORDER BY clause at design time, specify the WHERE clause and ORDER BY clause programmatically at runtime.

WHERE Clause

There are two types of WHERE clauses:

- Declarative WHERE Clause - set in the view object XML through the view object wizard.
- Programmatic WHERE Clause - set through `ViewObject.setWhereClause`.

The method `ViewObject.getWhereClause` returns only the where clause portion that you programmatically set through `setWhereClause`. The method `setWhereClause` does not alter the declarative WHERE clause set through the view object wizard.

WHERE Clause Parameters

Use `setWhereClauseParam(s)` to set the parameter values for all bind variables in the declarative and programmatic where clauses. The method `getWhereClauseParam(s)` returns the where clause parameter(s) that you set through `setWhereClauseParam(s)`.

If you put a WHERE clause with parameters in a view object SQL definition using the view object wizard, BC4J can still correctly find the bind parameters (even those embedded in SQL and not set explicitly through `setWhereClause`) and perform correct positional binding.

ORDER BY Clause

As with the WHERE clause, there is also a distinction between declarative and programmatic ORDER BY clauses. The method `getOrderByClause` returns just the ORDER BY clause that you programmatically set through `setOrderByClause`.

There is one variation in behavior, however, for the ORDER BY clause for the runtime SQL statement. For the example shown in Table 2 below, for both expert mode and non-expert mode view objects, if you call `setOrderByClause("empno")`, the "order by ename" part is *replaced* with "order by empno". But if you call `setOrderByClause(null)`, the original declarative ORDER BY clause, "order by ename" is restored.

For the expert mode view object example shown in Table 2, if you manually move the ORDER BY clause into the Query Statement section of the view object wizard, upon a runtime query, the ORDER BY would be part of the "from" subselect (works in Oracle 7 and 9 with no SQLException) as you'd expect. In addition, when you call `setOrderByClause("empno")`, "order by empno" would be placed outside the "from" subselect without removing the ORDER BY clause in the subselect.

Table 1: View Object SQL in view object XML (declaratively set through the view object wizard)

Expert Mode Unchecked	Expert Mode Checked

Generated Statement:
`select ename, empno from emp`
(derived from view object attributes)
 	
Query Clauses:
`Where: deptno = :1`
`Order By: ename`
	Query Statement:
	`select ename, empno from emp`
	`where deptno = :1`
	
	Query Clauses:
	`Order By: ename`
	
	(When Expert Mode is checked, the view object wizard places the ORDER BY clause separately in the Query Clauses section.)

Table 2: Programmatic Method Calls and Resulting Runtime SQL Statements

Sequence of Method Calls	Expert Mode Unchecked	Expert Mode Checked

1. `executeQuery();`	`select ename, empno from emp where deptno = :1 order by ename`	`select * from (`
		`select ename, empno from emp where deptno = :1)` `QRSLT`
		`order by ename`
		
		(BC4J puts the ORDER BY clause outside the FROM clause because Oracle 8.0 cannot handle ORDER BY within the "from" subselect clause; although Oracle 7 and 9 can handle ORDER BY in "from" subselect.)

2. `getWhereClause();`	Returns null.	Returns null.
3. `setWhereClauseParam(0,new oracle.jbo.domain.Number(10));`	Performs positional binding.	Performs positional binding.

4. `setWhereClause("empno > :2");` `setWhereClauseParam(1, new oracle.jbo.domain.Number(7000));`	`select ename, empno from emp where deptno = :1 and (empno > : 2) order by ename` (The new WHERE clause is appended with an AND operator.) :1 is bound to 10 (from previous `setWhereClauseParam`) :2 is bound to 7000.	`select * from ( select ename, empno from emp where deptno = :1) QRSLT where (empno > :2) order by ename` :1 is bound to 10 (from previous `setWhereClauseParam`) :2 is bound to 7000.
5. `getWhereClause();`	Returns "empno > :2"	Returns "empno > :2"
6. `setWhereClause(null);`	Just removes the extra "and (empno > :2)".	Just removes the extra "where (empno > :2)"

Application Modules in Detail

Overview

Contents

- Lazy Loading
- Nested Application Modules
- "Standalone" Application Modules
- WebAppsContext Application Coding Standards

Lazy Loading

Previously, any view objects, view links or nested application modules that you declaratively associated with an application using the BC4J Application Module wizard were eagerly instantiated by BC4J when it instantiated the container. Now, it is possible to enable "lazy loading" so child view objects, view links, and nested application modules associated with a containing application module at design time are instantiated on demand. Specifically, they are instantiated when you or OA Framework calls a `findViewObject()`, `findViewLink()` or `findApplicationModule()` on your application module instance, and *not* when the application module is created.

Tip: The term "load" in this context means "instantiate." This has nothing to do with query execution.

As described in the OA Framework Model Coding Standards, lazy loading should be enabled for all application modules.

- **Declarative Implementation** -- See Implementing the Model.

  Note: Any new application modules that you create in Release 12 automatically have lazy loading enabled.

- **Programmatic Implementation** -- Lazy loading should *not* be implemented programmatically. In other words, do *not* call `oracle.jbo.server.ApplicationModuleImpl.setLoadComponentsLazily()` as this method behaves in an unexpected manner.

  Note for Oracle's internal E-Business Suite application developers: You may use the mass conversion script published by the OA Framework team.

Usage Considerations

As described above, BC4J automatically creates the requested object, if necessary, when you call the `find*` methods. If you call `findViewObject()` and the view object does not yet exist, BC4J first creates and then returns a reference to the new view object.

Calls to any of the following `get*` methods do *not* prompt BC4J to create any objects. Instead, BC4J simply returns information about the instances that exist at the point at which the method is called.

- `oracle.jbo.ApplicationModule` -- `getViewObjectNames()`, `getViewLinkNames()` and `getApplicationModuleNames()`

  Tip: The `get*` methods in `oracle.jbo.ApplicationModule` include parameters (`boolean inclLoadedOnes`, `boolean inclNotLoadedOnes`) that let you decide whether to include the names of objects that have been loaded (instantiated), not loaded (but associated with the application module declaratively at design time), or both. See the Javadoc for additional information.

- `oracle.jbo.server.ApplicationModuleImpl` -- `getViewObjects()`, `getViewLinks()` and `getApplicationModuleImpls()`

- `oracle.jbo.server.ViewObjectImpl` -- `getViewLinkNames()` and `getViewLinks()`

The `get*Names()` methods in the `oracle.jbo.server.ApplicationModuleDefImpl` class return the complete list of objects defined at design time *regardless of whether or not they have been instantiated*. It still does not trigger any instantiation.

- If you have any initialization logic that you need to execute, you should add it to appropriate methods like `ViewObjectImpl.create()`, `ApplicationModuleImpl.createViewObject()` or `findViewObject()` instead of looping through the list of objects returned by the similar `get*` methods. This also applies to view links and application modules. The use of these hook methods ensure that objects are initialized properly regardless of whether lazy loading is enabled or not.

- To perform a cleanup operation on previously instantiated objects, using the `get*` methods is acceptable.

- To report on created objects for diagnostic purposes, using the `get*` methods is acceptable.

Note: You can use the `ApplicationModuleImpl.isLoadComponentsLazily()` method in this context to discover whether the application module has lazy loading enabled. You can also pass **true** for the `incNotLoadedOnes` parameter in your `get*Names()` method calls when reporting on objects that were associated with the application module at design time, but have not yet been instantiated.

- Do *not* use the `get*` methods for an existence check before creating new objects. Always use the `find*` methods as recommended in the OA Framework Model Coding Standards. The reason for this is the `create*` method could throw a `NameClassException` since it checks the names of objects that were created as well as the names associated with the application module at design time. If the name you're trying to use is in either list, it will complain because the `create*` method was designed to be used with lazy loading and the `find*` methods.

If you need to force component loading for an application module because you do not have appropriate initialization logic as described above, use `oracle.jbo.server.ApplicationModuleImpl.finishLoadingComponents(boolean recurse)` before using the `get*` methods.

Warning: Use this method carefully as a last resort because its use could degrade performance.

Lazy Loading and Application Module Extensions

When an application module extends a parent application module, it inherits the runtime component instantiation behavior of its parent. For example, if the parent application module has lazy loading enabled, then the extended application module will also have lazy loading enabled.

The child application module can also override its parent's instantiation configuration. In other words, if you explicitly set lazy loading characteristics in the BC4J Application Module wizard that differ from the parent, BC4J honors your choice for the child application module and you can override these declarative settings if needed. In general, children inherit their parents' BC4J properties.

Nested Application Modules

As discussed earlier in the Developer's Guide, it is possible to define a data structure for your root application module that includes nested application modules. To associate a nested application module with a region, specify either of the following properties:

- `AM Instance` -- The application module instance name as specified in the application module's data model. For example, `PoSummaryAM1`.
- `AM Definition` -- The fully qualified name of the application module instance. For example, `oracle.apps.fnd.framework.toolbox.tutorial.server.PoSummaryAM`.

If you specify the `AM Instance` property, OA Framework attempts to find and return the AM instance with the given name. It searches the application module associated with the parent web bean.

If you specify the `AM Definition` property, OA Framework assigns a generated AM instance name to the nested region. This name is comprised of the following values:

- Region code of the associated nested region (if the region was originally created in AK)
- Nested region application ID
- Nested region ID
- AM definition name.

OA Framework checks to see if the nested AM with the system-generated name has been already created under the application module of the parent web bean. If the instance is *not* found, OA Framework creates and uses a nested AM with the system-generated name. Otherwise, it reuses the pre-existing AM with the system-generated name.

When specifying *both* the `AM Instance` and `AM Definition` properties:

- If a matching AM instance is found, OA Framework compares its definition with the `AM Definition` property value. If there is a mismatch, OA Framework throws an exception.
- If a matching AM instance is *not* found, OA Framework creates and returns a nested application module instance with the given AM instance name and the definition.

Multiple Pages With the Same Root Application Module and Nested Regions

Two pages share the same root application module. To ensure that the nested region in the second page reuses the same nested AM instance associated with the nested region in the first page, follow these guidelines:

1. Specify the `AM Instance` property value to be the same for the nested regions in both pages.
2. The level of nesting should be the same. If the second page has an extra parent web bean with an extra nested AM *above* the nested region of our interest, then the nested region will return a new instance.

"Standalone" Application Modules

To create an application module outside the context of an OA Framework page or existing application module, use the `oracle.apps.fnd.framework.OAApplicationModuleFactory.createRootOAApplicationModule(AppsContext appsContext, String amDefName)` as shown below. You *must* use this signature to ensure compatibility with the Oracle database secure authentication mode. For additional information, see the `OAApplicationModuleFactory` Javadoc.

Warning: OA Framework does *not* pool factory-created application modules, therefore their scalability is not assured.

Note: When you have finished using the application module, call `ApplicationModuleImpl.remove()`. This performs cleanup operations for the application module and releases the JDBC connection back to the connection pool.

```
import oracle.apps.fnd.common.WebAppsContext;
import oracle.apps.fnd.framework.OAApplicationModuleFactory;
import oracle.apps.fnd.framework.OAException;
import oracle.apps.fnd.framework.server.OAApplicationModuleImpl;
import oracle.apps.fnd.framework.server.OAExceptionUtils;
private static OAApplicationModuleImpl getServiceApplicationModule()
{
  WebAppsContext ctx = new WebAppsContext
    ("<drive>:\\users\\dbc_files\\secure\\<server>.dbc");

  // Process the error stack to make sure that the WebAppsContext object
  // was created correctly.
  OAException oaex = OAExceptionUtils.processAOLJErrorStack(ctx.getErrorStack());
  if (oaex != null)
  {
    throw oaException;
  }
  // You can now initialize the user session for the new WebAppsContext
  // either by validating the session if you have an Oracle E-Business Suite
  // user session id, or by logging in with a user name and password
  // as shown.
  ctx.getSessionManager().validateLogin("<username>", "<password>");
  ctx.getSessionManager().setResp(601, 57417);

  // Process the error stack again before creating the root application
  // module with the WebAppsContext instance.
  oaex = OAExceptionUtils.processAOLJErrorStack(ctx.getErrorStack());
  if (oaex != null)
  {
     throw oaException;
  }
  return
    (OAApplicationModuleImpl)OAApplicationModuleFactory.createRootOAApplicationModule(ctx,
                "oracle.apps.fnd.framework.toolbox.labsolutions.server.SupplierSAM");
}
```

WebAppsContext Application Coding Standards

This section discusses coding standards for:

- Connection usage.
- Error handling.

Coding Standards for Connection Usage

Abide by the following rules with regards to `WebAppsContext` connection usage:

Note: The first rule is validated in developer mode. If this first rule is violated, a developer mode exception will be thrown. (However, this developer mode check cannot detect the release event if the same connection was checked out again through the `getJDBCConnection` call on an `WebAppsContext` instance after the release. This case is still unsecured.)

1. Do *not* call `releaseJDBCConnection()`, `logOut()`, or `freeWebAppsContext()` on a `WebAppsContext` instance, if the `WebAppsContext` instance is associated with a BC4J application module. In addition, do *not* call `Connection.setAutoCommit(true)` on WebAppsContext's main connection, if the `WebAppsContext` instance is associated with a BC4J application module.

   Tip: An application module needs to continue to use this connection for its transaction. Therefore, *only* the application module should manage WebAppsContext's main connection. An application module needs to use the connection in auto-commit = false mode to allow posted changes to be rolled back.

   Note: A `WebAppsContext` instance is associated with an application module, if it was either used to create an application module

or was handed out from the application module's transaction through the `OADBTransactionImpl.getAppsContext()` call. An application module uses the associated WebAppsContext's main connection (`WebAppsContext.getJDBCConnection()`) for its transaction; in other words, `OADBTransactionImpl.getJdbcConnection()` references the connection handed out from `WebAppsContext.getJDBCConnection()`.

If a `WebAppsContext` instance is *not* associated with an application module, (although these occasions are rare), you *must*:

- Call `releaseJDBCConnection()` on the `WebAppsContext` instance after you are done using the instance. You must do this even if you did not explicitly call the `getJDBCConnection()` method.

  Note: This ensures that the JDBC connection you checked out, or an AOL/J API call implicitly checked out (such as for getting a profile option value), is released back to the connection pool to prevent connection leaks. If no connection was checked out or the connection was already released, the `releaseJDBCConnection()` call results in a loop.

- Avoid the `logOut()` call unless you need to reset the user settings and reuse the `WebAppsContext` instance. This is because it performs some extra operations, which could be expensive if the `WebAppsContext` object is just going to get garbage-collected.

- Use the `freeWebAppsContext()` on `WebAppsContext` instances that will be garbage-collected and *not* reused. `freeWebAppsContext()` is intended to perform cleanup, including the `releaseJDBCConnection()` operation before the `WebAppsContext` object is destroyed, which prevents resource leaks.

2. The *extra* JDBC connections handed out from `WebAppsContext` instances, via the `WebAppsContext.getExtraJDBCConnection` call, are *not* used for the application module's transaction and the application module is also unaware of them. Therefore you *must*:

   - Always release the *extra* JDBC connections with the `WebAppsContext.releaseExtraJDBCConnection` call after you are done using them. This prevents connection leaks.

   - Use the *extra* JDBC connection for a short-lived transaction within a request. In other words, get the extra connection, use it, and release it within a request. This not only prevents connection leaks but also prevents the connection from becoming timed out, which may later cause SQL exceptions.

3. Never call `Connection.close()` on the connection handed out from `WebAppsContext` (through `getJDBCConnection` or `getExtraJDBCConnection` call on WebAppsContext) or `OADBTransactionImpl` (through the `OADBTransactionImpl.getJdbcConnection()` call). OA Framework connections are pooled through AOL/J connection pool for reuse, therefore, you should *not* close any pooled connections.

   Note: Only the AOL/J connection pool manager can close or destroy unused connections.

Warning: Prematurely releasing the connection unlocks it for use by other threads, which can lead to concurrency issues. In addition, the unlocked connections may get closed by the AOL/J connection pool manager, and as a result, cause SQL exceptions when the closed connections are later used in the application module's transaction.

Coding Standards for Error Handling

Instead of throwing an exception, the `AppsContext` object logs the errors in its error stack. It is the caller's responsibility to check for errors in the error stack after calling an `AOL/J` method. For information and code example on how to process the AppsContext error stack, see the Javadoc for `OAExceptionUtils.processAOLJErrorStack(ErrorStack es)`.

Warning: You *must* actively poll for the exception with AOL/J APIs. If you don't, an important error may be ignored and affect your application behavior.

Service Frequently Asked Questions (FAQ)

During the process of developing an application, when do I consider the entity layer?

The entity layer serves as the basis from which all other parts of an application are derived. Integrations, reports, and user interfaces, are all variations of the foundation described in entities. The general rule is that the more time you spend getting the entity model right, the less time is needed in other layers trying to fit the implementation into the requirements. We recommend that you invest the appropriate time to build out the entity layer, adding associations, rules, business processes, validations, and state transitions. Focus first on the business areas that customers may want to modify.

What are the implications of implementing entities with PL/SQL?

The preferred implementation of an entity is with Java. The following is a list of disadvantages to implementing entities with PL/SQL:

- It is not as extensible and customizable as Java.
- Code executes on the database, which makes it impossible to call to the middle-tier.
- To run on the middle-tier, any code written in PL/SQL may have to be replicated in Java.
- You cannot take advantage of features such as, declarative validations, state transitions, and declarative business event publishing.

Can I create new entities that are implemented with PL/SQL?

No. Not unless results have been verified and show a significant gain in performance.

Is there a standard that product teams should follow when creating entities that wrap PL/SQL?

Yes. Product teams should wrap all PL/SQL with entities that subclass a base java class, which is standard for PL/SQL based entities. This enforces a clean, consistent implementation for PL/SQL based entities and can provide information leveraged by other tools.

Is it necessary to define all associations between entities?

Yes. It is essential that the entities and the relationships between entities be leveraged to the fullest. Having the associations defined properly helps to improve the long-term ownership costs of the application. New and existing Jdeveloper tools take advantage of this metadata to increase productivity, ease customization, and manage impact analysis.

Can I extend another product teams entity or service view object?

No. By subclassing another product teams entity object or service view object, a dependency is created that is difficult to manage. If the entity object or service view object changes, there is a risk of conflicts spreading across product teams. We recommended that you discuss the changes with the product team and have them support your needs, preferably through a service.

When my Java test program calls the service bean methods on the application module, the database is not updated. Nothing happens.

See the ToolBox Service Bean lab Create your Test Class. See notes on using `invokeServiceBatch()`.

We have implemented parameter passing into the service via DataObjects, though the SDOs do not have any underlying VO (in other words, these SDOs are standalones implementing the relevant Service classes). Is this okay?

No. All SDOs must have an associated SVO.

If I have a purchase order with multiple lines, can clients add lines, update lines and delete lines with a single method call?

No. We support updating and adding in a single call using the merge operation. Clients must make separate calls to perform delete operations.

How do I handle versioning my service interface and methods if I need to make changes that are not backwardly compatible?

See Method and Service Data Object Versioning in Creating and Testing Service Beans.

How do I automatically create task data (child) objects for the parent service request during the creation of a service request object? A service request object may have 0 or more task data objects.

Automatically creating task data objects for the parent service request object should be performed in a layer below the application module - preferably in the entity object (EO). You *must* flag the Service View Object for the child row as `AssociationConsistent` to insure that when the child EOs are created by the parent EO, a row is added to the corresponding service view objects. This is necessary because the `processDataList` on the service interface allows the client to request a copy of the business object upon successful completion of the Create option. This business object needs to include the child SDO rows.

See the `setAssociationConsistent` on the `ViewObjectImpl` class Javadoc for more information.

Testing Business Service Objects

Overview

When you've completed your implementation, you can use the BSO API Testing Framework to record test scripts stored as structured XML. Source control these scripts with the corresponding services and run them when you check in your service code or as part of a periodic unit regression test. To create your test scripts, follow the instructions and example provided in this chapter.

Content

- BSO API Tester
- Creating a Test Script
- Known Issues

BSO API Tester

The BSO API Testing Framework provides a user interface (UI) that helps in the recording of test scripts and playing them back, as part of a periodic unit regression test. The tests are recorded in documents, as structured XML data. A saved XML document is called a test file. Each test file can contain multiple test suites and each test suite can have multiple test cases.

Once you build your tests for the business service objects, you don't need to test the generated external technology interfaces (web services) because:

- The BSO interface parameters are all `Serializable` value objects and data primitives passed by value. Testing and validating the results of a Java call is logically identical to testing the results of a web service call.
- Assuming the generated logic is tested by the team building the generation utilities, then the generated results do not need to be re-tested by each product team.

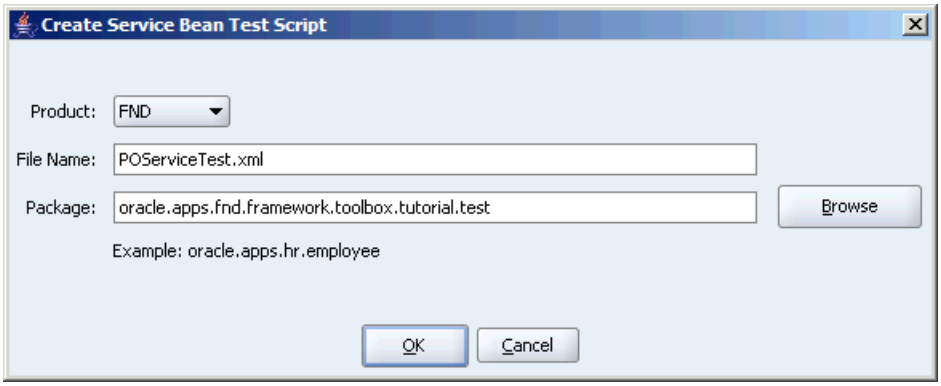
Creating a Test Script

Step 1: To create a test script for your service, select the service application module BC4J component in the **System-Navigator** and then choose **Create Test Script...** from the context menu.

Step 2: In the **Create Service Test Script** dialog, select your product shortname from the **Product** poplist, and specify a **File Name** and a **Package** for the test script, as shown in Figure 1.

Note: XML test scripts should be created under the package space *.test where * is the location where the Service interfaces reside. For this example of the Purchase Order service web bean, the test script is created under oracle.apps.fnd.framework.toolbox.labsolutions.test. To name the test file, start with a file named Test. Prepend the service name to Test and optionally append an index number to it: <*ServiceName*>Test<*n*>.xml. For example: POServiceTest.xml, POServiceTest1.xml, POServiceTest2.xml, and so on. Enforcing a standard package namespace for all test scripts becomes very useful when doing any analysis or batch operations at a later stage. See the OA Framework File Standards for a summary of all recommended naming conventions related to services.

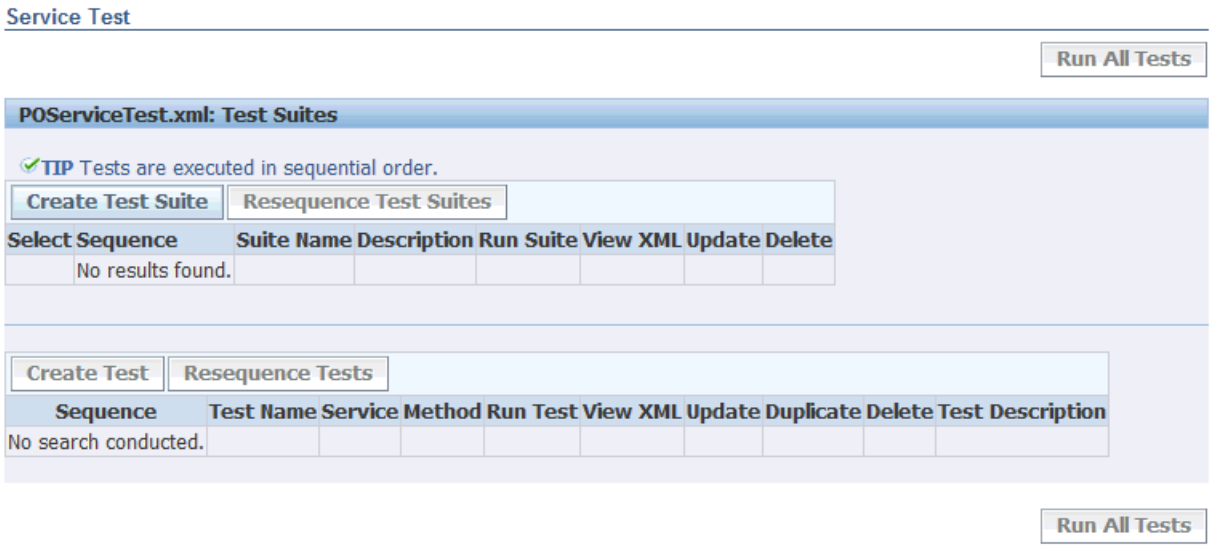
Figure 1: Create Service Test Script dialog



Note: The **Product** poplist shows the products defined in the Integration Repository tables. If you do not see your product in the poplist, check to see if the Integration Repository patch has been applied to your target environment.

Step 3: Once you select **OK**, JDeveloper creates the XML test script in the designated package and launches the Service Testing Framework user interface in your target browser as shown in Figure 2.

Figure 2: Service Testing Framework User Interface



Step 4: Create a test suite by selecting the **Create Test Suite** button. Enter the test suite name and description in the **Create Test Suite: Description page** that appears. Select the **Suppress Commit** check box if you do not want to commit modifications to the database when you run the test suite. However, any changes made by one test in the suite is visible to other tests that are later in the sequence.

Step 5: Create a test script by selecting the **Create Test** button from the **Service Test** page. A **Create Test** flow initiates:

Step 5.1: In the **Create Test: Description** page, provide information to describe the test:

Note: See the OA Framework File Standards for a summary of all recommended naming conventions related to services.

Name	Description
Test Name	Enter a test name such as `getPurchaseOrderTest`.
Service	Select a service from the a poplist. For example /oracle/apps/fnd/framework/toolbox/tutorial/PurchaseOrderService. The **Service** poplist gets its values from the Service manifest of your current JDeveloper workspace.
Method	Select a service method from the poplist to test. For example, `getPurchaseOrder` or `createPurchase`. You can also select generic methods such as `queryDataSource` and `processDataSource` to test.
Description	Enter a description for the test, such as "Test getPurchaseOrder method".

Step 5.2: In the **Create Test: Method Parameter Values** page, enter input values for the method parameters. The hierarchy of parameters for the method is displayed as a tree while the specific parameters for a selected node in the tree are displayed in a table, as shown in Figure 3.

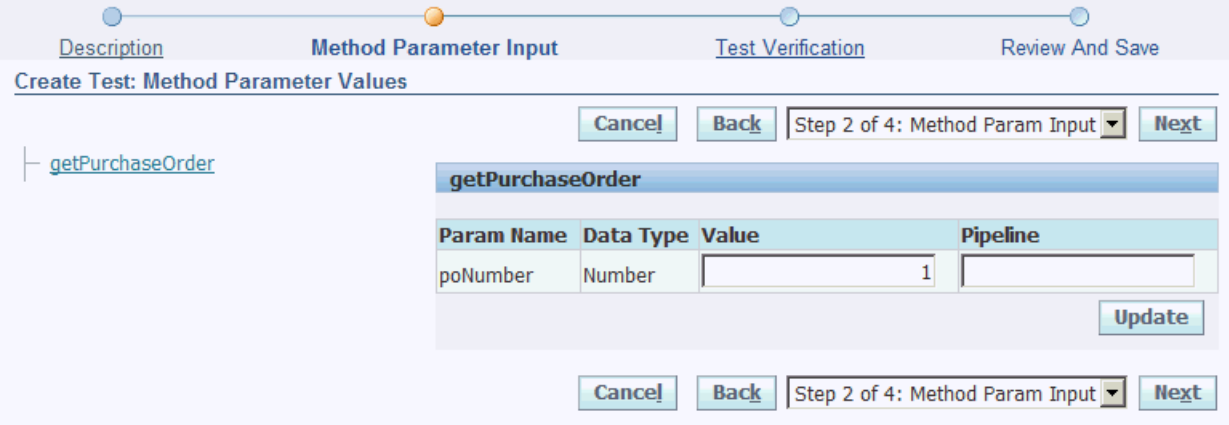
In addition to specifying an input value for a parameter, you can also create a "pipeline" between multiple test scripts. A pipeline allows you to send the output for a parameter from one test script to use as input for a parameter in another test script. You can pipeline from one test script to multiple test scripts as long as all pipelined test scripts are in the same test suite.

For the parameter in a test script whose value you want to send through the pipeline, enter a **Pipeline** value. Let's call this the sending test script. In your other receiving test script(s), for the input parameter that you want to receive the pipelined value, enter the same **Pipeline** value as specified in the sending test script. You can specify any value for **Pipeline** as long as you use the same value for all test scripts that are pipelined together.

Refer to the Services Lab of the Toolbox Tutorial for a more detailed example of how to create a pipeline between two test scripts.

Note: The hierarchy shown on this page will vary based the method you are testing. When you are done specifying the input parameters, choose **Next** to execute the method with the parameter values you entered.

Figure 3: Create Test: Method Parameter Values page



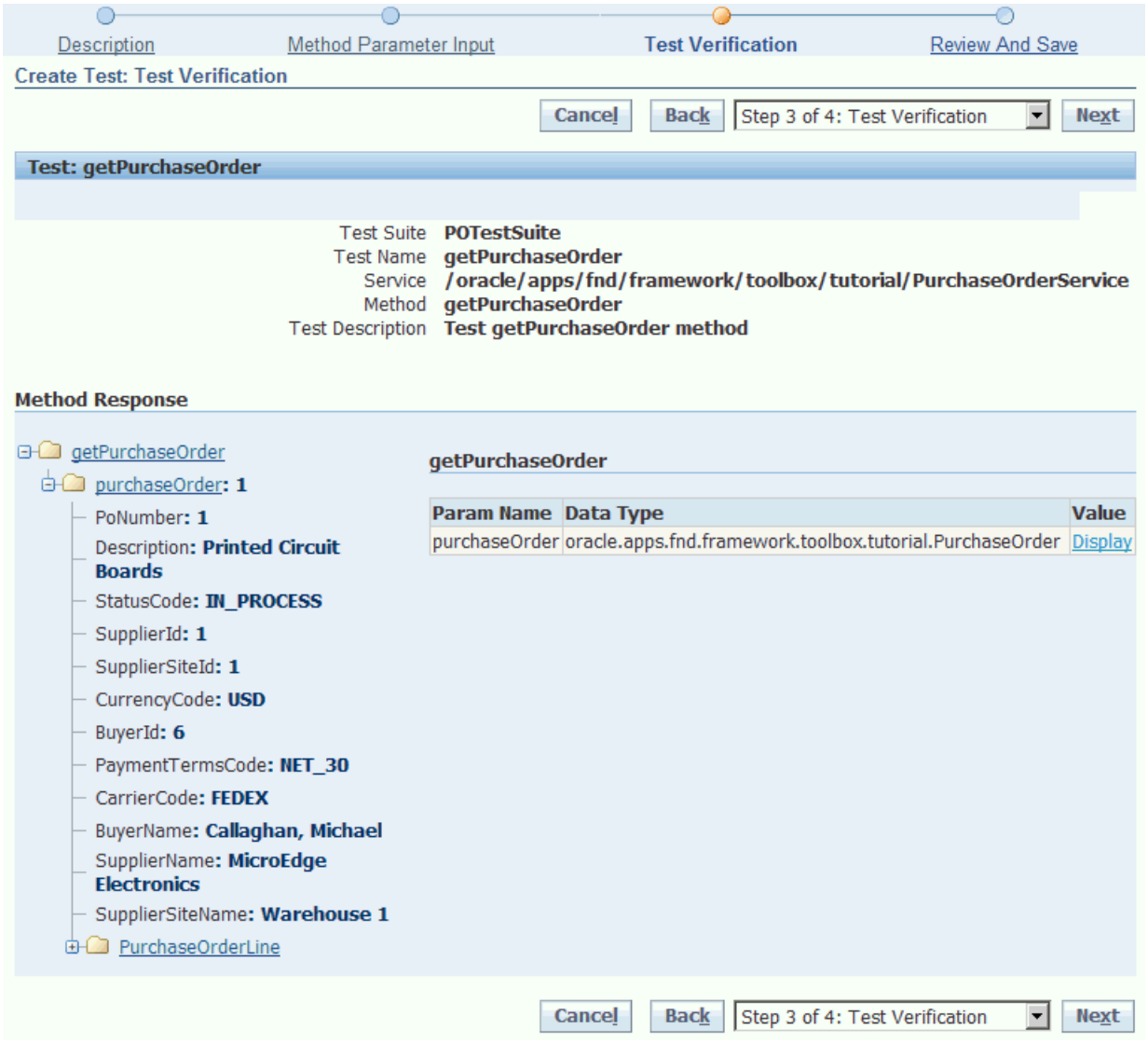
Step 5.3: In the **Create Test: Test Verification** page, verify the resulting method response output as shown in Figure 4 and select **Next** when you are done.

The method response output is shown in a table as well as a tree to illustrate the hierarchy, if any, of the method output. The tree is expanded one level and both the tree and the table display only those parameters that contain a value. In the example shown in Figure 4, the `getPurchaseOrder` method returns a purchase order object. Select the **Display** link to show the detailed values of the purchase order object in the table: PONumber (Number), Description (String), StatusCode (List), SupplierId (Number), and so on.

You can then further select the **Display** link for the PurchaseOrderLine list to show the index number for each PurchaseOrderLine in the list. When you select a PurchaseOrderLine link, the table displays the values of the PurchaseOrderLine parameters.

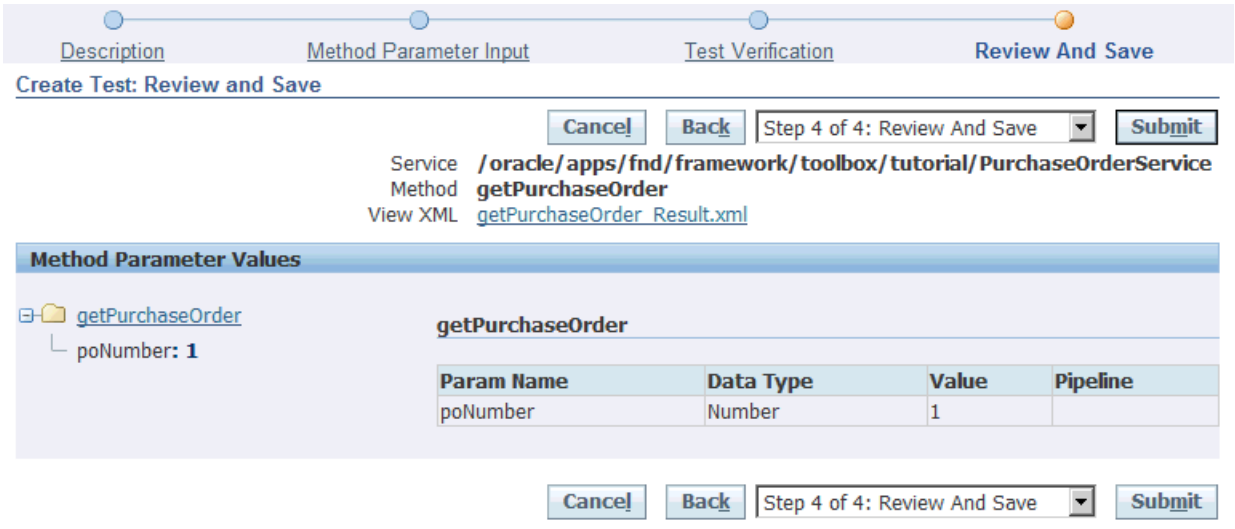
Generally, a **Display** link appears in the parameters table for any parameter that is an object or list that can expanded. Alternatively, you can expand/collapse the parameter tree directly to view the response output structure and parameter values.

Figure 4: Create Test: Test Verification page



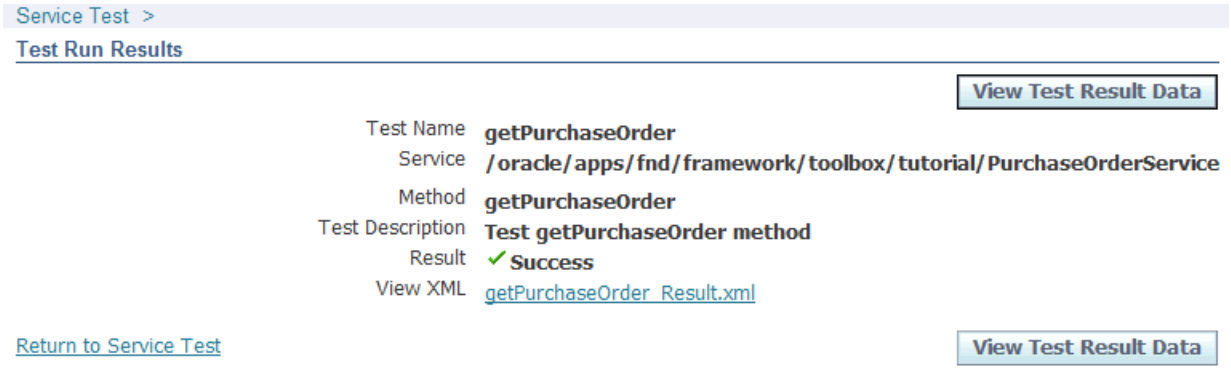
Step 5.4: In the **Create Test: Review and Save** page, as shown in Figure 5, review the input parameters and select the **View XML** linked file to review the test script as structured XML. If you are satisfied with the test script, select **Submit** and then **OK** in the following confirmation page to save the test results. The saved test results will be what you compare to the output of future regression testing.

Figure 5: Create test: Review and Save page



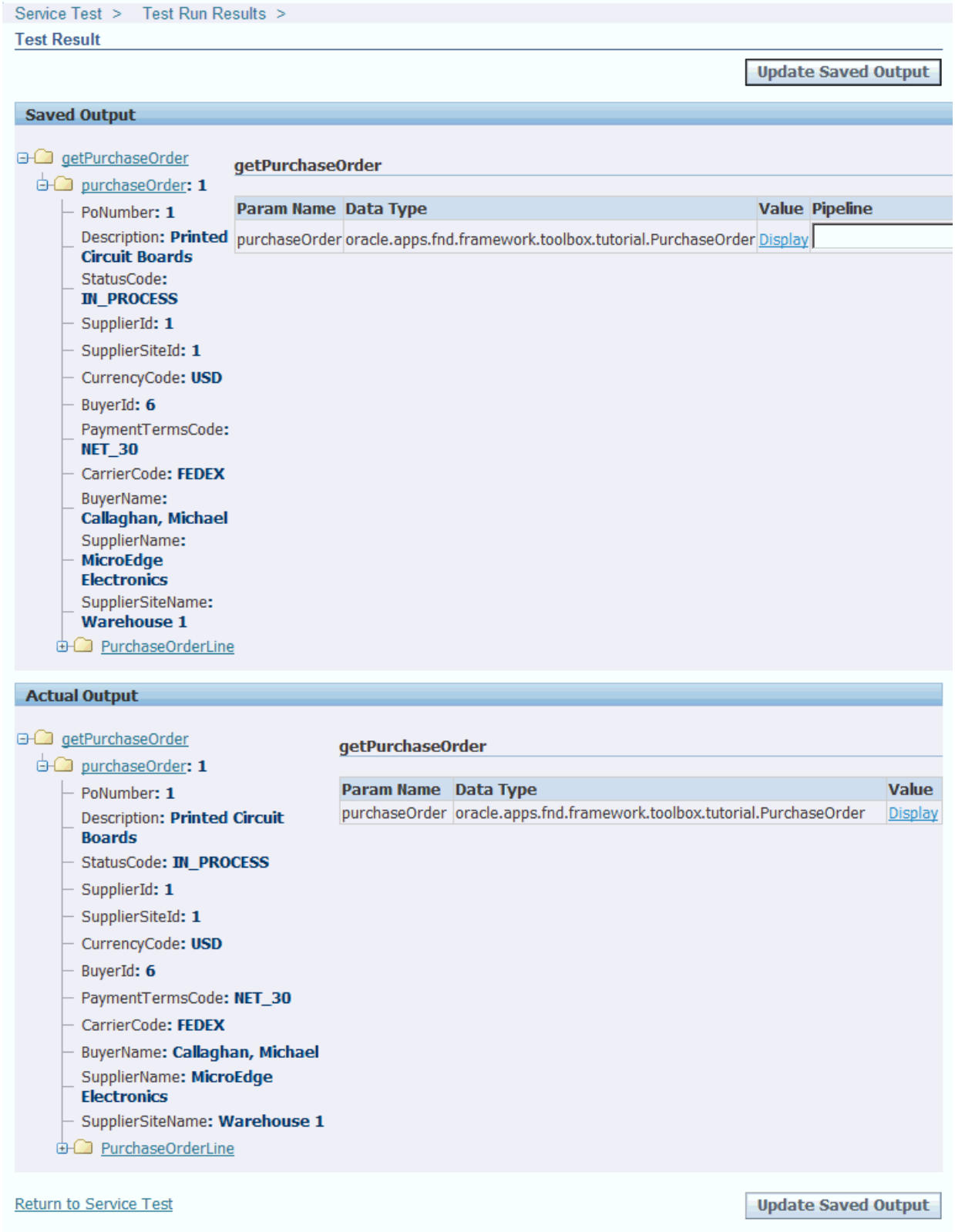
Step 6: Back in the **Service Test** page your new test is listed under the table that lists the tests for the selected test suite. Select the **Run Test** icon to run the test you just created and view the result in the **Test Run Results** page as shown in Figure 6.

Figure 6: Test Run Results page



Select the **View Test Result Data** button to compare the **Saved Output** with the **Actual Output** as shown in Figure 7.

Figure 7: View Test Result Data page



Step 7: Some methods, such as the `createPurchaseOrder` method, require system-generated data like primary key attributes and dates that cannot be predetermined when you create a test. Therefore, do not include these attributes in your saved output for comparison against actual output in your test script. To mask specific attributes in the saved output so they are excluded from the comparison that determines whether a test run is successful, select the appropriate node in the Saved Output tree that contains the attribute you wish to mask. In the parameter table for that node, check the **Masked** checkbox for the attribute(s) you do not want to compare in your future test runs. Select **Update Saved Output** when you are done.

Step 8: When you are done with the **Test Result** page, you can return to the **Service Test** page by selecting the **Return to Service Test** link. At this point, you can:

- create additional tests or test suites
- view the XML of a test suite or an individual test script
- update, delete or run a test suite or test script
- re-sequence the order that your test suites or test scripts (if you have more than one) run before you select the **Run All Tests** button

To re-launch the Service Testing Framework UI in the JDeveloper Navigator, locate the XML test script file you created in Step 2 under the appropriate Service interface test package. Select the XML test script file and choose **Edit Test Script...**, **Run Test Script...**, or **Debug Test Script...** from the context menu.

Using the Method Parameter Values Page

The **Method Parameter Values** page, as shown in Figure 3 of the **Create Test** process flow, allows you to specify input values for parameters in the method to be tested. The method's parameters are displayed in a table and its hierarchy is also rendered as a tree structure. The parameter table refreshes to show the appropriate context of parameters for the tree level you select. The **Value** field of the parameter table also varies to reflect the context of each parameter's Data Type.

For example, if a parameter has a data type such as Boolean, String, Date, or Number, the **Value** field displays a text input field or poplist for you to input a parameter value. However, if a parameter has a data type such as Object or List, the **Value** field displays linked text such as **Create** or **Add <*ListName*>**. Selecting the link would re-render the parameter table to display the supporting parameters needed to create the prior parameter Object or add to the prior parameter List. In the case of an Object parameter, after you specify the parameters needed to create the Object, and navigate back to the prior level, the **Create** link gets replaced with an **Update** and **Remove** link.

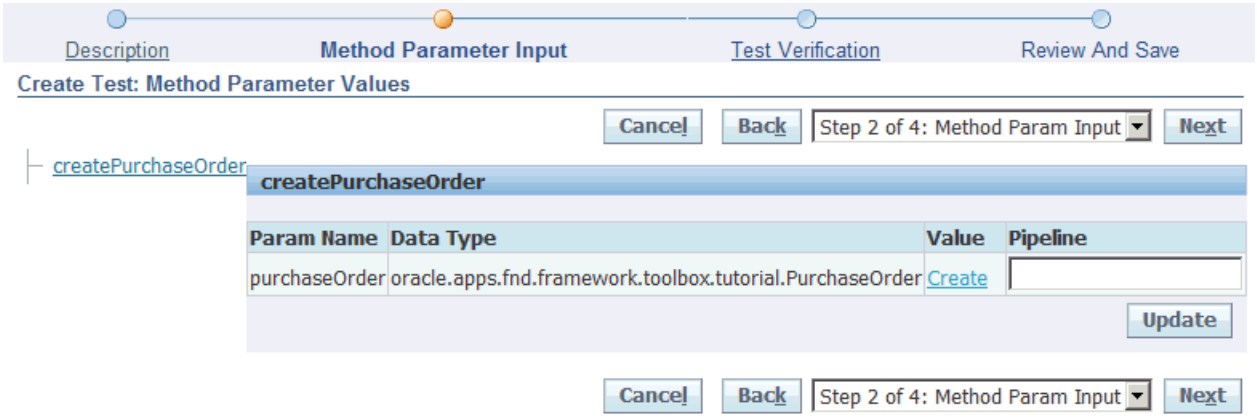
Rather than list all the possible parameter data types and describe how the **Value** field behaves for each data type, just be aware that the **Value** field will vary according to the context of each parameter's data type.

After you specify input values in the table and select the **Update** button on the page, the page refreshes the tree with the input values you specified and refreshes the tree and table to the prior parameter input level. Even if you do not specify new input values in the parameter table, you can always select the **Update** button to navigate back to a prior level in the parameter hierarchy. The parameter table displays all the possible parameters you can enter, while the tree displays the hierarchical structure of the method's parameters and only those parameters for which you have specified input values.

Example

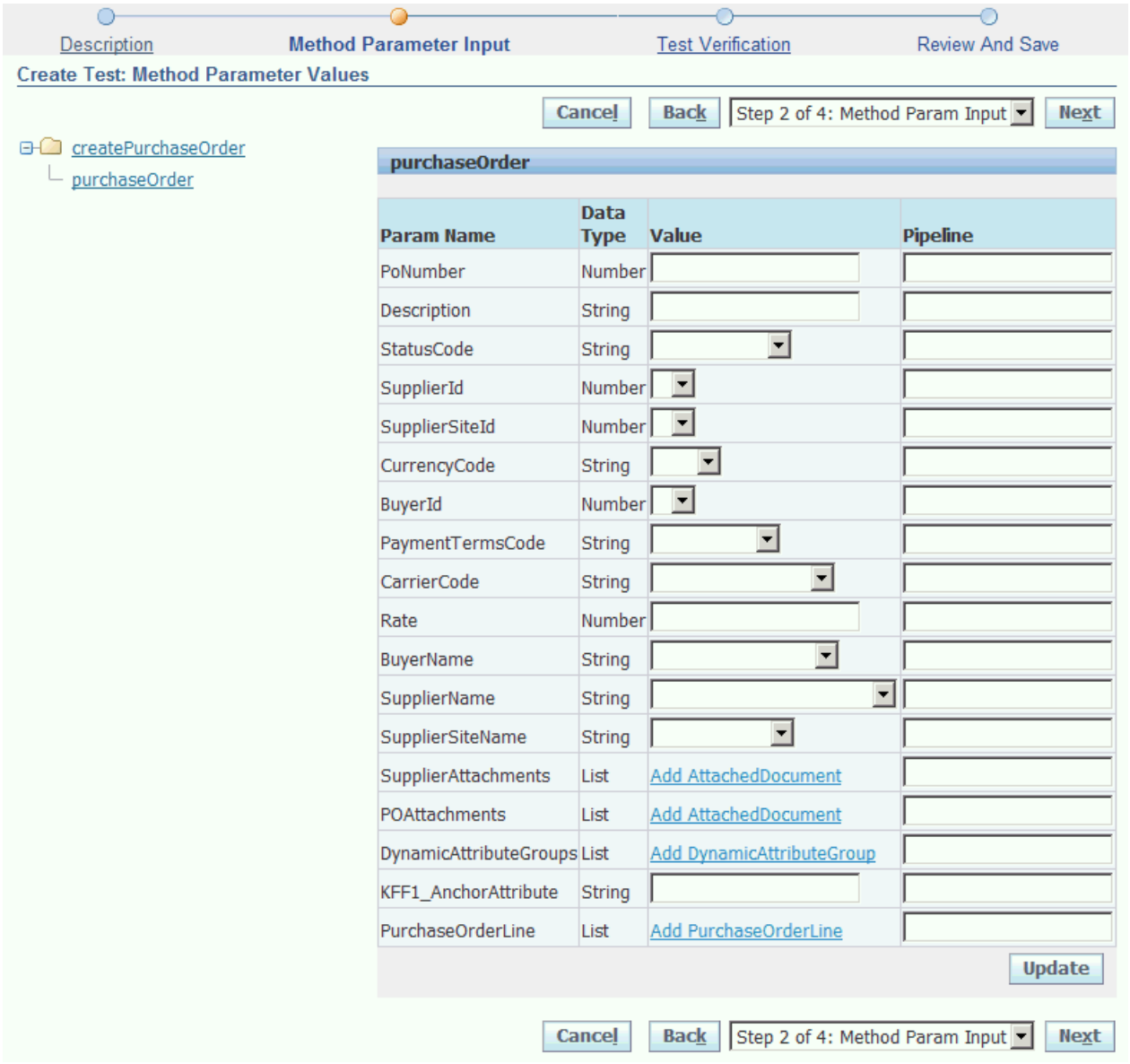
Consider the `createPurchaseOrder` method of the `PurchaseOrder` service, in theTutorial project of the OA Framework Toolbox Tutorial. This method requires a parameter called `purchaseOrder`, which is of data type oracle.apps.fnd.framework.toolbox.tutorial.PurchaseOrder. The **Value** field for this parameter initially displays a **Create** link, as shown in Figure 8.

Figure 8: Create Test: Method Parameter Values page for createPurchaseOrder Method



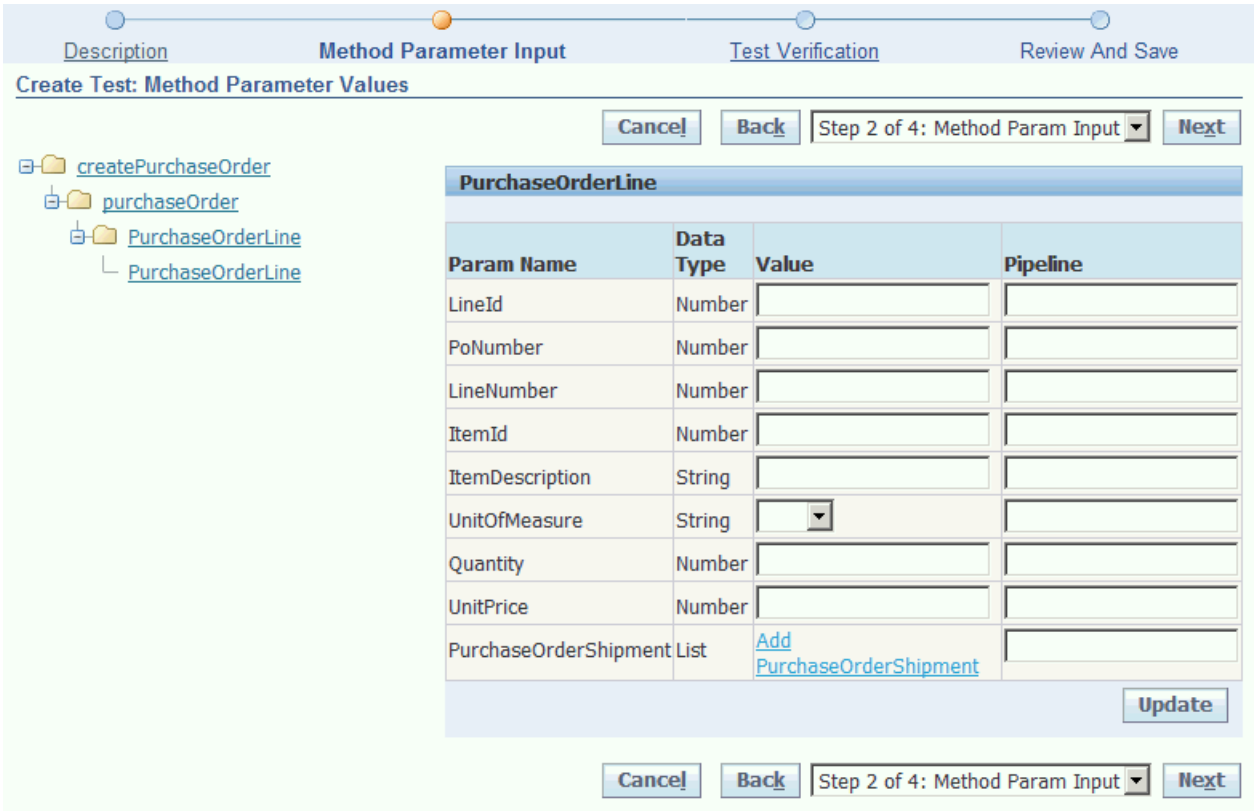
When you select the **Create** link, the Parameter table refreshes to display the list of parameters you need to create the `purchaseOrder` object, as shown in Figure 9.

Figure 9: Create Test: Method Parameter Values page showing parameters for purchaseOrder object



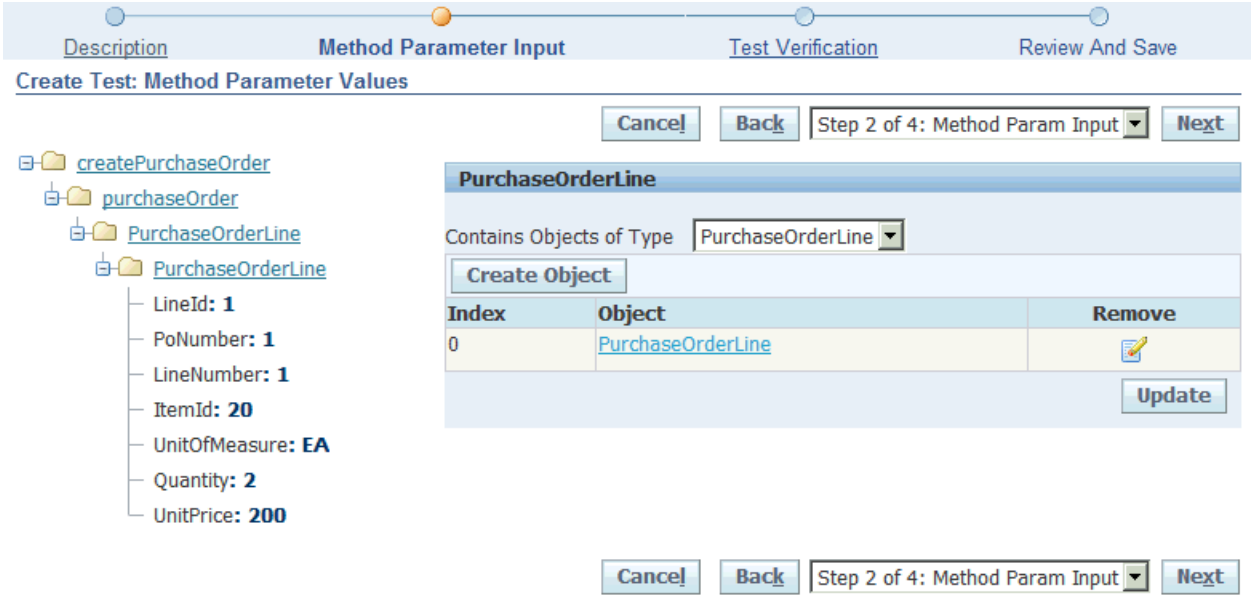
Note: The `purchaseOrder` object also contains a list of `PurchaseOrderLine`. If you select the **Add PurchaseOrderLine** link, the table refreshes to show the list of parameters you can specify to create a PurchaseOrderLine to add to that list, as shown in Figure 10.

Figure 10: Create Test: Method Parameter Values page showing parameters for PurchaseOrderLIne object



Once you enter input values for the PurchaseOrderLine and select the Update button, the page refreshes as shown in Figure 11, to indicate that a new `PurchaseOrderLine` object has been added to the `PurchaseOrderLine` list.

Figure 11: Create Test: Method Parameter Values page showing updated tree with newly created PurchaseOrderLine.



The tree now reflects the parameter input you've specified and the table refreshes to the prior parameter level. In this case, it shows the PurchaseOrderLine list. If you want to create another PurchaseOrderLine object to add to the list, you can select the **Create Object** button on the table. You can also select the **Remove** icon to remove the `PurchaseOrderLine` object that you just added to the list.

Parameter	Description	Comments
-choice	<APPLTOP> or <JDEV>	APPLTOP or JDEV are the only valid values for this parameter.
-jdev_ide	<JDEVELOPER IDE>	Required if -choice = JDEV.

-appl_top_cp	<APPLTOP CLASS PATH>	Required if -choice = APPLTOP
-svc_cp	<SERVICE CLASS PATH>	Required if -choice = JDEV
-test_scripts_path	<TEST SCRIPTS PATH>	
-dbc_file	<DBC FILE>	
-username	<USER NAME>	The default value for this parameter = FWKTESTER.
-responsibility_name	<RESPONSIBILITY NAME>	Must be within quotation marks. The default value for this parameter = "OA Framework Toolbox Tutorial Labs".
-resp_appl_name	<APPLICATION SHORT NAME>	The default value for this parameter = AK
-sec_grp_id	<SECURITY GROUP ID>	The default value for this parameter = 0
-nls_lang	<NLS LANGUAGE>	The default value for this parameter = AMERICAN

Executing Test Scripts Against JDEVELOPER

Parameter	Description	Example Value
-choice *	Select to run against JDEVELOPER	JDEV
-jdev_ide *	Jdev IDE	/jdevbin/NT/120drop2
-svc_cp	Directory of service classes, SVC.mf	/home/jfrost/jdevhome/jdev/myclasses
-test_scripts_path *	Directory of test scripts	/home/jfrost/jdevhome/jdev/myprojects/oracle/apps/yourtestdir
-dbc_file *	Dbc file name	/jdevhome/users/dbc_files/secure/ap664sdb_fwk12dev.dbc
-username	User name	FWKTESTER
-responsibility_name	Responsibility name	"OA Framework Toolbox Tutorial Labs"
-resp_appl_name	Application short name	AK
-sec_grp_id	Security group Id	0
-nls_lang	NLS language	AMERICAN

Executing Test Scripts Against APPLTOP

Parameter	Description	Example Value
-choice *	Select to run against Quick Apache	APPLTOP
-appl_top_cp *	Directory of classes, SVC.mf	/nfs/lafrz/atg4_top/applmgr/fwk12dev/fwk12devcomn/java
-test_scripts_path *	Directory of test scripts	/lafrz/atg4_top/applmgr/fwk12dev/fwk12devappl/yourtestdir
-dbc_file *	Dbc file name	...../fwk12dev/config/appl/fnd/12.0.0/secure/fwk12.dbc
-username	User name	FWKTESTER
-responsibility_name	Responsibility name	"OA Framework Toolbox Tutorial Labs"
-resp_appl_name	Application short name	AK
-sec_grp_id	Security group Id	0
-nls_lang	NLS language	AMERICAN

Known Issues

- When an individual test is executed, the transaction is committed even if the **Suppress Commit** checkbox is selected for the test suite.
- During the creation of a test suite, at least one test case should be created within it. Otherwise, that test suite becomes unusable.

Deploying Services

Overview

This chapter discusses deployment methodologies for the Business Service Objects. When deploying a service, all its related artifacts, including the XML files and the Java files should be included.

Content

- Internal Deployment Methodology
- External Deployment Methodology

Internal Deployment Methodology

This methodology uses ARU to build the patch. When a patch is created for a product, ARU looks for standard annotations, generates loader files (one per interface), and automatically includes these loader files in the patch.

When services are applied to a customer site for the first time, the customer decides which ones to deploy as web services. Additional entries are created in the Integration Repository for each interface/technology combination. When subsequent patches are applied to the same environment, the patch application process detects generated technologies for changed components and automatically regenerates updated artifacts using the processes described in detail above.

External Deployment Methodology

Note: This is applicable to all BSOs, including customer-developed services.

The easier but more time-consuming way of deploying a service is to create a patch and apply it on the environment where the service should be available. The ServiceDeployer tool will be useful in such cases. Follow the instructions in the Service Deployment Wiki page for details on using the tool.

The deployment process involves three steps:

- Copying the Java class files and XML files to JAVA_TOP of the environment.
- Creating an input file containing list of Service XMLs (SAM and SVO).
- Running the deployer tool.

Introduction to Service-Oriented Architecture and Services

This document provides an overview of service-oriented architecture (SOA) and services, a terminology list, and discusses how to adopt service-oriented architecture.

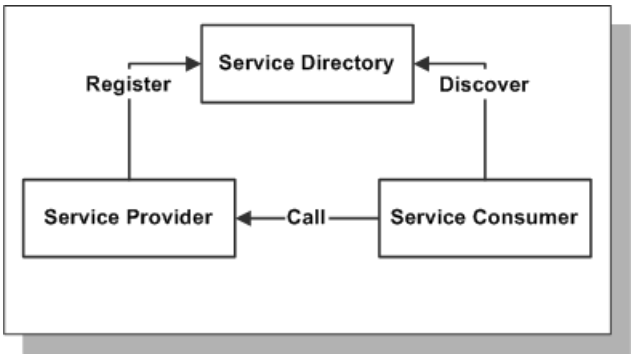
Service-Oriented Architecture Overview

The concept of a service has been around for a long time, but the concept of service-oriented architecture is relatively new. Service-oriented architecture is a style of building applications to promote loosely coupled, independently managed components that can be reused and arranged into active and fully functional applications. A SOA provides the following:

- The ability to build and register a service.
- The ability for consumers to discover and call a service.
- A platform in which to execute the service.

The following diagram demonstrates the basic concept of a SOA:

Basic concept of a service-oriented architecture



With service-oriented architecture you can:

- Easily integrate Oracle E-Business Suite with each other.
- Leverage existing investments in currently deployed applications.
- Easily adapt your applications to demanding needs.

Services Overview

Services are a key component in the Oracle E-Business Suite's cohesive architecture for the development, testing, cataloging, and securing of public and private interfaces. Developers work to a single set of guidelines and implementation instructions to create services, which are automatically published to a centralized interface repository when they source control the code.

Services are self-describing, standalone components that can be deployed as web services, EJB session beans or as co-located Java APIs (meaning they are deployed in the same Java Virtual Machine [JVM] as any clients that use them). There are different types of service interfaces that are largely differentiated by their intended usage. Oracle supports business object services.

A *business object* is a self-contained representation of a real-world object in the business domain: a document, a place, a person or a thing. From an implementation perspective, a business object involves one or more view objects (VOs) mapped to underlying entities (EOs) related through composition (meaning that child entities cannot exist without their parent). For example, the simplified purchase order business object in the OA Framework Toolbox Tutorial is comprised of a header with one or more lines, each of which has one or more shipments. The shipments can't exist without a line, and the lines can't exist without a header. (The shipments and lines are *child business object components*). If you delete a header, you delete all of the lines and the shipments. Often the only difference between a business object view object and the underlying entity object is the presence of foreign attributes on the view object, such as a supplier name on a purchase order.

The following is a summary of use cases for business object services:

- Business-to-Business Integration.
- Application-to-Application (service-to-service).
- BPEL
- Data import, export, and synchronization.
- Rich reporting frameworks, such as XML Publisher.
- Support for the SDO standard.

Creating Web Services Automatically

Automated technology-specific generators read the interface repository service definitions to create web services without requiring additional effort by the developer. This approach is extremely efficient from a resource allocation perspective. For example, individual developers don't need to know how to build web services. It also isolates product team code from any changes to relevant deployment technology standards.

Refer to the Oracle E-Business Suite Integration Repository User's Guide for information on how a service becomes a web service.

Terminology

Business Object	Self contained representation of a real world object in the business domain. The implementation of a business object involves one or more entities related through composition. It is a document, and therefore has a definition as well as physical instances, but has no methods or operations.
Canonical Business Object	The version of the business object that is understood by Oracle E-Business Suite as being the definitive definition. Commonly used for business integration and application synchronization. To better support standards based business integration it uses standardized naming for the attributes and business object components. To support application synchronization, it generally contains all data and child business object components necessary to accurately reproduce the business object on another database instance.
Business Object Service	Provides access to, and operations on, business objects. A business object service may support a single or many business objects. Business Object Services are stateless.

Adopting Service-Oriented Architecture

One of the biggest challenges in adopting a service-oriented architecture is creating appropriate services with the right level of abstraction. Where do you start and how deep should you go? When analyzing your business requirements, carefully consider what software components would benefit as services. As a general rule, services are coarse grained, that is, they expose standard create, update, delete functionality rather than update simple attributes.

See the Service Design and Project Guidelines document for more information.

Guidelines to determine strategic areas for services vary based on a product's business requirements and needs.

Consider the following:

- What are your external integration points?

  Simplifying integration is a key benefit of services. You can create services to update data directly, call PL/SQL programs, or call known processes that manage integration points.

- What processes does your application provide and reuse in many places?

  Making these processes known and discoverable facilitates reuse and provides a layer of consistency that you can benefit from.

- Does your application contain building blocks that customers would like to re-arrange based on their specific needs?

  SOA enables the benefit of Business Process Modeling or service orchestration. Building a layer of abstraction or services over existing business processes provides product teams and customers the ability to adopt these processes to meet their business needs.

Your level of adoption may vary from that of other product teams . Knowing and understanding the benefits of integration, reuse, and service orchestration can help determine the depth you go to in building your service-oriented applications.

Related Information

- Service Architecture
- Service Design and Project Guidelines
- Creating and Maintaining Services
- Services in Detail
- Using Services

Using Services

Overview

Contents

- Co-located Service Invocation
  - Calling Methods and Services
    - queryDataSource Client Usage
    - processDataSource Client Usage
    - InvokeServiceBatch Client Usage
  - Using Factories
  - Using Filters
    - Example: Filtering a Parent Service View Object Using Child Criteria
  - Using Domain Value Sets
- Using Service API in BPEL
  - Creating PartnerLinks for Services
  - Calling Services
  - Creating Input XML for QueryDataSource
- Using Client Proxy to call Service APIs
  - JDeveloper
  - Apache Axis
- Understanding the Service Response
  - Retrieving Information from ServiceMessage
- Related Information

Co-located Service Invocation

Calling Methods and Services

The service interface contains the following three core methods:

- `queryDataSource`: Use this method to query data from a service's data source.
- `processDataSource`: Use this method to insert, update, merge, or delete data into and from a service's data source.
- `invokeServiceBatch`: Use this method to bundle multiple operations into a single request/response message pair.

Note: These methods are also used to implement additional operations such as, approve, cancel, close, and copy.

queryDataSource Client Usage

The `queryDataSource` method is used to query data from a service's data source. It returns a root data object, which is a generic DataObject containing an attribute named for the data source. This attribute contains the list of service data objects (SDOs). The following is an example of using the `queryDataSource` method to print the first five purchase orders with the description "Transistors":

```
//print first 5 purchase orders with description=
//Transistors from purchase order service
ServiceImpl svcImpl = ....
OADBTransactionImpl trxn = (OADBTransactionImpl)svcImpl.getOADBTransaction();
DataObjectFactory factory = trxn.getDataObjectFactory();
ExpressionFilter filter = (ExpressionFilter)factory.createDataObject
  (PurchaseOrderFilter.QUALIFIED_NAME);
filter.addValueExpression("Description", ValueExpression.EQUAL, "Transistors");
DataCriteria dc = (DataCriteria)factory.createDataObject(DataCriteria.QUALIFIED_NAME);
dc.setFetchSize(new Number(5));
dc.setFilter(filter);
DataObject ret = svcImpl.queryDataSource("PurchaseOrder", dataCriteria, null);
List purchaseOrders = (ret==null)?null:(List)ret.get("PurchaseOrder");
if(purchaseOrders != null)
{
  DataObject dataObject;
  for (int i = 0; i < purchaseOrders.size(); i++)
  {
    dataObject = (DataObject)purchaseOrders.get(i);
    System.out.println(dataObject);
  }
}
```

processDataSource Client Usage

The `processDataSource` method is used to upload data. It processes and returns a root data object, which is a generic DataObject containing an attribute named for the data source. This attribute contains the list of service data objects (SDOs). Clients can use this method to insert, update, merge or delete data into and from a service's data source. The following is an example of using `processDataSource` to create a purchase order with a line:

```
//create a purchase order with a line
ServiceImpl svcImpl = ....
OADBTransactionImpl trxn  = (OADBTransactionImpl)svcImpl.getOADBTransaction();
DataObjectFactory factory = trxn.getDataObjectFactory();
PurchaseOrder po =
  (PurchaseOrder)factory.createDataObject(PurchaseOrder.QUALIFIED_NAME);
po.setDescription("...");
po.setBuyerId(...);
...
PurchaseOrderLine line = (PurchaseOrderLine)factory.createDataObject
  (PurchaseOrderLine.QUALIFIED_NAME);
line.setLineNumber(new Number(1));
```

```
List lines = new ArrayList(1);
lines.add(line);
po.setPurchaseOrderLine(lines);
List pos = new ArrayList(1);
pos.add(po);
DataObject root = factory.createDataObject
  (PurchaseOrderServiceRoot.QUALIFIED_NAME);
root.set("PurchaseOrder", pos);
ProcessControl ctl =
  (ProcessControl)factory.createDataObject(ProcessControl.QUALIFIED_NAME);
ctl.setReturnMode(ProcessControl.RETURN_KEY);
DataObject ret = svcImpl.processDataSource
  ("PurchaseOrder", DataProcessorService.OPERATION_CREATE, ctl, root);
List poNumbers = (ret==null)?null:(List)ret.get("PurchaseOrder");
if(poNumbers != null)
{
  PurchaseOrder retPO;
  for (int i = 0; i < poNumbers .size(); i++)
  {
    retPO = (PurchaseOrder )poNumbers .get(i);
    System.out.println(retPO.getPoNumber());
  }
}
```

Note: When sending a `DataObject` tree to be processed by the `processDataSource()` method, you do not have to set the parent primary key on the children SDOs because it is done automatically by OA Framework.

InvokeServiceBatch Client Usage

The `InvokeServiceBatch` method allows clients to batch method invocations in one method call. For example:

```
DataObjectFactory fac = ...
BatchRequest  batchReq = (BatchRequest)fac.createDataObject(BatchRequest.QUALIFIED_NAME);

List myMethod1Params = new ArrayList();
myMethod1Params.add(new Number(1));
myMethod1Params.add("Hello");
batchReq.addMethodRequest("MyRequest1", "MyMethod1", myMethod1Params);

List myMethod2Params = new ArrayList();
myMethod2Params.add(new Number(13));
myMethod2Params.add(new Boolean(Boolean.TRUE));
batchReq.addMethodRequest("MyRequest2", "MyMethod2", myMethod2Params);

BatchResponse batchResponse = svcImpl.invokeServiceBatch(batchReq);
```

Using Factories

`DataObjectFactory` is used to create data objects. All data objects must be created through factory because attempting to 'new' the data object directly causes an exception to be thrown when trying to operate on the data object. The following is an example of creating a data object through factory:

```
ServiceImpl svcImpl = ....
OADBTransactionImpl trxn = (OADBTransactionImpl)svcImpl.getOADBTransaction();
DataObjectFactory factory = trxn.getDataObjectFactory();
PurchaseOrder po = (PurchaseOrder)factory.createDataObject
  (PurchaseOrder.QUALIFIED_NAME);
DataCriteria dc =
  (DataCriteria)factory.createDataObject(DataCriteria.QUALIFIED_NAME);
```

Using Filters

Example: Filtering a Parent Service View Object Using Child Criteria

Once you create a parent SVO expression filter, a client can then use that filter at runtime. The ability to filter the parent service view object based on child criteria is embedded in the `queryDataSource` method on `oracle.jbo.server.ApplicationModuleServiceImpl`. The following example shows how to use a parent SVO expression filter to get all purchase orders that have a line price greater than 100:

```
DataCriteria dc =
  (DataCriteria)factory.createDataObject(DataCriteria.QUALIFIED_NAME);
PurchaseOrderFilter filter =
  (PurchaseOrderFilter)factory.createDataObject
  (PurchaseOrderFilter.QUALIFIED_NAME);
PurchaseOrderLineFilter lineFilter =
  (PurchaseOrderLineFilter)factory.createDataObject
  (PurchaseOrderLineFilter.QUALIFIED_NAME);
lineFilter.addUnitPrice(ValueExpression.GREATER_THAN, new Number(100));
filter.addPurchaseOrderLines(lineFilter);
dc.setFilter(filter);
java.util.List list = svcImpl.queryDataSource("PurchaseOrders", dc);
```

Note: You may override the predicate SQL statement that is constructed from the child filter object by overriding the `applyChildFilter` method on `oracle.jbo.server.ViewObjectServiceImpl`.

Using Domain Value Sets

In simple terms, a "domain value set" is a view object that defines all the valid values for a column. To learn how to define a domain value set and a domain value set usage, refer to the Domain Value Sets and Define Domain Value Set Usages topics in the Creating and Maintaining Services document. This section discusses

the two ways to use a domain value set.

Note: Define a domain value set view object (DVO) as a queryable data source to query the values in a domain value set based on usage 2 below.

Usage 1

A client may use domain value sets for validation and derivation. For example, when a service uploads data by calling `processDataSource`, the domain value sets are fired to make sure that the related attribute values are valid. In some cases, an attribute may be derived from another attribute.

To illustrate, suppose a PurchaseOrder has a SupplierDomain defined, which specifies the valid suppliers. PurchaseOrder also has SupplierName and SupplierId attributes, which are involved in the SupplierDomain. When a service uploads a PurchaseOrder, the SupplierDomain domain value set is fired to make sure the inbound SupplierName and/or SupplierId are valid. If either SupplierName or SupplierId is null in the inbound attributes (but not both), the supplier name can be derived from the supplier ID or the supplier ID can be derived from the supplier name in the SupplierDomain.

Usage 2

A client may use a domain value set to return a list of valid values for an attribute.

To accomplish this, call `queryDataSource(domainValueSetDataSourceName, dataCriteria, queryControl)` on `oracle.svc.DataSourceService` to get a list of valid values for the attribute. You need to provide a domain value set description to determine the `domainValueSetDataSourceName` needed by the `queryDataSource` method. You can find the domain value set description and domain value set name on `oracle.svc.description.ServiceDescription` and `oracle.svc.description.AttributeDescription`.

The following code example gets the domain value set data source for a domain value set defined on attribute BuyerName of PurchaseOrder.

```
List descs = svcImpl.queryDataSource("SvcServiceDescriptions",
  (DataCriteria)null, null);
ServiceDescription svcDesc = (ServiceDescription)descs.get(0);
DataObjectDescription poDesc =
  svcDesc.getDataObjectDescription(PurchaseOrder.QUALIFIED_NAME);
AttributeDescription attrDesc =
  poDesc.getAttributeDescription("BuyerName");
String domainValueSetName = attrDesc.getDomainValueSetName();
if(domainValueSetName != null)
{
  DomainValueSetDescription domainDesc =
    poDesc.getDomainValueSetDescription(domainValueSetName);
  String domainSdoQName = domainDesc.getDataObjectQualifiedName();
  List dsNames = svcDesc.getDataSourceNames(domainSdoQName);
  if(dsNames != null)
  {
    println(out, "Data source for the domain value set "
      +domainValueSetName+" defined on BuyerName are: ");
    for(int i=0; i<dsNames.size(); i++)
    {
      println(out, dsNames.get(i));
    }
  }
}
```

Using Service API in BPEL

This section discusses:

- Creating PartnerLinks for Services
- Calling Services
- Creating Input XML for QueryDataSource

Note: The information provided is intended to help you with utilizing services in your BPEL flows. You should be familiar with both BPEL and services.

Creating PartnerLinks for Services

To use services or web services in a BPEL flow, you must create a *PartnerLink* to represent it.

Note: *PartnerLink* is BPEL's concept of describing a webservice endpoint.

To create a *PartnerLink*:

1. Right click on the **Services** lane of BPEL designer and choose **Create PartnerLink**.
2. Enter the url for your service's wsdl.

   Tip: You can get the url for your service's wsdl from the "InterfaceRepository" user interface.

You must also define the following properties for the *PartnerLink* as they are needed for runtime authentication:

Property	Value
`wsseUsername`	*<username>*
`wssePassword`	*<password>*
`wsseHeaders`	credentials
`wsseOASIS2004Compliant`	true

Note: Set the values for `wsseUsername` and `wssePassword` according to your environment.

When using the Integration Repository user interface, you have controls for tasks such as deploying the service and granting access to individual users.

Calling Services

To call your service from the BPEL flow, drag and drop an *Invoke* activity into the flow and then choose which *PartnerLink* and which operation to use. You can also create input and output variables for the operation.

During runtime, the content of the input variable is passed to the service. The result of the service call is populated in the output variable.

Note: You must initialize the input variable before the *Invoke* activity. Keep in mind that everything in BPEL is XML, including the content of variables.

The BPEL designer provides various ways of manipulating variables, including the Transform and Assign activities:

- **Transform** - Use this activity to perform XSLT transformations from one variable to another.

  Tip: You don't need to know the details of XSLT syntax and the BPEL designer provides a user interface to for you use to define the transformation rules.

- **Assign** - Use this activity to create various operations such as copy, append, and remove.

The Following is an example of initializing the input variable for `PurchaseOrderService_GetPurchaseOrder`:

```
<assign name="Assign_2">
  <copy>
    <from expression="'1'"/>
    <to variable="Invoke_2_PurchaseOrderService_GetPurchaseOrder_InputVariable"
      part="body"
      query="/ns2:PurchaseOrderService_GetPurchaseOrder/poNumber"/>
  </copy>
</assign>
```

Note: Remember, you do not have to write the above xml. The BPEL designer provides the user interface for you to define the copy operation.

Creating Input XML for QueryDataSource

You can use the copy operation to copy an xml fragment into a variable. The following is an example of a copy operation demonstrating this technique:

```
  <copy>
    <from>
      <oans:PurchaseOrderService_QueryDataSource xmlns:
             xsi="http://www.w3.org/2001/XMLSchema-instance"
           xmlns:xsd="http://www.w3.org/2001/XMLSchema"
           xmlns:oans="http://xmlns.example.com/apps/fnd/framework/toolbox/tutorial">
        <dataSourceName>PurchaseOrder</dataSourceName>
        <dataCriteria>
          <FetchIndex>0</FetchIndex>
          <FetchSize>15</FetchSize>
          <LastFetchFull>false</LastFetchFull>
          <Filter xsi:type="ns:PurchaseOrderFilter"
              xmlns:ns="http://xmlns.example.com/apps/fnd/framework/toolbox/tutorial">
            <ConjunctionOperator>And</ConjunctionOperator>
            <Negation>false</Negation>
            <Expression xsi:type="ns:MultiValueExpression"
                xmlns:ns="http://xmlns.example.com/svc/expression">
              <AttributeName>PoNumber</AttributeName>
              <Operator>Between</Operator>
              <Value xsi:type="xsd:decimal">999999999</Value>
              <Value xsi:type="xsd:decimal">1000000010</Value>
            </Expression>
          </Filter>
          <PartialAttribute>PoNumber</PartialAttribute>
          <PartialAttribute>Description</PartialAttribute>
        </dataCriteria>
      </oans:PurchaseOrderService_QueryDataSource>
    </from>
    <to variable="PurchaseOrderService_QueryDataSource_InputVariable"
      part="body"/>
  </copy>
```

Important: The current BPEL (Version 10.1.3) has limited support for xsi:type construct. There are also problems in XDK that may cause BPEL flow compilation issues. Because of the usage of xsi:type in the above xml and the problems in XDK, the BPEL flow may contain compilation errors.

As a workaround, you can populate the BPEL variable from a file. The following is the content of the file:

```
  <oans:PurchaseOrderService_QueryDataSource
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:oans="http://xmlns.example.com/apps/fnd/framework/toolbox/tutorial">
    <dataSourceName>PurchaseOrder</dataSourceName>
    <dataCriteria>
      <FetchIndex>0</FetchIndex>
      <FetchSize>15</FetchSize>
      <LastFetchFull>false</LastFetchFull>
      <Filter xsi:type="ns:PurchaseOrderFilter"
          xmlns:ns="http://xmlns.example.com/apps/fnd/framework/toolbox/tutorial">
        <ConjunctionOperator>And</ConjunctionOperator>
        <Negation>false</Negation>
        <Expression xsi:type="ns:MultiValueExpression"
            xmlns:ns="http://xmlns.example.com/svc/expression">
          <AttributeName>PoNumber</AttributeName>
          <Operator>Between</Operator>
          <Value xsi:type="xsd:decimal">999999999</Value>
          <Value xsi:type="xsd:decimal">1000000010</Value>
```

```
        </Expression>
      </Filter>
      <PartialAttribute>PoNumber</PartialAttribute>
      <PartialAttribute>Description</PartialAttribute>
    </dataCriteria>
  </oans:PurchaseOrderService_QueryDataSource>
```

The following is another variation of how to use the copy operation to populate the BPEL variable from a file:

```
  <copy>
    <from expression="ora:doc('mydata.xml')"/>
    <to variable="PurchaseOrderService_QueryDataSource_InputVariable"
        part="body"
        query="/ns2:PurchaseOrderService_QueryDataSource"/>
  </copy>
```

Note: The above copy operation will copy the content of the file *mydata.xml* into the variable `PurchaseOrderService_QueryDataSource_InputVariable`. This file must reside in the "bpel" subdirectory of the project.

Tip: Although the xml input for `QueryDataSource` appears complicated, there is no need to write this xml by hand. Use the API Tester to create a test script for this test case and then copy the *InputParams* portion of your test suite xml file to *mydata.xml*.

Using Client Proxy to call Service APIs

JDeveloper

Step 1: Client Proxy generation

- Create a new Jdev application and Project
- Right-click on the project and select - New (Business -> Web Services -> Web Service Proxy)
- Provide the WSDL URL and click on "Next" button
- Click on "Finish" button
- Once the files are generated, right click on the web service file and click on "Secure Proxy"
- In each step, de-select all options and proceed to the end.

Step 2: Service Method Invocation

In the `main` method of the generated client proxy,

- Supply the apps username password:

  ```
  myPort.setUsername("<username>"); myPort.setPassword("<password>");
  ```

- Create ServiceBean header:

  ```
  ServiceBean_Header head = new ServiceBean_Header();
  ```

  You can optionally pass the following applications context properties:
  1. NLS_LANGUAGE
  2. RESPONSIBILITY_APPL_NAME
  3. RESPONSIBILITY_NAME
  4. SECURITY_GROUP_NAME
- Create the body object based on the method to be invoked (A class will be created corresponding to each service method/ each operation in the WSDL definition).

  ```
  <ServiceName>_<MethodName>_body = new <ServiceName>_<MethodName>();
  ```

- You should create the input objects for the method arguments. For example, an Employee object needs to be created for a method that takes Employee SDO as an argument. A class corresponding to each SDO reference will automatically be created for each SDO reference in the service interface.
- Invoke the service method. Following is how you should do it:

  ```
  <ServiceName>_<MethodName>_Response op =  myPort.<serviceName>_<methodName>(body, head);
  ```

  A response class (<ServiceName>_<MethodName>_Response) will be generated based on the Service method definition in the service interface.

Apache Axis

Note: This document is specific for Axis 1.4 only.

Step 1: Downloading Axis

Axis core is available here. The libraries *wss4j and xmlsec* should be used to secure the generated proxy. They can be downloaded from here.

Step 2: Generating files

- Download the WSDL of the BSO Web Service from the Integration Repository UI. You can use JDeveloper's *fetchWSDL* tool to do that.
- Generate the Client Proxy. Use the command "java org.apache.axis.wsdl.WSDL2Java " to generate the source code.

Step 3: Securing the Proxy

- A Callback class should be created that will be used as a placeholder for inputting the password. Sample Code is below:

  ```
  public class PWCallback implements CallbackHandler {
      public void handle(Callback[] callbacks) throws IOException,
                      UnsupportedCallbackException {
          for (int i = 0; i < callbacks.length; i++) {
  ```

```
            if (callbacks[i] instanceof WSPasswordCallback) {
                WSPasswordCallback pc = (WSPasswordCallback)callbacks[i];
                // set the password given a username
                if ("<username>".equals(pc.getIdentifer())) {
                    pc.setPassword("<password>");
                }
            } else {
                throw new UnsupportedCallbackException(callbacks[i], "Unrecognized Callback");
            }
        }
    }
}
```

- Create a WSDD file and point its Callback class to the Handler file generated in the previous step. Sample wsdd file can be found below:

```
<deployment xmlns="http://xml.apache.org/axis/wsdd/" xmlns:java="http://xml.apache.org/axis/wsdd/providers/java">
 <transport name="http" pivot="java:org.apache.axis.transport.http.HTTPSender"/>
  <globalConfiguration >
   <requestFlow >
    <handler type="java:org.apache.ws.axis.security.WSDoAllSender" >
     <parameter name="action" value="UsernameToken"/>
     <parameter name="user" value="sysadmin"/>
     <parameter name="passwordCallbackClass" value="PWCallback"/>
     <parameter name="passwordType" value="PasswordText"/>
    </handler>
   </requestFlow >
  </globalConfiguration >
</deployment>
```

  Note that you need to set the "user" and "passwordCallbackClass" parameters accordingly.

Step 3: Creating the Test Proxy and running it

The Java file (proxy) creates a SOAP Binding for your service and invokes the corresponding method in it. This needs to be created by you, as per Axis standards. As the proxy is already secured, it needs to be run with some additional JVM parameters. Example:*java -Daxis.ClientConfigFile=client_deploy.wsdd -Djava.protocol.handler.pkgs=com.sun.net.ssl.internal.www.protocol TestCase* where TestCase.java is your proxy file and client_deploy.wsdd is the WSDD file created in previou section.

Step 4: Analyzing the SOAP Messages

To analyze and edit the incoming as well as outgoing SOAP messages, you can either use TCPMon which is provided with Axis by default, or JDeveloper's HttpAnalyzer feature.

Understanding the Service Response

The Method Response object contains three elements. They are explained below.

- **Method output:** This corresponds to the object returned by the service method. The type of this element will be determined by the return type of the service method.
- **Warning messages:** This is a list of service message objects, each of which correspond to the OAExceptions registered through `OADBTransaction.putDialogMessage()`. Any warning or information message can be registered through this utility method at the server side. The BSO infrastructure collects all such messages into a list and sends via this element.
- **Exceptions:** This corresponds to the exceptions thrown during method invocation. The BSO infrastructure bundles together all OAExceptions thrown during service invocation and sends them in the response via this element.

Retrieving Information from ServiceMessage

The `oracle.svc.ServiceMessage` data object is used by the Service Interface engine to return information about errors and warnings when a client calls `Service.invokeServiceBatch` to invoke methods on a service.

When an exception is thrown in a service, the Service Interface engine converts it into a `ServiceMessage`. If the thrown `oracle.jbo.JboException` or `oracle.apps.fnd.framework.OAException` is a bundled exception, the Service Interface engine creates a list of `ServiceMessages` that correspond to the list of peer exceptions in the bundle. The details of a `JboException` or `OAException` are also converted into detail `ServiceMessages`.

The Service Interface engine returns a `ServiceMessage` data object at the method invocation level as part of `oracle.svc.MethodResponse`, and at the batch level as part of `oracle.svc.BatchResponse`. The convenience methods `findAllMessage` and `findAllErrorMessage` on `BatchResponse` allows you to get all messages returned at the method invocation level as well as at the batch level.

The following table summarizes a few of the key methods on `ServiceMessage` that you can use to retrieve information about Service error message. Refer to the `ServiceMessage` Javadoc for additional information.

Method	Description
`getCode`	This method returns a string message code that identifies the error or warning. This corresponds to the message code of the thrown `OAException`.
`getType`	This method returns one of the following message types that corresponds to the type of the `OAException` thrown: `TYPE_SEVERE`, `TYPE_ERROR`, `TYPE_WARNING`, and `TYPE_CONFIRMATION`.
`getText`	This method returns the translated text of the message that corresponds to the translated message text of the thrown `OAException`.
`getDetail`	If a `ServiceMessage` contains other detail `ServiceMessages`, this method returns those detail messages to provide further information to help explain the problem.

`getDataSourceName`	If a `ServiceMessage` is thrown while processing or getting a certain data source, this method returns that data source name. This method returns `null` if the message does not belong to a data object.
`getDataObject`	If a `ServiceMessage` represents a message for a certain data object, this method may return the actual fully populated data object or an instance of the data object with just the primary key set. This method returns `null` if the message does not belong to a data object.
`getDataObjectQualifiedName`	If a `ServiceMessage` represents a message for a certain data object, this method returns the data object qualified name. This method returns `null` if the message does not belong to a data object.
`getDataObjectAttributeName`	If a `ServiceMessage` represents a message for a certain data object, where specifically, it represents an error for a certain attribute on the data object, this method returns the attribute name. This method returns `null` if the message does not belong to a data object.
`getDataObjectAttributeValue`	If a `ServiceMessage` represents a message for a certain data object, where specifically, it represents an error for a certain attribute on the data object, this method returns the value of that attribute that caused the error. This method returns `null` if the message does not belong to a data object.
`getPath`	If a `ServiceMessage` represents a message for a certain data object, this method returns an XPath (xml path) string representation of the path to the data object that caused the error. For example, if an error occurred in the second Purchase Order Line of the first Purchase Order record, the following XPath string would be returned: `/PurchaseOrders.0/PurchaseOrderLines.1`
`getPathElement`	If a `ServiceMessage` represents a message for a certain data object, this method returns a list of `oracle.svc.ServiceMessage.PathElement` that represents the path to the data object that caused the error. The PathElement data object is an object representation of the XPath string returned by `getPath` and provides the element name and index for each segment in the path. For example, if an error occurred in the second Purchase Order Line of the first Purchase Order record, this method would return a list of two PathElements: the element name PurchaseOrder, whose index is 0 and the element name PurchaseOrderLines whose index is 1.

Related Information

- Creating and Maintaining Services
- Services in Detail
- Javadoc:
  - `oracle.jbo.server.ApplicationModuleServiceImpl`
  - `oracle.jbo.server.ViewObjectServiceImpl`
  - `oracle.svc.Service`
  - `oracle.svc.util.DataObjectFactory`
  - `oracle.svc.DataSourceService`
  - `oracle.svc.description.ServiceDescription`
  - `oracle.svc.description.AttributeDescription`

Entity Object and View Object Attribute Setters

Overview

There are a number of methods in the `oracle.apps.fnd.framework.server.OAEntityImpl` and `oracle.apps.fnd.framework.server.OAViewRowImpl` classes that let you programmatically set entity object and view object attribute values. This document explains each of the available options and suggests appropriate usage in different scenarios.

Note: This does not explain how to set entity object and view object attribute values declaratively. See Defaulting in Implementing the View for an overview of valid defaulting approaches.

Contents

This document is organized into two major sections according to where the attribute values are cached:

- Attribute Values Cached on the View Object
- Attribute Values Cached on the Entity Object

Prerequisite Reading

- Java Entity Objects
- View Objects in Detail - View Object Attribute Types and Caching (to learn about the different view object attribute types)

Attribute Values Cached on the View Object

Method Survey

For attributes whose values are stored at the view object layer (they are not mapped to entity object attributes), you can use the following `OAViewRowImpl` methods to set the attribute value.

Option 1: Basic Setters - setAttribute( )

```
1. setAttribute(String name, Object val)
2. setAttribute(int index, Object val)
```

These two methods exercise any programmatic validation that you define for them, and then in `super.setAttribute()`, they use a lookup mechanism to find and call the associated `set<AttributeName>()` method (note that you must call `super.setAttribute()` *after* your code).

```
3. set<AttributeName>() // Attribute setter in OAViewRowImpl subclass
```

This exercises any programmatic validation that you define for the method, and then calls `setAttributeInternal()`. Note that you must call `setAttributeInternal()` *after* your code.

All three of these methods mark the dependent view object's query collection as being dirty (`OAViewObjectImpl.isDirty()` returns **true** after you call them). See Advanced View Object Development Topics -> Entity Event Notification for additional information about the `isDirty()` check.

Option 2: setAttributeInternal( )

```
1. setAttributeInternal(int index, Object val)
```

This method performs any *declarative validation* specified for the view object attribute in the view object's XML file. As with `setAttribute()`, this method marks the dependent view object's query collection as being dirty.

Since this method is intended to perform declarative validation only, you should not override it (you don't need to since it shouldn't include programmatic validation); you should simply call it as needed.

Option 3: "Populate" Methods

```
1. populateAttribute(int index, Object value)
2. populateAttributeAsChanged(int index, Object value)
```

These methods populate the attribute with a value, but do not exercise any validation or affect the view object's query collection dirty flag.

You should not override and insert programmatic validation logic in `populate*` methods, as these methods are intended to simply

populate the attribute with a value.

Note: For *entity object layer attributes*, the `populateAttributeAsChanged(int index, Object value)` method behavior differs; see the Attribute Values Cached on the Entity Object section below.

The following example on a custom view row illustrates how to use this to set a table selector value without impacting the state of the view object:

```
public void setSelectFlag(String value)
{

  Do not call setAttributeInternal as usual in this method.
  // setAttributeInternal(SELECTFLAG, value);

  populateAttribute(SELECTFLAG, value);

}
```

Behavior Summary

- *`set*`* methods perform validation and mark the view object's query collection as being dirty. *`populate*`* methods simply populate the attributes without performing any validation or changes to the view object state.

Usage Notes

As a rule, use the basic setters (Option 1) unless you have the following special circumstances:

- If you need to bypass programmatic validation and invoke only declarative validation, then you may use `setAttributeInternal(int index, Object val)` instead.
- To populate an attribute value *without triggering validation or affecting the view object's query collection dirty flag*, use `populateAttribute(int index, Object value)`. For instance, as shown above, you can override a table bean's "selector" attribute setter method to delegate to `populateAttribute()`. This lets you store UI-specific state without changing the state of the VO. See Tables - Classic and Tables - Advanced for additional information about the "selector."

  Note: Never use the `populate*` methods to set *primary key* attribute values (this could have adverse consequences for composite associations). Use the basic setters for primary key attributes.

Attribute Values Cached on the Entity Object

For attribute values that are cached at the entity object level, you can use the following `OAEntityImpl` methods to set the attribute values

Note: For the most part, you'll generally set entity object values by calling the methods on the `OAViewObjectImpl` and `OAViewRowImpl` class described above (the VO methods ultimately delegate to the underlying entity object methods). When writing code in your `OAEntityImpl` subclass, however, you can call the following methods directly to set entity attribute values.

When you call an entity object method like `set<AttributeName>()`, your call impacts the following entity object state:

- **Validation state** -- checked by calling `OAEntityImpl.isValid()`
- **Post state** -- checked by calling `OAEntityImpl.getPostState()`
- **Transaction state** -- checked by calling `OAEntityImpl.getEntityState()`

Since BC4J maintains this additional state for entity objects (over and above the basic "dirty" flag maintained for view objects), the `OAEntityImpl` methods fine-grained control over how attribute values are set.

Method Survey

Option 1: Basic Setters - setAttribute( )

```
1. setAttribute(String name, Object val)
2. setAttribute(int index, Object val
```

These two methods exercise any programmatic validation that you define for them, and then in `super.setAttribute()`, they use a lookup mechanism to find and call the associated `set<AttributeName>()` method (note that you must call `super.setAttribute()` *after* your code).

```
3. set<AttributeName> // Attribute setter in OAEntityImpl subclass
```

This exercises any programmatic validation that you define for the method, and then calls `setAttributeInternal()` (note that you must call `setAttributeInternal()` *after* your code).

All three of these methods mark the entity object as being invalid and change the entity object post state and transaction state to **STATUS_NEW** or **STATUS_MODIFIED** as appropriate so that BC4J knows that the entity object has pending changes that need to be posted.

Finally, all three of these methods also mark the dependent view object's query collection as being dirty (`OAViewObjectImpl.isDirty()` returns **true** after you call them).

Option 2: setAttributeInternal( )

```
1. setAttributeInternal(int index, Object val)
```

This method performs *declarative validations* specified in the entity object's XML for the attribute.

As in `setAttribute()`, this method marks the entity object invalid, changes the post state and transaction state, and also marks the view object's query collection as being dirty.

Since this method is intended to perform declarative validation only, you should not override it (you don't need to since it shouldn't include programmatic validation); you should simply call it as needed.

Option 3: "Populate" Methods

Before reviewing the individual methods, it's important to start with a description of parameters that can be passed to the most granular `populate*` methods to which all the `populate*` method calls ultimately delegate.

Method Parameter	Description
`sendNotification`	Controls whether the entity attribute value change event is broadcast to the dependent view objects. The change event may affect the view object's query collection state (the "dirty" flag). See Advanced View Object Development Topics -> Entity Event Notification for additional information about this.
`markAsChanged`	When this parameter is set to **true**, the attribute is marked as *changed*. • BC4J posts changed persistent attributes to the database during the DML phase, but *only if the post state of the containing entity object's state is STATUS_NEW or STATUS_MODIFIED* as a consequence of inserting a new row or making updates with calls to `setAttribute()`, `setAttributeInternal()` or `set<AttributeName>()` on another attribute.
`saveOriginal`	Controls whether BC4J should save the original attribute fetched from the database in a separate store prior to making the value change. If you opt to save the original values: • BC4J compares the original attribute values with the database column values on lock to detect whether the cached value is stale. • BC4J preserves the original attribute value when a post or commit fails so these actions can be retried after the user corrects validation errors. If the original attribute values are *not* saved, then the current attribute values will be used by BC4J for these operations.

```
1. populateAttribute(int index, Object value)
```

The net behavior of this method is equivalent to calling:

```
 populateAttribute(index, value,
                   false,  // sendNotification
                   false,  // markAsChanged
                   false); // saveOriginal
```

```
2. populateAttributeAsChanged(int index, Object value)
```

The net behavior of this method is equivalent to calling:

```
 populateAttribute(index, value,
                   false,  // sendNotification
                   true,   // markAsChanged
                   false); // saveOriginal

3. populateAttribute(int index,
                     Object value,
                     boolean sendNotification,
                     Boolean markAsChanged)
```

This method is *deprecated*. It delegates to `populateAttribute(index, value, sendNotification, markAsChanged, false)`.

```
4. populateAttribute(int index,
                     Object value,
                     Boolean sendNotification,
                     Boolean markAsChanged,
                     Boolean saveOriginal)
```

- You should not override and insert programmatic validation logic in `populate*` methods, as these methods are intended to simply populate the attribute with a value.
- This method does *not* affect the entity object validation state, post state, or transaction state.
- The view object's query collection dirty flag value can be affected by the `sendNotification` flag value.

Recommended Parameter Values

- For OA Framework applications, set `sendNotification` to **false** so that the view object's query collection dirty flag is not affected by your change. When the view object's query collection becomes dirty, operations such as sorting within a table are not permitted (see Tables - Classic and Tables - Advanced for additional information).
- Set `markAsChanged` to **true** if you want to post the attribute value to the database together with other user changes during the DML phase.
- Set `saveOriginal` to **true** if you set `markAsChanged` to **true**. As a rule, to facilitate proper locking and cope with post/commit failures, the `saveOriginal` parameter should be set to **true** if you plan to post the changed attribute value.

Behavior Summary

- *`set*`* methods perform validation, mark the entity object invalid, change the entity object post state, change the entity object transaction state, and mark the view object's query collection as being dirty.
- *`Populate*`* methods simply populate the attribute values without performing any validation or changing the entity object states. However, the view object's query collection state (including its dirty flag) is affected by the `sendNotification` parameter value.

Usage Notes

As a rule, use the basic setters (Option 1) unless you have the following special circumstances.

- If you need to bypass programmatic validation and invoke only declarative validation, then you may use `setAttributeInternal(int index, Object val)` instead.
- The use of the `populate*` methods for entity object attributes is generally discouraged. That said, however, there are some exceptional cases where you may need to use these methods; the following describes these use cases and makes recommendations specific to attribute types (if these type names are unfamiliar, see View Objects in Detail - View Object Attribute Types and Caching).

  Note: For Java entity objects, never use `populate*` methods to set *primary key* attribute values (this could have adverse consequences for composite associations). Use the basic setters for primary key attributes. For PL/SQL entity objects with refresh-on-insert behavior, see PL/SQL Entity Objects - Creating Primary Keys for additional information.

  For all entity-derived attribute types, do not use `populateAttributeAsChanged(int index, Object value)` unless you really don't care about being able to perform a post/commit after a post/commit failure (so the `populateAttributeAsChanged()` method **saveOriginal = false**).

Entity-Derived Persistent Attributes

For *entity-derived persistent* attributes, in most cases, you won't need to call the `populate*` methods. To default these attribute values for a new row, it is better to use the basic setters and then call `OAViewRowImpl.setNewRowState(Row.STATUS_INITIALIZED)` to effectively treat the row as being "temporary" until the user makes a change. See View Object State Management Coding Standard M69 for additional information.

Exceptions

1. To implement refresh-on-insert for PL/SQL entity objects, you need to call `populateAttribute(index, value, false, false, false)`. This is required to sync up the attribute values with the database values immediately after a DML statement and before a commit. As mentioned above, see PL/SQL Entity Objects - Creating Primary Keys for additional information.

2. If you would like to populate an entity-derived persistent attribute value so it is posted together with other user-driven changes *without triggering validation or affecting the entity object validation, post, or transaction state or the view object's query collection dirty flag*, then use `populateAttribute(index, value, false, true, true)`. For instance, the OA Framework populates standard WHO and `ObjectVersionNumber` attribute values this way.

Entity-Derived Transient Attributes

For entity-derived transient attributes, in general, you should calculate and return the values in the entity object attribute getter (`get<AttributeName>()`) method instead of setting or populating the attribute.

- If you cannot use this approach for some reason, you can still use the basic setters -- but be aware that these calls will change the entity object validation, post, and transaction state and the view object's query collection dirty flag as described above. Also be aware that you should not store any UI state in an entity-derived transient attribute.

  Note: Entity-derived transient attribute values may become lost when view object row set is refreshed after a commit. If you need to make these values persist see Persisting Entity-Derived Transient Attribute Values After Commit for more information.

- Instead of the basic setters, you can use `populateAttribute(index, value, false, true, true)` (with `markAsChanged` parameter value set to **true**) to avoid triggering validation or affecting the entity object validation, post, or transaction state, or the view object's query collection dirty flag.

  Note: The transient attribute value is not mapped to any database table column, and therefore, will not get posted. Also, be aware that you should not store any UI state in an entity-derived transient attribute.

- Avoid using `populateAttribute(int index, Object value)` or `populateAttribute()` with the `markAsChanged` parameter value set to **false**.

Chapter 6: Advanced OA Framework Development Topics

Supporting the Browser Back Button

Overview

This document provides a list of target goals and describes the coding standards that you must follow to ensure robust browser Back button support.

Usability tests show that users rely heavily on the browser Back button. Unfortunately, this navigation preference introduces a series of potential failure points in transactional applications. For example, consider the following scenarios and potential problems in an OA Framework application that does not fully anticipate browser Back button navigation:

User Navigation Scenario	Problem
The user deletes a row from a table and the page redraws with a confirmation message indicating that the row has been deleted (the row no longer appears in the table). The user then presses the browser Back button, and the page renders with the row still present in the table. The user then tries to delete the row a second time.	The browser caches page contents. If the user performs an action that changes data state, and then uses the browser Back button, the browser's page cache doesn't reflect the correct middle tier state (in this case, that a row no longer exists). An attempt by the user to delete, or otherwise transact, this deleted row in the cached page would likely result in an ugly runtime application error. A Back button compliant page would detect the attempt to transact a deleted row and fail gracefully by displaying a user-friendly message indicating that the row has already been deleted.
At the end of a shopping cart checkout process, the user selects a "Submit Order" button to purchase items. For whatever reason, the user navigates from the confirmation page back to order submission page using the browser Back button and selects the "Submit Order" button a second time (perhaps she thinks she can make a change to an order quantity and have it properly reflected on the original order).	This scenario is similar to the one described above, however, the unguarded action could result in "successful" duplicate transaction (the order might be created twice, which is unlikely to be the user's expectation). A Back button compliant page would detect an attempt to submit the same order twice and fail gracefully by displaying a user-friendly error message indicating that the order had already been placed.
The user navigates from Page 1 to Page 2, and then back to Page 1 using the browser Back button. The user presses a form submit component in Page 1, and an unhandled exception (`NullPointerException`, `IndexOutOfBoundsException` and the like) is thrown.	The OA Framework "Page 1" expects the web bean hierarchy to be in a certain state, and/or it expects transaction, session or BC4J object state values to be present. When navigating with the browser Back button, this state may be lost. If the page doesn't anticipate this, unhandled exceptions may display to the user. A Back button compliant page would anticipate the potential loss of state behind the scenes, and would either be capable of recreating the state so it can continue functioning normally, or fail gracefully by displaying a user-friendly error message indicating that the page cannot be accessed after navigating with the browser Back button.

Target Goals

To avoid the problems presented above as well as other similar problems, use these target goals as a guide to understanding what it means to *support* the browser Back button.

1. Provide consistent behavior in handling the browser Back button

2. Avoid severe, unexpected exceptions from state change

Goal 1: Provide Consistent Behavior in Handling the Browser Back Button

It is important that the application pages handle the browser Back button in a consistent manner. Use the following list of subgoals as a guide to achieve this consistency:

- **Allow basic, straightforward operations to be repeated unless technical limitations are present.**

  Using inquiry flows where the user performs a search and drills down to a detail record as an example; when the user navigates back and forth in the inquiry flow using the browser Back button, allow the search and drilldown actions to seamlessly work without an error.

- **Allow transactions to be repeated as long as the logical transaction is active.**

  A *logical transaction flow* in this context is a task comprised of one or more pages with a definitive end point (commit). Examples include a single page update, a multi-step create, and a delete action (with commit) for one or more selected rows. The user should be able to navigate between pages in a logical transaction using the browser Back button, and perform operations without an error as long as the logical transaction is still in progress.

- **Avoid ambiguous transactions, transactions on already deleted data, or transactions that could result in unintended user operations.**

  To ensure transaction integrity for any of these transactions, if possible restrict the action after a browser Back button press. For example, Create flows should not be re-entrant when a form submit action is issued after a browser Back button press. This is because it is not clear whether the user wants to update the previously created record or create a new duplicate record. An error dialog should be shown in this case. Operations on already deleted data should also result in an explicit error dialog.

  If you have a special Update flow where the user selects rows in one page and updates the selected rows in another page; choose the restrictive policy as in the Create flow if your Update page does not not explicitly tell the user which rows it is updating or if you do not have a mechanism to remember the selections and repeat the same exact update operation.

  Tip: For the browser Back button support, although it is more desirable to allow page re-entry for obvious use cases, when in doubt, it is better to restrict page re-entry rather than show unhandled exceptions or perform unintended transactions that could be harmful to users.

Goal 2: Avoid Severe, Unexpected Exceptions From State Change

Your minimum goal should be to avoid severe, unexpected exceptions such as `NullPointerException` or `IndexOutOfBoundsException`, which result in an unreadable message to the user. A well-formatted error dialog with a readable user message text should be displayed instead.

See Use Case-Specific General Standards for more detail information about these goals.

Contents

- Survey of Relevant Tools and Behaviors
  - Unique Page Instance ID and Page State Caching
  - Method for Detecting Browser Back Button Navigation
  - Web Bean Hierarchy Synchronization
  - UI "Transaction Unit"
  - View Object Primary Key Comparison
  - BC4J Stale Data Check During Locking
  - Standard Error Dialogs
  - AM State Required Flag (Deprecated)
- Coding Standards
  - General Standards
    - 1. Define View Object Primary Keys (Model Coding Standard M39)
    - 2. Avoid View Object State Assumptions (Model Coding Standards M38, M37)
    - 3. Modify Web Bean Properties Exclusively in processRequest (Controller Coding Standards C6, C15)
    - 4. Avoid Unconditional View Object/Transaction State Initializations (Controller Coding Standard C32)
    - 5. Correctly Configure Dialog Pages (Controller Coding Standard C30)
    - 6. Correctly Communicate State Across Pages (Controller Coding Standards C20, C17)
    - 7. Conditionally Disallow Form Submit
  - Use Case-Specific Standards
- Testing Back Button Navigation

Prerequisite Reading

This document assumes that you are familiar with the following content:

- Anatomy of an OA Framework Page
- OA Framework State Management
- Implementing the Model
- Implementing the View
- Implementing the Controller

Survey of Relevant Tools and Behaviors

Before diving into the coding standards, it's important to understand the various tools and built-in behaviors provided by the OA and BC4J frameworks to help you build Back button compliant pages. The Coding Standards in the next section provide specific instructions for applying these to the Back button problem.

Unique Page Instance ID and Page State Caching

OA Framework now identifies each discrete page request -- including multiple requests of the same page -- with a unique page instance ID (an incrementing page counter parameter is added to the form action URL and each link in the generated page). When a form submit occurs *after* the user selects the browser Back button, OA Framework uses the page counter indicator to detect the browser Back button navigation. Previously, OA Framework detected Back button navigation by comparing URLs.

Additionally, new methods have been added to `oracle.apps.fnd.framework.webui.OAPageContext`, which allows you to save and retrieve page state: `savePageState()`, `getPageState()` and `removePageState()`. See the Javadoc for additional information.

Method for Detecting Back Button Navigation

The `OAPageContext.isBackNavigationFired()` method lets you explicitly detect browser Back button navigation for the current request. This leverages the unique page instance ID described above.

See the `OAPageContext` Javadoc for additional information about using this method.

Web Bean Hierarchy Synchronization

As described in Chapters 2 and 3, OA Framework recreates the web bean hierarchy when handling an HTTP `POST` if the original, cached hierarchy has been lost. Specifically, it reenters `processRequest()` for the entire web bean hierarchy in the following cases:

To recover a lost web bean hierarchy:

- A `POST` request invokes activation in a new servlet session, or on a recycled application module instance.
- A `POST` request is failed over to a new JVM.

To synchronize the client UI state with the middle tier web bean state:

- A `POST` request is issued on a page after the user navigates to it using the browser Back button. OA Framework detects this case by calling the `OAPageContext.isBackNavigationFired()` method.
- A `POST` request is issued from an `oracle.apps.fnd.framework.webui.OADialogPage` to the originating page that opened the dialog.

The ability to recreate the web bean hierarchy when the form is submitted is an important affordance because it ensures that the user's work can proceed as if there were no disruption or inconsistency behind the scenes. However, it imposes strict coding standards to ensure that any page that supports Back button access can be recreated under these circumstances.

UI "Transaction Unit"

The `oracle.apps.fnd.framework.webui.TransactionUnitHelper` class lets you delineate a user interface "transaction unit" for the purpose of identifying inappropriate navigation patterns after a transaction commits. This is particularly helpful in those cases where multiple discrete UI tasks (each with their own commit points) share the same root UI application module -- and therefore the same database transaction. With the transaction unit identifier, you can indicate when a specific UI task begins and ends, and if the user navigates in an unexpected manner such as using the browser Back button. You can check the transaction unit status and react to ensure that inappropriate actions are not performed and unwanted data in the BC4J cache is not committed. For example:

1. The user searches for employees in an Employee Summary page and selects a **Create Employee** button.

   You begin a *create employee* transaction unit.

2. The user enters some data in the Create Employee page and selects **Apply** to commit their changes.

   You end the *create employee* transaction unit. As part of the **Apply** button handling, the Create Employee page forwards to the summary page where a confirmation message is displayed.

3. The user then selects the browser Back button to return to the Create Employee page where they attempt to make data changes and select the **Apply** button a second time.

   By checking the transaction unit status OA Framework displays an error dialog because updates to newly created rows are not allowed in this example module.

   Note: There is no current *create employee* transaction unit because the user didn't navigate using the **Create Employee** button, which would have triggered the start of a new create transaction unit.

The Use Case-Specific Standards below provide detailed instructions for leveraging this technique.

View Object Primary Key Comparison

Assuming a view object has a primary key or a ROWID attribute, OA Framework automatically checks the submitted form data during the `processFormData()` post-back process to look for "stale data." If the page row's primary key or ROWID doesn't match the view object's primary key or ROWID *for the same row* (if the row set has been changed or the row was deleted), OA Framework displays a standard "stale data" error dialog.

If a primary key is defined, instead of relying on a ROWID attribute, OA Framework attempts to find a matching row in the result set using the primary key. If found, a "stale data" error is avoided and the user can continue working.

Note: OA Framework searches for matching rows *only within the current view object result set*. If the view object's WHERE clause settings restrict the result set, a match may not be possible and the "stale data" error displays.

Usage Notes

- The "stale data" detection mechanism is primarily designed to catch operations on rows that have already been deleted.
- To optimize performance, OA Framework does not perform stale data checking for read-only tables that have no form fields. If, for any reason, you want the stale data check to be enforced for a read-only table, add a dummy updateable hidden field (**formValue** style) last in the table region.
- To get more information on the root cause of the "stale data" error, either enable Developer Mode or Diagnostic Mode. When these modes are enabled, additional developer-facing content is displayed in the dialog message to help diagnose possible coding errors. See Testing OA Framework Applications for information about enabling these modes.
- The "stale data" error can also be displayed unexpectedly because of coding mistakes related to view object initialization. Please see avoid unconditional view object/transaction state initialization below for additional information.
- **Known Issue:** The current view object primary key comparison logic tends to be overdefensive; in certain scenarios, a "stale data" error may be displayed when it isn't necessary. For example, executing a search after browser Back button navigation with new query criteria results in a "stale data" error. To overcome this problem for read-only tables, you can use the `setSkipProcessFormData(pageContext, true)` API on `oracle.apps.fnd.framework.webui.beans.table.OATableBean` or `oracle.apps.fnd.framework.webui.beans.table.OAAdvancedTableBean`.

BC4J Stale Data Check During Locking

When BC4J attempts to lock a row before the database post operation using the default OA Framework locking scheme, it automatically tries to determine if the row has been deleted or changed by another user since being queried for the current user:

- If the row was deleted, BC4J throws an `oracle.jbo.RowAlreadyDeletedException`.
- If changes are detected, BC4J throws an `oracle.jbo.RowInconsistentException`.

In both cases, OA Framework automatically converts these low-level exceptions to user-friendly error messages.

This check is also helpful when users attempt to save changes on stale data after using the Back button.

For additional information about BC4J Locking, see Implementing Java Entity Objects in Chapter 5.

Standard Error Dialogs

OA Framework includes several standard error dialogs that can be leveraged in your code as shown in the example below:

- `NAVIGATION_ERROR` - This error message inform users that they cannot proceed as a consequence of navigating with the browser Back button. Recovery navigation instructions direct users to return to the global Home page and restart the transaction.
- `FAILOVER_STATE_LOSS_ERROR` - This error message inform users that they cannot proceed as a consequence of an expired user session or system failure. Recovery navigation instructions direct users to return to the global Home page and restart the transaction.
- `STATE_LOSS_ERROR` - This generic combination of the two more specialized error messages inform users that they cannot proceed as a consequence of navigating with the browser Back button, an expired user session, or a system failure. Recovery navigation instructions direct users to return to the global Home page and restart the transaction.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  ...

  // Display a standard error message related to state loss due to Back button
  // navigation or failover as appropriate.
  if (pageContext.isBackNavigationFired(true))
  {
    pageContext.redirectToDialogPage(new OADialogPage(NAVIGATION_ERROR));
  }
  else
  {
    pageContext.redirectToDialogPage(new OADialogPage(FAILOVER_STATE_LOSS_ERROR));
  }
}
```

You can also use the more generic `pageContext.redirectToDialogPage(new OADialogPage(STATE_LOSS_ERROR))`. However, we strongly recommend that you create context-specific messages for improved usability and customer support.

AM State Required Flag (Deprecated)

The application module `AM State Required` flag is deprecated and should not be used for new development. If you have pages that leverage this feature, and they behave correctly as described in the coding standards below, there is no need to change your implementation.

Coding Standards

Although all the OA Framework coding standards are consolidated as checklist items in the Coding Standards chapter, this section provides more detailed information about each standard and how to apply it to your code.

Note: These topics also link to the summary checklist items in Chapter 8.

The standards are organized into the following two broad categories:

- General standards that apply to all use cases.
- Use-case specific standards that describe how to handle the Back button in specific application scenarios, such as how to handle a Delete action in a read-only summary page, or how to handle a multi-step Create action related to an updateable page.

General Standards

1. Define View Object Primary Keys (Model Coding Standard M39)

To enable the view object primary key comparison described above, all view objects must have a designated primary key or a ROWID surrogate. A primary key is preferable.

Exception: If a read-only view object simply does not have a logical primary key, there is no need to create one. For example, a single column view object that queries the sum of a column's value does not have a logical primary key.

If you need information about how to define a primary key for your view object, see Implementing the Model. To identify a ROWID if you would not otherwise have a primary key, simply select the `ROWID` pseudo column in your view object SQL and name the corresponding view attribute **RowId** or **RowId**.

2. Avoid View Object State Assumptions (Model Coding Standards M38, M37)

Never make assumptions about the state of your view object:

- Don't always assume that your view object has rows. Perform an explicit check for null for the rows returned from the view object.
- For view objects used for search pages with dynamic WHERE clauses, always clear preexisting WHERE clause parameters before executing a query. This ensures that previous settings do not linger and interfere with the current query.
- For dynamically created view objects, always *find* before simply creating them.

The following code examples illustrate these rules:

```
// Bad Code

Row[] rows = vo.getAllRowsInRange();
rows[0].getAttribute(...);   <-- assumes the rows exist

// Good Code

Row[] rows = vo.getAllRowsInRange();
if (rows == null)   <-- allows for the possibility that rows cannot be found
{
  // Take proper action.
}
rows[0].getAttribute(...);
```

Whenever you dynamically set the WHERE clause (whether it is to a null or non-null value), always clear any parameter bindings first:

```
// Bad Code

vo.setWhereClause("empno < 10");
vo.executeQuery();
...
vo.setWhereClause("empno = :1");
vo.setWhereClauseParam(0, <some value>);
vo.executeQuery();
...
vo.setWhereClause(null);
vo.executeQuery();

// Good Code

vo.setWhereClause("empno < 10");
vo.setWhereClauseParams(null);
vo.executeQuery();
...
vo.setWhereClause("empno = :1");
vo.setWhereClauseParams(null);
vo.setWhereClauseParam(0, <some value>);
vo.executeQuery();
...
vo.setWhereClause(null);
vo.setWhereClauseParams(null);
vo.executeQuery();
```

If you dynamically create a view object or view object attribute, always check to see whether an object with that name already exists:

```
// Bad Code

ViewObject vo = am.createViewObject("MyVO", ...);
vo.addDynamicAttribute("MyAttr");

// Good Code

// First check whether the VO with the same name exists or not.

ViewObject vo = am.findViewObject("MyVO");

if (vo == null)
{
  vo = am.createViewObject("MyVO", ...);
}

// Check whether the attribute with the same name exists or not.
// Note that for attributes, we need to use try-catch clause.
```

```
if (vo.lookupAttributeDef("MyAttr") == null)
{
  vo.addDynamicAttribute("MyAttr");
}
```

3. Modify Web Bean Properties Exclusively in processRequest (Controller Coding Standards C6, C15)

Never add web bean modification logic in `processFormData()` or `processFormRequest()`. If you need to modify the web bean hierarchy or change web bean properties while handling a form submit request, you may:

- Leverage property SPEL bindings and partial page rendering as described in the Dynamic User Interface topic in Chapter 4.
- Forward back to the page using the `OAPageContext.setForward*()` methods with the `retainAM` parameter set to **true**, and then execute your bean manipulation logic in `processRequest()`.

If OA Framework needs to recreate the web bean hierarchy for any reason as described above, all the appropriate logic to support this must be included in the `processRequest()` methods. Furthermore, any information that drives changes to the web bean hierarchy (including individual web bean properties), must be added to the request as a **URL parameter**. In other words, if you programmatically change the web bean properties or hierarchy in any way when you forward to a page, then the information that you check to decide what to do with the web beans must be specified in the "parameters" `com.sun.java.util.collections.HashMap` of the forward (`OAPageContext.setForward*()`) method call as shown in the following example:

```
import com.sun.java.util.collections.HashMap;
...

// As a result of a button press, return to the current page setting
// a URL parameter value to be used to modify the page layout.

public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormRequest(pageContext, webBean);

  if (pageContext.getParameter("someButton") != null)
  {
    HashMap params = new HashMap(1);
    params.put("showRegion", "RegionA");
    pageContext.setForwardURLToCurrentPage(params, // added to the request as URL parameters
                                           true, // retain the AM
                                           OAWebBeanConstants.ADD_BREAD_CRUMB_NO,
                                           OAWebBeanConstants.IGNORE_MESSAGES);

  }
}


public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  if ("RegionA".equals(pageContext.getParameter("showRegion")))
  {
    // Change your layout accordingly...
  }
}
```

Note: Using `OAPageContext.putParameter()` or hidden fields isn't sufficient in this case because these values are *not* added to the request as URL parameters.

The following example describes how a key value passed in the `setForward*()` method `HashMap` parameter can be used to correctly rebuild the web bean hierarchy:

- Page A includes a dynamic section where you display different regions based on a user's poplist selection. Page A renders with **Region1** displayed by default.
- The user makes a poplist selection, and the poplist is configured to submit the form. Your code forwards back to the same page and displays **Region2** in the dynamic content section.

  The user's poplist selection value is used to decide which region to display. When you forward back to the page, you set this value in the "parameters" `HashMap` of the `OAPageContext.setForwardURLToCurrentPage()` method. This ensures that this value is added to the request as a URL parameter.

- The user changes her mind and makes a different poplist selection so your code forwards back to the page again and displays **Region3**.
- The user selects the browser Back button to return to Page A with **Region2** displayed.

  Note: The browser's HTML cache, which displays **Region2**, no longer matches the web bean hierarchy in the middle tier, which includes **Region3**.
- The user enters some values in the fields displayed in **Region2** and presses the a **Submit** button.

  OA Framework detects that there was a browser Back button navigation, and in response, rebuilds the web bean hierarchy to correctly include **Region2** with the user's data added to the request. This ensures that the middle tier is correctly synchronized with the browser's cache and the action proceeds without a failure.

4. Avoid Unconditional View Object/Transaction State Initializations (Controller Coding Standard C32)

Given that the `processRequest()` method can be reentered repeatedly as described in the Web Bean Hierarchy Synchronization section above, avoid unconditional view object and transaction state initializations.

Initialization operations include the following:

- Calling `executeQuery()` on a view object used for querying data from the database.
- Calling `setMaxFetchSize(0)` followed by calling `insertRow(row)` on a view object whose rows will be inserted manually, and not queried from the database.
- Calling `OAPageContext.putTransactionValue()`, `OAPageContext.putTransactionTransientValue()`, `OAViewObjectImpl.putValue()`, `OAApplicationModuleImpl.putValue()` or `OAPageContext.putSessionValue()` to create transaction or session-specific state.

You should avoid unconditional initialization for the following two reasons:

- When the `processRequest()` method is reentered, unconditional initialization can result in an undesirable loss of transaction state. For example, user updates made to transient view object attributes will be lost, and the view object current row and range will be reset.
- Redundant query execution and state initialization is not performant.

See View Objects in Detail: Initialization Guidelines for detailed instructions on how to properly perform the view object initializations described here.

Transaction State

If you maintain transaction state using a method such as `OAPageContext.putTransactionValue()`, always check whether the value exists before initializing it. For example:

```
if (pageContext.getTransactionValue("poTrxStep") == null)
{
  pageContext.putTransactioinValue("poTrxStep", "1");
}
```

Note: This check is important if the transaction value can change over the course of the transaction and you only want to initialize the value once.

5. Correctly Configure Dialog Pages (Controller Coding Standard C30)

If you navigate to a dialog page, and need to transact data based on the user's response to a question asked in the dialog, configure its buttons to submit its form to the calling page. See Dialog Pages for detailed information about how to do this. OA Framework reenters `processRequest()` in the calling page before proceeding to `processFormData()` and `processFormRequest()`, so make sure that your page anticipates this as described in the Unconditional View Object/Transaction State Initialization section above.

Notes:

- Generally, you don't have to worry about Back button support for the dialog page itself. In the rare case where the dialog page submits the form to itself as opposed to the calling page, you must ensure that the dialog page can be reconstructed correctly as you would for any other page. In other words, any dialog controller code you write in `processRequest()` and `processFormRequest()` must anticipate and handle a potential loss of state.
- Do not use the deprecated methods, `setReleaseAllRootApplicationModules` and

`setReleaseRootApplicationModule` in `OADialogPage`. If these methods are used, the dialog page fails to rebuild its state when the user navigates to an OA Framework page such as Preferences page, and returns to the dialog page using the **Cancel** button. See the Javadoc for the methods for more information.

6. Correctly Communicate State Across Pages (Controller Coding Standards C20, C17)

Before considering which communication method to use in specific situations, understand the options available.

Method	Description
URL Parameter	You can add a parameter and value directly to the URL.
Hidden Field `OAFormValueBean`	Hidden value in the HTML page added to the request when the form is submitted.
Form Parameter `OAFormParameterBean`	Hidden value in the HTML page added to the request when the form is submitted. However, unlike the `OAFormValueBean`, this value is cleared each time the form is submitted (before a new value is set by a `fireAction` or `firePartialAction` event). **Note:** This component's use is reserved exclusively for the declaratively configured `fireAction` and `firePartialAction` events. See the Declarative Form Submit and Dynamic User Interface topics in Chapter 4 for additional information. Never add Form Parameter beans directly to your pages.
`OAPageContext.putParameter()`	Adds a value to a special page cache that persists for a single request. **Note:** If, for example, users navigate to the Personalizations module or Preferences and then return to your page, it will not rebuild correctly if you depend solely on a value set with `putParameter()`. This is because navigation flow entails multiple requests.
Transaction Value `OAPageContext.putTransactionValue()`	Adds a value to the application module transaction that persists as long as the application module is retained.
Application Module Value `OAApplicationModuleImpl.putValue()`	Adds a value to the application module that persists as long as the application module is retained.
View Object Value `OAViewObjectImpl.putValue()`	Adds a value to the view object that persists as long as the application module is retained.
Transient Transaction Value `OAPageContext.putTransactionTransientValue()`	Transaction value that persists within the current application module, in the current JVM.
Application Module Transient Value `OAApplicationModuleImpl.putTransientValue()`	Application module value that persists in the current JVM.
View Object Transient Value `OAViewObjectImpl.putTransientValue()`	View object value that persists in the current JVM.
Session Value `OAPageContext.putSessionValue()`	Adds a value to the servlet session that persists for the duration of the session.

Transient Session Value `OAPageContext.putTransientSessionValue()`	Session value that persists within the current session in the current JVM.

Browser cookies are not included in the list of valid communication techniques. Do not use a browser cookie to store information for your applications. Browsers can reject cookie values and there is a limit in the number of browser cookies a client can accept. Browser cookie usage is reserved for OA Framework internal use only.

Note: It is also not appropriate to automatically force a page refresh to solve Back button issues. If you need to use a page refresh for other reasons and you have additional questions, please contact the OA Framework team.

Usage Recommendations

This section describes the correct way to communicate page-related state in anticipation of browser Back button navigation. These recommendations also enable reliable "Return to" link navigation between modules and ensures that the Personalization and Preferences modules integrate seamlessly with your pages. If, for example, the user selects the **Cancel** button in the Personalizations module, your page should rebuild correctly.

See OA Framework State Management for additional information about the techniques described here.

Intra-Application Navigation

To communicate short, simple state values used to build the target page, use URL parameters.

- If you are communicating state in the context of a form submit, use the `OAPageContext.setForward*()` method "parameters" `HashMap`. If possible, avoid using `OAPageContext.putParameter()`.
- If you are communicating state in the context of a GET request, any bound or static values that you add directly to a component's `Destination URI` become URL parameters. You can also add URL parameters to the request when the user selects a menu item. SeeTabs / Navigation for instructions.

  Note: URLs have a length limit, therefore it is essential that you add only short parameter values using this technique.

Tip: Values set on the URI appear in the browser's address field, and are therefore visible to the user. Users can also see hidden field values if they opt to view the page source. In both of these scenarios, sensitive values must be encrypted as described in Implementing the View: URL Parameters.

To communicate complex or long state values used to build the target page, use any one of the following application module-related caches:

- To cache a value on the transaction, use `OAPageContext.putTransactionTransientValue()`.
- To cache a value on the application module, call `OAApplicationModuleImpl.putTransientValue()`.
- To cache a value on a view object, call `OAViewObjectImpl.putTransientValue()`.

When you use any of these caching strategies, remember that your code must gracefully handle the potential loss of this state. In other words, you must always perform an existence check before trying to use the value, and if the expected value is null or empty, you should either rebuild and re-cache the value, or if that is not possible, show a customized, user-friendly state loss error dialog message.

Exception: If your page is already protected by the UI Transaction Unit coding pattern shown in the Use Case-Specific Coding Standards, then you do not need to perform the existence check for these values.

Additionally, for this caching strategy to work properly, you must comply with the following rules:

- Do not reuse the generic `OAApplicationModule` as a convenience for your read-only pages; always use your own application module class.
- Avoid issuing calls like `OAPageContext.releaseRootApplicationModule()` to release your application module before the page renders; this will cause premature loss of page state. Instead declare your root application module to have a stateless connection by configuring **CONNECTION_AGNOSTIC** or **MANAGE_STATE** retention levels.

Inter-Application Navigation

As in the intra-application page flow case described above, to communicate short, simple state values used to build the target page in cross-product page flows, use URL parameters.

To communicate complex or long state values used to build the target page where the originating and target pages use different application modules, you cannot rely on the intra-application approach described above. Instead, the originating page should call

`OAPageContext.putParameter()` or `OAPageContext.putSessionValue()` / `OAPageContext.putTransientSessionValue()` to pass values to the target module, which then retrieves the values and caches them on its application module. The target page must then:

1. Copy ("transition") the parameter / session values into the root UI application module. In other words, get the parameter / session values and store them on the transaction, application module or view object as described in the intra-application navigation example above.

2. If the incoming values were cached on the session, remove them from the session after copying them.

3. Read the values from the application module cache, always allowing for the possibility that the application module state may be lost.

The following example illustrates a controller `processRequest()` method for the target page with an associated private utility method that "transitions" the passed values to the application module cache. (This assumes that `OAPageContext.putParameter()` was used by the originating page).

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  // Call this before trying to retrieve the value from the transaction.
  transitionPageState(pageContext);

  Object val = pageContext.getTransactionValue("<required page state 1>");

  // Handle the loss of page state values when the application module is
  // released. If one of the required page state values does not exist
  // redirect to a context-specific state loss error page.
  if (val == null)
  {
    // Note that you are strongly recommended to use context-specific message
    // instead of the following generic messages.
    if (pageContext.isBackNavigationFired(true))
    {
      pageContext.redirectToDialogPage(new OADialogPage(NAVIGATION_ERROR));
    }
    else
    {
      pageContext.redirectToDialogPage(new OADialogPage(FAILOVER_STATE_LOSS_ERROR));
    }
  }

  // Otherwise, use the page state and continue processing the request.
  ...

} // end processRequest()


private void transitionPageState(OAPageContext pageContext)
{
  // First inspect one of the required page state values.
  // Use getParameterObject() instead of getParameter() for
  // non-String object types.
  Object val = pageContext.getParameterObject("<required page state 1>");

  if (val != null)
  {
    // Transition the required value.
    pageContext.putTransactionValue("<required page state 1>", (Number)val);

    // If you used session values instead of OAPageContext.putParameter to transport
    // the values, then remove the values from session after each transition.

    // Transition other values.
    val = pageContext.getParameterObject("<required page state 2>");
    pageContext.putTransactionValue("<required page state 2>", (Number)val);

    // For optional values, if the value is not present, explicitly remove values
    // from the transaction.
    val = pageContext.getParameterObject("<optional page state 1>");
```

```
    if (val != null)
    {
      pageContext.putTransactionValue("<optional page state 1>", (Date)val);
    }
    else
    {
      pageContext.removeTransactionValue("<optional page state 1>");
    }
    ...
  }
} // end transitionPageState()
```

Tip: You need at least one required page state value to determine whether transition operations are needed. If there is only one required session value that is shared across UI regions and you decide to code `transitionPageState()` in each region's controller, you can have the topmost region's controller (page layout region's controller) perform the transition of the required session value and immediately call `OAPageContext.putParameter()` to pass down an extra indicator denoting that the required value existed for the current request to the child regions. Child region controllers can check that extra indicator instead.

Hidden Fields

Hidden fields may be used to capture application data that should not render on the page, such as an internal primary key. They may not be used to pass state intended solely for the purpose of building the target page. This is because when your page is accessed using a **Return to** navigation link from another page, such as Personalization page, the former request value that was put into `OAPageContext.putParameter()` to build your page is no longer available in the new request to process the return navigation. As a result, the page state gets lost and cannot be saved again in the hidden field for subsequent form submit requests. Therefore, this technique is deprecated and should be replaced with the techniques described above as appropriate.

7. Conditionally Disallow Form Submit Actions

In certain situations, users should not be able to submit the form on a page after navigating with the browser Back button. See the Use Case-Specific General Standards for a summary of the guidelines followed by implementation details for individual use cases.

Use Case-Specific Standards

In addition to the generic standards above, find your use case from the following list and review the instructions for appropriate Back button behavior and a corresponding implementation.

- Read-Only Page w/ Single Step Create
- Read-Only Page w/ Multistep Create
- Read-Only Page w/ Delete
- Read-Only Page w/ Single Step Update (Iterative Updates Allowed)
- Read-Only Page w/ Single Step Update (Single Update Action Allowed)
- Read-Only Page w/ Multistep Update
- Summary and Transaction Pages in Inter-Application (Cross-Product) Page Flows
- Updateable Summary Page w/ Actions in Table (Duplicate and Remove)
- Partial Page Rendering (PPR) Page Refresh

Testing Back Button Navigation

To fully test your application for Back button compliance:

1. Manually exercise the navigation paths described in the Use Case-Specific Standards listed above. Upon navigating back to the summary page from the transaction page via the browser Back button, also try the following test:

   Intentionally enter invalid form data in the transaction page and perform a form submit action to generate a validation error. Navigate back to the summary page using the browser Back button. The invalid object should not cause any side effect errors when you properly re-enter the transaction page to start a new transaction.

2. From each page, select the global **Home page** link and return to your page using the browser Back button. Press a form submit link (or a button) in your page. This checks the robustness of your page when confronted with a loss of application module and session state. Your page should either rebuild correctly or show a customized, user-friendly state loss error dialog message.

3. From each page, select a **Personalize** link and navigate back to your page with the Personalization page's **Return to..** link. Repeat the same test with the Preferences global link, but return to your page by selecting the Preferences **Cancel** button. These page flows retain your application module state and therefore, your page should rebuild correctly.

Browser Back Button Support Use Cases

Overview

This document describes common application design patterns and associated standard Back button behavior including step-by-step implementation details.

Contents

- General Guidelines
- Read-Only Page w/ Single Step Create
- Read-Only Page w/ Multistep Create
- Read-Only Page w/ Delete
- Read-Only Page w/ Single Step Update (Iterative Updates Allowed)
- Read-Only Page w/ Single Step Update (Single Update Action Allowed)
- Read-Only Page w/ Multistep Update
- Summary and Transaction Pages in Inter-Application (Cross-Product) Page Flows
- Updateable Summary Page w/ Actions in Table (Duplicate and Remove)
- Partial Page Rendering (PPR) Page Refresh

Prerequisite Reading

Review the target goals that we are trying to achieve:

Supporting the Browser Back Button - Target Goals

General Guidelines

In general, your products should behave as follows when the user navigates with the browser Back button. See the specific transaction flows below for implementation and behavior details.

Inquiry Flows

Always continue to work when the browser Back button is clicked.

In the following example, **go Back** is shorthand for *navigate with the browser Back button*.

Example User Action	Result
1. Execute a search, drill down to details for a single record, then **go Back** to the search page. 2. Perform a new search, or navigate to the next set of table records.	Search page is displayed with new search result or the next set of table records as requested.

Navigation within Logical Transaction Flows

A "logical transaction flow" is a task comprised of one or more pages with a definitive end point (commit). Examples include: a single page update, a multistep create, and a delete action (with commit) for one or more selected rows.

The user should be able to navigate between pages in a logical transaction using the browser Back button. All pages should work normally *unless a row has been removed from a rowset in the flow*. For example, if the user removes some objects on the first page of a multistep update, proceeds to the next page and then returns to the first with the browser Back button, subsequent attempts to submit this page will most likely produce a "stale data" error.

In the following examples, **go Back** is shorthand for *navigate with the browser Back button*.

Example User Actions	Result
1. In Step 2 of a 3+ step process, **go Back** to Step 1. 2. Make changes in Step 1 and select Next.	Page 2 displays and the user can continue working.
1. In step 1 of a 3+ step process, the user removes rows from a table and selects Next. 2. Go Back from Step 2 to Step 1. 3. Make additional changes to the data and select Next.	"Stale data" error displays.

Navigation Across Logical Transaction Flows

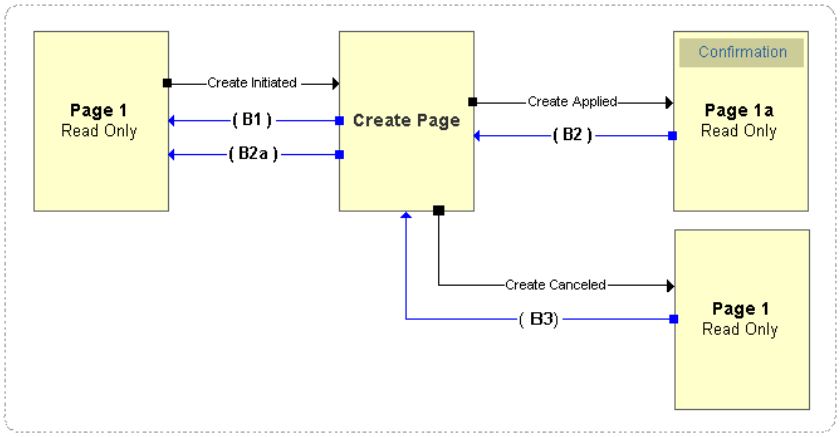
When the user clicks the browser Back button to navigate into a transaction flow that has already been committed, the behavior should be as follows:

- For delete flows, it is never acceptable to resume processing once a row has been deleted. So, if the user navigates back to a page with a rowset including a deleted row and tries to submit the form, an error dialog should display.
- For create flows, it is never acceptable to resume the transaction once committed. So, if the user navigates back to a create page with the browser Back button and tries to submit the form, an error dialog should display.
- For update flows, including "Save for Later" flows, that trigger subsequent processing, such as an approval workflow, or exhibit any other behavior that would prohibit safe reentry; users should not be allowed to resume the transaction once committed. As in the create case, an error dialog will most likely display if the user attempts to submit the form in an update flow page. Otherwise, users should be able to return to an update flow and resume the transaction.
- For any other custom actions, once the user leaves the scope of a logical transaction, you have to assess whether the transaction can be resumed. If not, follow the rules described for the other transaction types above.

Read-Only Page w/ Single Step Create

Tip: This example is fully implemented in the OA Framework ToolBox Tutorial. See the Create lab.

Note: The dotted line in each use case image represents the retained application module boundary.



Scenario Description

Users can search for objects in a summary page (Page 1). This page includes a **Create <*Object*>** button that navigates to the Create Page and creates a new object behind the scenes. When the user selects the **Apply** button on the Create page, their work is saved and Page 1 displays with a Confirmation message (Page 1a). If the user selects the **Cancel** button on the Create page, Page 1 displays with the cached search criteria and results.

Key Assumptions

- All pages share the same root UI application module definition. We also retain the application module while the user works in this module; we don't release it after the transaction is committed. We made this decision because the UI guidelines stipulate that the user's search and sort state in the summary page should be preserved.
- We chose to use a different view object for the Create and Search pages. We did this because -- in real life -- summary view objects are often far smaller than view objects designed to include all the attributes of a typical Oracle E-Business Suite entity object, so we would recommend that you use two different view objects for these cases.
- Per the UI guidelines, we navigate back to the search page to display the Confirmation message where we display the user's cached search results. We do not re-query.

  Note: This means the new row does *not* appear in the result set.
- If the user does any non-standard navigation, or abandons a transaction before returning to the summary page, it's important that the user does not encounter partial (invalid) entity objects in the middle tier cache. Furthermore, once a user creates a new object, the data cannot be updated in the Create page. This action is ambiguous from both a usability and implementation perspective.

Browser Back Button Navigation Events

Navigation Event	Navigation Event Description	Post-Navigation Application Behavior
B1	User abandons create action and navigates back to the summary page (Page 1) using the browser Back button.	No visible impact to the user. They can perform whatever actions they want on this page. Active browser Back button protection is required to ensure subsequent transactions perform correctly. If the user presses the **Create <*Object*>** button again, for example, the user should not encounter partial (invalid) objects in the middle tier cache.
B2	User navigates back to the Create page from the summary w/ confirmation page (Page 1a) using the browser Back button.	If the user tries to make changes here and selects the **Apply** button, an error dialog displays. Active browser Back button protection required to prevent subsequent save actions on the Create page once a transaction is committed.
B2a	Following the B2 navigation, the user presses the browser Back button a second time (without selecting any other buttons on the Create page) to navigate to the originating page (Page 1).	No visible impact to the user. They can perform whatever actions they want on this page.

B3	After selecting the Cancel button in the Create page to abandon the transaction, the user returns to the Create page using the browser Back button and then tries to resume the transaction.	If the user tries to make changes here and selects the **Apply** button, an error dialog displays. Active browser Back button protection required to prevent subsequent save actions on the Create page once a transaction is abandoned and not correctly restarted.

Implementation Instructions

Note: This assumes that you know how to implement a Create; it focuses only on the work that you need to do for the browser Back button. For the complete example, see the Create lab in the OA Framework Toolbox Tutorial.

In this flow, the `TransactionUnitHelper` is the primary tool for ensuring that the user doesn't update newly created records, inadvertently create the same record multiple times, or try to commit with abandoned objects in the middle tier cache.

Step 1: Follow all the general coding standards as described in Supporting the Browser Back Button.

Step 2: Define your Create <*Object*> page. Make sure the **Cancel** and **Apply** buttons are both submit buttons. You should also disable client-side validation and server-side validation for the **Cancel** button.

Step 3: Associate a controller with the Create <*Object*> page and add the following `processRequest()` logic. This detects valid navigation using the **Create <*Object*>** button and starts the "*obj*CreateTxn" user interface transaction unit. If navigation to the page using the browser Back button is detected, an error dialog is displayed.

The following code samples assume that we're creating an employee:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  // If isBackNavigationFired = false, we're here after a valid navigation
  // (the user selected the Create button) and we should proceed
  // normally and initialize a new employee.
  if (!pageContext.isBackNavigationFired(false))
  {
    // We indicate that we are starting the create transaction (this
    // is used to ensure correct Back button behavior). Note that you
    // can assign whatever name you want to your transaction unit.
    TransactionUnitHelper.startTransactionUnit(pageContext, "empCreateTxn");


    if (!pageContext.isFormSubmission())
    {
      OAApplicationModule am = pageContext.getApplicationModule(webBean);

      // Call your method to handle creating the new row.
      am.invokeMethod("createEmployee", null);
    }
  }
  else
  {
    if (!TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "empCreateTxn", true))
    {
      // We got here through some use of the browser "Back" button, so we
      // want to display a state loss error and disallow access to the page.

      // If this were a real application, we would probably display a more
      // context-specific message telling the user they can't use the browser
      // "Back" button on the "Create" page. Instead, we wanted to illustrate
      // how to display the Applications standard NAVIGATION_ERROR message.
      OADialogPage dialogPage = new OADialogPage(NAVIGATION_ERROR);
      pageContext.redirectToDialogPage(dialogPage);
    }
  }
} // end processRequest()
```

Known Limitation: There is a known technical limitation with the code presented above for the Create page. If the user navigates from the Create page to the Personalization or Preferences page then returns to the Create page using the **Return to ...** link or **Cancel** button, a duplicate record gets created because this navigation is not a browser Back button navigation. There may be no error when the Create page is re-rendered, but the duplicate records could later cause validation errors if the user performs a commit action. In addition, if the user actually filled the form data and encountered a validation error in the Create page before navigating to the Personalization page, then upon returning to the Create page, the user sees validation errors that make no sense. It is difficult to trap all possible Return navigation links, so to manage these situations, adopt either or both of the following strategies:

- **Tip 1:** Initialize your new row state with `ViewRowImpl.setNewRowState(Row.STATUS_INITIALIZED)` call per Model Coding Standard M69 for a view object based on entity objects. In addition, centralize your validation logic in `EntityImpl` subclass `validateEntity()` method instead of placing your validation logic in the `ViewRowImpl` subclass `validate()` method. The entities in `STATUS_INITIALIZED` state are considered temporary and are ignored upon validation or commit. However, this strategy does not solve the second problem where the Create page has validation errors; the second scenario is a corner case though.
- **Tip 2:** Make your model's Create logic (for instance, `createEmployee` code in the above example) re-entrant. To be more specific, to cope with both problems mentioned above, the Create logic can skip creating a new row if a new row has already been created. For example:

```
ViewRowImpl currentRow = (ViewRowImpl)vo.getCurrentRow();
if (currentRow != null)
{
  byte entityState = currentRow.getEntity(0).getPostState(); // assuming your first EO is updateable
  if ((entityState == Entity.STATUS_INITIALIZED) || (entityState == Entity.STATUS_NEW))
  // For the deprecated OAPlsqlViewRowImpl case, OAPlsqlViewRowImpl.getPlsqlState()
  // can be inspected.
  {
    // The current row exists, and it is a new row.
    // Skip creating a new row and return.
```

```
        return;
      }
    }

    // Perform normal processing -- create a new row.
    ...
```

Step 4: Add `processFormRequest()` logic to your Create <*Object*> controller to handle the **Cancel** and **Apply** button selection. In both cases, we mark the "*obj*CreateTxn" as being finished and navigate back to the read-only summary page. In the Cancel case, we also explicitly rollback the transaction to ensure that objects with pending changes are cleared from the middle tier cache.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processFormRequest(pageContext, webBean);

  OAApplicationModule am = pageContext.getApplicationModule(webBean);

  // Pressing the "Apply" button means the transaction should be validated
  // and committed.
  if (pageContext.getParameter("Apply") != null)
  {
    Number employeeNumber = ...
    String employeeNum = ...

    // Simply telling the transaction to commit will cause all the Entity Object
    // validation to fire.

    // Note: there's no reason for a developer to perform a rollback. This is handled by
    // the framework if errors are encountered.  The apply() method ultimately issues
    // a commit.
    am.invokeMethod("apply");

    // Indicate that the Create transaction is complete.
    TransactionUnitHelper.endTransactionUnit(pageContext, "empCreateTxn");

    // Assuming the "commit" succeeds, navigate back to the "Search" page with
    // the user's search criteria intact and display a "Confirmation" message
    // at the top of the page.
    MessageToken[] tokens = { new MessageToken("EMP_NAME", employeeName),
    new MessageToken("EMP_NUMBER", employeeNum) };

    OAException confirmMessage = new OAException("AK", "FWK_TBX_T_EMP_CREATE_CONFIRM", tokens,
                                 OAException.CONFIRMATION, null);

    // Per the UI guidelines, we want to add the confirmation message at the
    // top of the search/results page and we want the old search criteria and
    // results to display.

    pageContext.putDialogMessage(confirmMessage);

    pageContext.forwardImmediately("OA.jsp?page=/oracle/apps/fnd/framework/labs/toolbox/labsolutions/webui/EmpSearchPG",
                null,
                OAWebBeanConstants.KEEP_MENU_CONTEXT,
                null,
                null,
                true, // retain AM
                OAWebBeanConstants.ADD_BREAD_CRUMB_NO);
  }
  else if (pageContext.getParameter("Cancel") != null)
  {
    // Issue a rollback to ensure that the middle tier cache is cleared of pending
    // changes.
    //
    // Note that the rollbackEmployee() method in the root UI application module
    // has an isDirty() check optimization so the rollback can be avoided
    // if it isn't required (this can be used for EO-based view objects only).
    // See oracle.apps.fnd.framework.toolbox.labsolutions.server.EmployeeAMImpl.java
    // in the ToolBox Tutorial for an example.
    am.invokeMethod("rollbackEmployee");

    // Indicate that the Create transaction is complete.
    TransactionUnitHelper.endTransactionUnit(pageContext, "empCreateTxn");

    pageContext.forwardImmediately("OA.jsp?page=/oracle/apps/fnd/framework/labs/toolbox/labsolutions/webui/EmpSearchPG",
                null,
                OAWebBeanConstants.KEEP_MENU_CONTEXT,
                null,
                null,
                true, // retain AM
                OAWebBeanConstants.ADD_BREAD_CRUMB_NO);
  }
} // end processFormRequest()
```

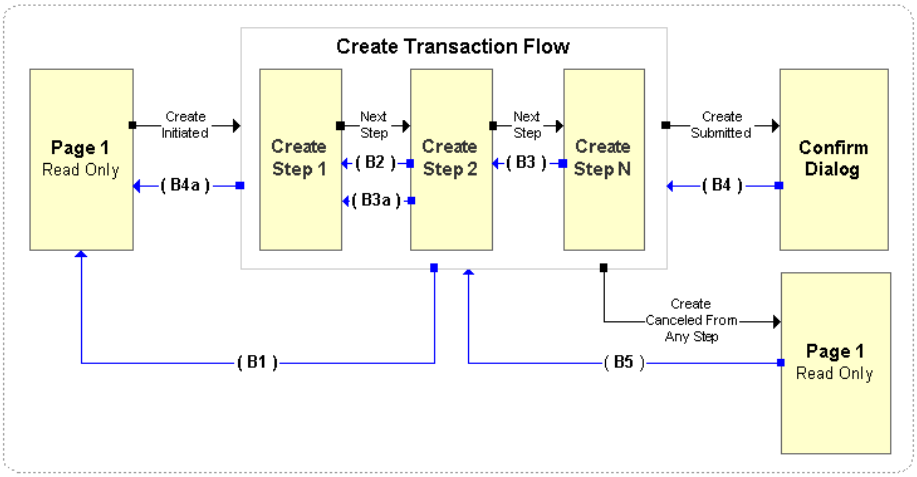
Step 5: Add logic to the summary page's `processRequest()` to guard against Back button navigation by ensuring that the middle tier cache is fully cleared before the user tries to start a new Create <*Object*> transaction.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  OAApplicationModule am = pageContext.getApplicationModule(webBean);
```

```
  // The following checks to see if the user navigated back to this page
  // without taking an action that cleared an "in transaction" indicator.
  // If so, we want to rollback any changes that she abandoned to ensure they
  // aren't left lingering in the BC4J cache to cause problems with
  // subsequent transactions. For example, if the user navigates to the
  // Create Employee page where you start a "create" transaction unit,
  // then navigates back to this page using the browser Back button and
  // selects the Create Employee button again, the OA Framework detects this
  // Back button navigation and steps through processRequest() so this code
  // is executed before you try to create another new employee.

  // Note that we pass "false" as the final parameter to the isTransactionUnitInProgress()
  // method because the current page is not part of the transaction flow.

  if (TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "empCreateTxn", false))
  {
    am.invokeMethod("rollbackEmployee");
    TransactionUnitHelper.endTransactionUnit(pageContext, "empCreateTxn");
  }

  ...
}
```

Read-Only Page w/ Multistep Create

Note: The dotted line represents the retained application module boundary. The solid box around the "Create Transaction Flow" simply represents a grouping for the transaction pages to simplify the description of navigation events.



Scenario Description

In this case, the user steps through several pages to complete the Create transaction launched with the selection of the **Create <*Object*>** button on a read-only Summary page. When the **Submit** or **Finish** button is selected in the final page to commit the transaction, a Confirmation dialog is displayed. When the user selects the **OK** button in the dialog, the originating Summary page displays.

Key Assumptions

- All pages share the same root UI application module definition. We also retain the application module while the user works in this module; we don't release it after the transaction is committed. We made this decision because the UI guidelines stipulate that the user's search and sort state in the summary page should be preserved.
- We chose to use a different view object for the Create and Search pages. We did this because "in real life" summary view objects are often far smaller than view objects designed to include all the attributes of a typical Oracle E-Business Suite entity object. We recommend that you use two different view objects for these cases.
- Per the UI guidelines, we display a Confirmation dialog at the end of the multistep transaction and then display the read-only summary page with the user's cached search state. We do not re-query.

  Note: The new row does *not* appear in the result set.
- If the user does any non-standard navigation, or abandons a transaction before returning to the summary page, it's important that the user does not encounter partial (invalid) entity objects in the middle tier cache. Furthermore, once a user commits a new object, the data cannot be updated in the Create flow. This action is ambiguous from both a usability and implementation perspective.

Browser Back Button Navigation Events

Navigation Event	Navigation Event Description	Post-Navigation Application Behavior

B1	User abandons the create flow and navigates back to the summary page (Page 1) using the browser Back button.	No visible impact to the user. They can perform whatever actions they want on this page. Active browser Back button is protection required to ensure subsequent transactions perform correctly . If the user presses the **Create <*Object*>** button again, for example, the user should not encounter partial (invalid) objects in the middle tier cache.
B2, B3, B3a	User moves back to previous Create steps within the flow using the browser Back button.	No visible impact to the user. They can perform whatever actions they want on this page. **Note:** You must still guard against the possibility of inadvertently creating duplicate rows.
B4	User navigates back to the Create flow from Confirmation dialog using the browser Back button.	If the user tries to make changes in any of the Create flow pages and submits the form, an error dialog displays. Active browser Back button protection required to prevent subsequent save actions on the Create page once a transaction is committed.
B4a	Following the B4 navigation, the user presses the browser Back button multiple times (without selecting any other buttons on the Create pages) to navigate to the originating page (Page 1).	No visible impact to the user. They can perform whatever actions they want on this page.
B5	After selecting the Cancel button from within the Create flow to abandon the transaction, the user returns to the Create flow using the browser Back button and tries to resume the transaction.	If the user tries to make changes here and selects the **Submit** or **Finish** button, an error dialog displays. Active browser Back button protection required to prevent subsequent save actions on the final Create page once a transaction is abandoned and not correctly restarted.

Implementation Instructions

Tip: This example assumes that you know how to implement a multistep Create flow; it focuses exclusively on Back button handling. For a complete multistep Create example, see the OA Framework ToolBox Sample Library's Create Purchase Order flow.

From a Back button handling perspective, the multistep create is almost identical to the single step flow described above.

Step 1: Follow all the general coding standards as described in Supporting the Browser Back Button.

Step 2: Define your Create <*Object*> page flow. Make sure the navigation bar **Cancel** and **Submit/Finish** buttons are submit buttons. You should also disable client-side validation and server-side validation for the **Cancel** button.

Step 3: Associate a controller with the first page in your Create flow and add `processRequest()` logic as shown in the following create purchase order example. This detects valid navigation using the **Create <*Object*>** button and starts the *"obj*CreateTxn" user interface transaction unit just as you would in the single-step case with one modification: you must ensure that you do not repeat the initialization logic if the user simply navigates back to this first page from subsequent pages in the flow using the page flow navigation buttons.

Assuming your multipage flow uses an `OANavigationBarBean` for Next/Back navigation, you can add the following test to your `isBackNavigationFired()` check. See Locator Element: Page Navigation for additional information about the `OANavigationBarBean`.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  // If isBackNavigationFired = false, we're here after a valid navigation
  // and we should proceed normally.

  // If the user navigates back to this flow by selecting the Back
  // button in the next page in the multipage flow, we don't want to do
  // anything on this page (we don't want to recreate the PO, and we don't
  // want to display a state loss error).

  if (!pageContext.isBackNavigationFired(false) &&
       (!"goto".equals(pageContext.getParameter(EVENT_PARAM))))
  {
    OAApplicationModule am = pageContext.getApplicationModule(webBean);

    TransactionUnitHelper.startTransactionUnit(pageContext, "poCreateTxn");


    if (!pageContext.isFormSubmission())
    {
      // Ask the application module to initialize a purchase order so the
      // user can complete it.
      am.invokeMethod("createPurchaseOrder");
```

```
    }
  }
  else
  {
    if (!TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "poCreateTxn", true))
    {
      // Please use a custom message for the dialog page!
      OADialogPage dialogPage = new OADialogPage(STATE_LOSS_ERROR);
      pageContext.redirectToDialogPage(dialogPage);
    }
  }
} // end processRequest()
```

Note: Also check the Known Limitation in the single step Create case.

Step 4: To guard against invalid actions on pages within the flow such as, if the user tries to navigate back to the Create flow to make changes after committing the new row, you must associate a controller with each of the subsequent pages in your flow and add the following `processRequest()` logic:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  if (!TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "poCreateTxn",true))
  {
    // Please use a custom message for the dialog page!
    OADialogPage dialogPage = new OADialogPage(NAVIGATION_ERROR);
    pageContext.redirectToDialogPage(dialogPage);
  }

  ...
} // end processRequest()
```

Step 5 (Conditionally Required): If your page flow includes the process of creating additional new rows at any point in the a `processRequest()` method (for example, you create an employee address row in the middle of the flow that started with the creation of the employee), add protection code as described above in Step 3 -- except skip the call to `TransactionUnitHelper.startTransactionUnit()`.

Step 6: Add `processFormRequest()` logic to whatever controller is handling the multipage flow **Submit/Finish** and **Cancel** buttons.

For the **Submit/Finish**, this marks the "*obj*CreateTxn" as being finished and navigates to a Confirmation dialog. For the **Cancel**, this marks the "*obj*CreateTxn" as being finished, rolls back the transaction to remove objects with pending changes from the middle tier cache, and navigates back to the read-only summary page.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processFormRequest(pageContext, webBean);

  OAApplicationModule am = pageContext.getApplicationModule(webBean);

  if (pageContext.getParameter("Submit") != null)
  {
    am.invokeMethod("savePurchaseOrder");

    TransactionUnitHelper.endTransactionUnit(pageContext, "poCreateTxn");

    // redirect to Confirmation dialog
    OADialogPage dialogPage = new OADialogPage(OAException.CONFIRMATION, ...);
    pageContext.redirectToDialogPage(dialogPage);
  }
  else if (pageContext.getParameter("Cancel") != null)
  {
    // Clear the middle tier cache.
    // Note that the rollbackPurchaseOrder() method in the root UI application module
    // has an isDirty() check optimization so the rollback can be avoided
    // if it isn't required (this can be used for EO-based view objects only).
    // See oracle.apps.fnd.framework.toolbox.tutorial.server.MultistepCreateAMImpl.java
    // in the ToolBox Sample Library for an example.
        am.invokeMethod("rollbackPurchaseOrder");

    // Remove the "in transaction" indicator.
        TransactionUnitHelper.endTransactionUnit(pageContext, "poCreateTxn");

    pageContext.forwardImmediately("OA.jsp?page=/oracle/apps/fnd/framework/labs/toolbox/tutorial/webui/PoSummaryCreatePG",
                null,
                OAWebBeanConstants.KEEP_MENU_CONTEXT,
                null,
                null,
                true, // retain AM
                OAWebBeanConstants.ADD_BREAD_CRUMB_NO);
  }
} // end processFormRequest()
```

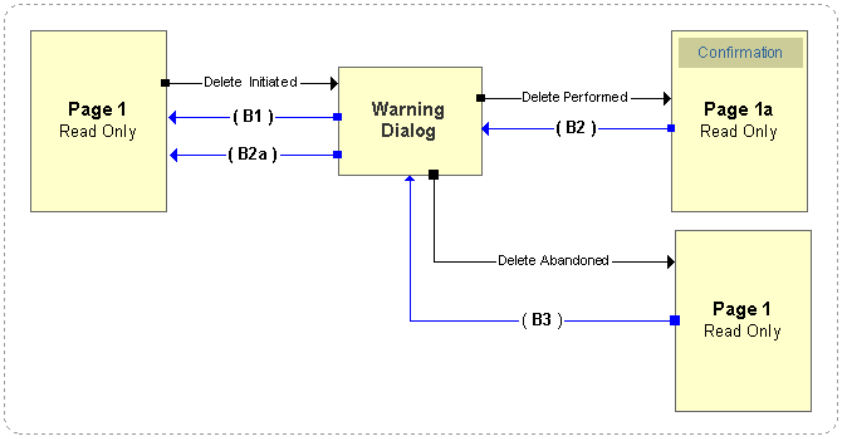
Step 7: Add logic to the summary page's `processRequest()` to guard against Back button navigation by ensuring that the middle tier cache is fully cleared before the user tries to start a new Create <*Object*> transaction.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  OAApplicationModule am = pageContext.getApplicationModule(webBean);

  // The following checks to see if the user navigated back to this page
  // without taking an action that cleared an "in transaction" indicator.
```

```
  // If so, we want to rollback any changes that she abandoned to ensure they
  // aren't left lingering in the BC4J cache to cause problems with
  // subsequent transactions.
  //
  // Note that we pass "false" as the final parameter to the isTransactionUnitInProgress()
  // method because the current page is not part of the transaction flow.
   if (TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "poCreateTxn", false))
  {
    am.invokeMethod("rollbackPurchaseOrder");
    TransactionUnitHelper.endTransactionUnit(pageContext, "poCreateTxn");
  }

  ...
}
```

Read-Only Page w/ Delete

Note: The dotted line in each use case image represents the retained application module boundary.



Scenario Description

A read-only summary page table includes a **Delete** icon for each row in the table. When the user selects the Delete form submit component, a Warning dialog displays asking if they really want to delete the selected object.

When the user selects the **Yes** or **No** button, the dialog page performs a form submit to the Summary page. If the user selects the **Yes** button, the Summary page deletes the record and commits the change.

Key Assumptions

- All pages share the same root UI application module, which is retained for all standard application navigation events (even after the commit) to preserve the search and sort state of the summary page .
- All the buttons and icons used for the standard application navigation perform a form submit. The physical navigation is performed with a JSP forward.
- Once the user deletes a row, they cannot perform any subsequent transactions on it.

Browser Back Button Navigation Events

Navigation Event	Navigation Event Description	Post-Navigation Application Behavior
B1	User returns from the Warning dialog to the originating page using the browser Back button.	No visible impact to the user. They can perform whatever actions they want on this page.
B2	User navigates from the originating page (Page 1) with the transaction Confirmation message to the Warning dialog using the browser Back button after the transaction completion.	If the user presses the **Yes** button (in effect asking to delete the row a second time), navigate to Page 1 and display an error message indicating that the row has already been deleted. Active Back button protection is required to detect and handle this event.
B2a	Following the B2 navigation, the user presses the browser Back button a second time to navigate from the Warning dialog to the originating Page 1. **Note:** They do not press any buttons in the Warning dialog before doing this.	Since the page will display in its original state (before the row was deleted), the user might try to select the same row for deletion a second time. In this case, the underlying frameworks automatically detect this condition and display an error dialog. Automated browser Back button protection is provided, assuming you follow the general coding standards.

B3	After selecting the **No** button in the Warning dialog to cancel the Delete action, the user selects the browser Back button to return to the Warning dialog.	No visible impact to the user. They can perform whatever actions they want on this page. For example, if the user selects the **Yes** button, the row is deleted.

Implementation Instructions

Note: This assumes that you know how to implement a typical Delete; it focuses only on the work that you need to do for the browser Back button. For the complete example, see the OA Framework ToolBox Sample Library's Delete Purchase Order flow.

Step 1: Follow all the general coding standards as described in Supporting the Browser Back Button.

Step 2: Define your Delete page flow. Make sure the **Delete** icon is configured to submit the form, and make sure the **Yes/No** buttons in the Warning dialog submit the form to your calling page.

Step 3: Define a `delete()` method in your root application module that returns a Boolean "row found" indicator as shown in the following example:

```
/*
 * Deletes a purchase order from the PoSimpleSummaryVO using the
 * poHeaderId parameter.  Returns true if the object was found
 * and deleted.
 */
public Boolean delete(String poHeaderId)
{
  // First, we need to find the selected purchase order in our VO.
  // When we find it, we call remove( ) on the row which in turn
  // calls remove on the associated PurchaseOrderHeaderEOImpl object.

  int poToDelete = Integer.parseInt(poHeaderId);

  boolean rowFound = false;

  OAViewObject vo = getPoSimpleSummaryVO();
  PoSimpleSummaryVORowImpl row = null;

  // This tells us the number of rows that have been fetched in the
  // row set, and will not pull additional rows in like some of the
  // other "get count" methods.

  // Note: instead of using a manual iterator, you could use the
  // convenience OAViewObjectImpl.getFilteredRows() methods. They
  // support both multiple row and single row matches. Since the
  // purchase order header value that we're using is the primary key,
  // you could also use OAViewObjectImpl.findByKey(). See the Javadoc
  // for these other approaches.

  int fetchedRowCount = vo.getFetchedRowCount();

  // We use a separate iterator -- even though we could step through the
  // rows without it -- because we don't want to affect row currency.

  RowSetIterator deleteIter = vo.createRowSetIterator("deleteIter");

  if (fetchedRowCount > 0)
  {
    deleteIter.setRangeStart(0);
    deleteIter.setRangeSize(fetchedRowCount);

    for (int i = 0; i < fetchedRowCount; i++)
    {
      row = (PoSimpleSummaryVORowImpl)deleteIter.getRowAtRangeIndex(i);

      Number primaryKey = row.getHeaderId();
      if (primaryKey.compareTo(poToDelete) == 0)
      {
        row.remove();
        getTransaction().commit();
        rowFound = true;
        break; // only one possible selected row in this case
      }
    }
  }

  deleteIter.closeRowSetIterator();
  return new Boolean(rowFound);
} // end delete()
```

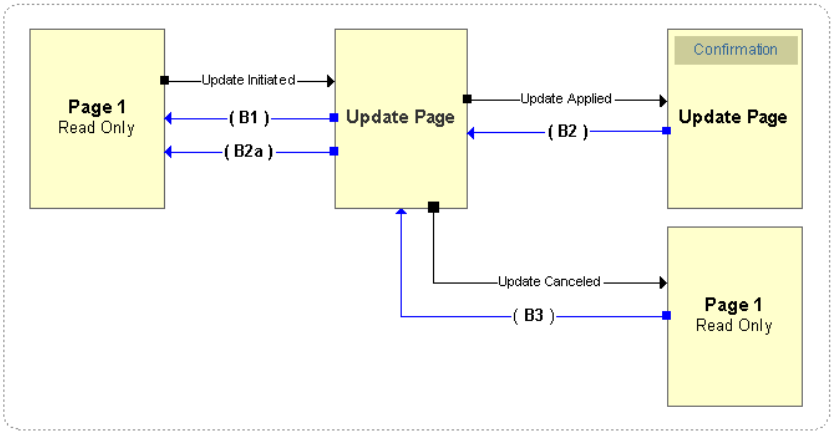
Step 4: In your controller code `processFormRequest()` handler for the **Yes** button in the Warning dialog, check to see if the row was found and deleted. If not, display the error message at the top of the summary page.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  if (pageContext.getParameter("<DeleteYesButton>") != null)
  {
    OAApplicationModule am = pageContext.getApplicationModule(webBean);
    String poHeaderId = pageContext.getParameter("poHeaderId");
    Serializable[] parameters = { poHeaderId };

    Boolean rowRemoved = ((Boolean)am.invokeMethod("delete", parameters)).booleanValue();

    // Remember to use Message Dictionary for message text; never hard-code the value
    if (rowRemoved)
    {
      pageContext.putDialogMessage(new OAException("<confirmation message>",
```

```
                                    OAException.CONFIRMATION);
    }
    else
    {
      // See the "Error" message in the ToolBox Tutorial Application "Delete" page
      // for example content.
      pageContext.putDialogMessage(new OAException("<error message>", OAException.ERROR);
    }
  }
}
```

Read-Only Page w/ Single Step Update (Iterative Updates Allowed)

Note: The dotted line represents the retained application module boundary.



Scenario Description

A read-only summary page table includes an **Update** icon for each row. When the user selects the **Update** icon, they navigate to an update page that is queried using the primary key value passed on with the request. When the user selects the **Apply** button, changes are committed and a Confirmation message is displayed at the top of the Update page.

Note: For this use case, note that the Apply action stays in the same Update page to make it convenient for users to perform multiple updates before returning to the summary page using the **Return to ...** link or **Cancel** button. This also implies that the user can repeat the Update action after the transaction is completed when the Update page is accessed using the browser Back button.

Key Assumptions

- All pages share the same root UI application module, which is retained for all standard application navigation events (even after the commit) to preserve the search and sort state of the summary page.
- We chose to use a different view object for the Update and Search pages. We did this because "in real life" summary view objects are often far smaller than view objects designed to include all the attributes of a typical Oracle E-Business Suite entity object. We recommend that you use two different view objects for these cases.
- All the buttons, links and icons used for standard application navigation perform a form submit. The physical navigation is performed with a JSP forward.
- Since the Confirmation message is displayed at the top of the Update page, we assume the user can make as many iterative changes as they want before navigating off the page.

Browser Back Button Navigation Events

Navigation Event	Navigation Event Description	Post-Navigation Application Behavior
B1	User abandons "update" action and navigates back to the summary page (Page 1) using the browser Back button.	No visible impact to the user. They can perform whatever actions they want on this page. Active browser Back button is protection required to ensure subsequent transactions perform correctly. If, for example, the user presses the **Update<*Object*>** button again, the user should not encounter partial (invalid) objects in the middle tier cache.
B2	User navigates back to the previous state of the Update page (before the Confirmation message displays) using the browser Back button.	Update transaction can be resumed. No visible impact to the user. They can perform whatever actions they want on this page.
B2a	Following the B2 navigation, the user presses the browser Back button a second time (without selecting any other buttons on the Update page) to navigate to the originating page (Page 1).	No visible impact to the user. They can perform whatever actions they want on this page.

B3	After selecting the **Cancel** button in the Update page to abandon the transaction, the user returns to the Update page using the browser Back button and then tries to resume the transaction.	Behavior should be as described in B2.

Implementation Instructions

Tip: This example assumes that you know how to implement a single step Update; it focuses exclusively on Back button handling. For a complete single step Update example with iterative updates allowed, see the Update Purchase Order in the ToolBox Sample Library.

Updating the record multiple times is considered a "benign" action, so you generally shouldn't prevent Back button access to this kind of a page. You do, however, want to ensure that the user isn't surprised by abandoned objects in the middle tier cache while working on a subsequent transaction. To do this: follow all the instructions for the Create case above with the following differences:

- You must pass the primary key for the object to update as a URL request parameter.
- There is no need to perform the `isBackNavigationFired()` check in the Update page's `processRequest()` logic. You will simply start the Update transaction unit.
- You don't need to worry about the `isFormSubmission()` check before finding your row for update purposes as this needed only when creating new rows. However, do pay close attention to the View Object Initialization Guidelines to ensure that you don't inadvertently lose user data.
- Unlike the Create case described above, you will not end the transaction unit when the user selects the **Apply** button. Instead, the transaction unit remains "alive" until the user explicitly indicates that they are finished with their edits and "leaves" ("exits") the Update page by selecting the **Cancel** button. When the **Return to ...** link is used to exit the Update page, the summary page ends the transaction unit instead, and therefore, it is desirable for the summary page to use the optimization with `Transaction.isDirty()` check to avoid a redundant rollback operation.

Step 1: Follow all the general coding standards as described in Supporting the Browser Back Button.

Step 2: Define your Update pages. Make sure both your **Apply** and **Cancel** buttons are submit buttons. You should also disable client-side validation and server-side validation for the **Cancel** button.

Step 3: Add a controller to your Update page with the following `processRequest()` logic:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  // Put a transaction value indicating that the update transaction
  // is now in progress (transaction-in-progress indicator).
  TransactionUnitHelper.startTransactionUnit(pageContext, "poUpdateTxn");

  ...

  // Now we want to initialize the query for our single purchase order with
  // all of its details.
  OAApplicationModule am = pageContext.getApplicationModule(webBean);
  Serializable[] parameters = { orderNumber };
  boolean rowFound =
      ((Boolean)am.invokeMethod("init", parameters)).booleanValue();

  // If a matching row is not found, it could have been deleted... Need to display
  // an error message to the user.
  if (!rowFound)
  {
    // Add logic to display a dialog page with a custom error message instead
    // of the standard state loss error shown here -- OA Framework Release 12
    // provides automatic protection for this case by showing stale data error.
    OADialogPage dialogPage = new OADialogPage(STATE_LOSS_ERROR);
    pageContext.redirectToDialogPage(dialogPage);
  }
  else
  {
    // Proceed normally with any other page processing.
    ...
  }
} // end processRequest()
```

Step 4: Add the following `processFormRequest()` logic to the Update page controller:

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processFormRequest(pageContext, webBean);

  OAApplicationModule am = pageContext.getApplicationModule(webBean);

  if (pageContext.getParameter("Apply") != null)
  {
    // If the user presses the "Apply" button, we want to save their changes
    // and display a confirmation message at the top of the page. The users
    // can make iterative edits. When they're ready to leave the page, they
    // can select "Cancel" to abandon any unsaved changes, the "Return to
    // Purchase Orders" link, or they can can simply navigate out of the page
    // by selecting a menu item.
    am.invokeMethod("apply");

    // Now, redisplay the page with a confirmation message at the top.

    // Get the purchase order number from the request.

    String orderNumber = pageContext.getParameter("headerId");
```

```
    MessageToken[] tokens = { new MessageToken("PO_NUMBER", orderNumber) };
    OAException message = new OAException("AK",
                                          "FWK_TBX_T_PO_UPDATE_CONFIRM",
                                          tokens,
                                          OAException.CONFIRMATION, null);

    pageContext.putDialogMessage(message);
  }
  else if (pageContext.getParameter("Cancel") != null)
  {
    am.invokeMethod("rollbackPurchaseOrder");

    // Remove the "in transaction" indicator.
    TransactionUnitHelper.endTransactionUnit(pageContext, "poUpdateTxn");

    pageContext.forwardImmediately("OA.jsp?page=/oracle/apps/fnd/framework/labs/toolbox/tutorial/webui/PoSummaryUpdatePG",
                null,
                OAWebBeanConstants.KEEP_MENU_CONTEXT,
                null,
                null,
                true, // retain AM
                OAWebBeanConstants.ADD_BREAD_CRUMB_NO);
  }
} // end processFormRequest()
```

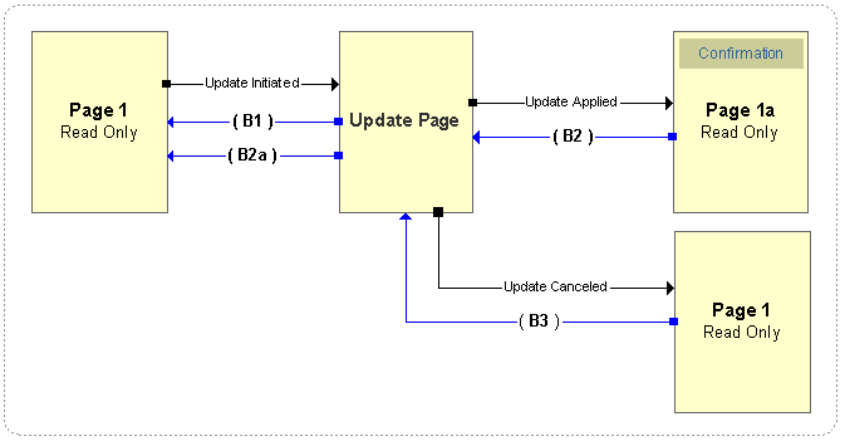
Step 5: Add a controller to your Summary page with the following `processRequest()` logic.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  super.processRequest(pageContext, webBean);

  OAApplicationModule rootAM = pageContext.getRootApplicationModule();

  // Note that we pass "false" as the final parameter to the isTransactionUnitInProgress()
  // method because the current page is not part of the transaction flow.
  if (TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "poUpdateTxn", false))
  {
    // Note that the rollbackPurchaseOrder() method in the root UI application module
    // has an isDirty() check optimization so the rollback can be avoided
    // if it isn't required (this can be used for EO-based view objects only.
    // See oracle.apps.fnd.framework.toolbox.tutorial.server.UpdateAMImpl.java
    // in the ToolBox Sample Library for an example.
    rootAM.invokeMethod("rollbackPurchaseOrder");
    TransactionUnitHelper.endTransactionUnit(pageContext, "poUpdateTxn");
  }

  ...
} // end processRequest()
```

With these changes, the user can revisit this page using the browser Back button and make changes after a commit. However, if the user starts the Update transaction, navigates to the Summary page using the browser Back button, and then tries to update a different record or select a **Create** button, the Summary page's `processRequest()` logic will be re-executed and the middle tier cache will be cleared of abandoned objects.

Read-Only Page w/ Single Step Update (Single Update Action Allowed)

Note: The dotted line represents the retained application module boundary.



From a navigation and key assumptions perspective, this flow is exactly the same as the single step Create flow described above: the user selects an **Update** icon or button in the read-only summary page and an Update page displays. The user makes changes, and presses the **Apply** button to save their changes. The summary page displays with a confirmation message at the top of the page. The **Cancel** button behaves the same way in both scenarios.

The key behavioral difference between the Single-step update (single update action allowed) case and the Single-step update (Iterative Updates Allowed) case is that the Apply action in the Single-step update (single update action allowed) case leaves (exits) the transaction page after the Apply action. Although Single-step update (Iterative Updates Allowed) case provides a *convenience* mechanism allowing users to make multiple updates before exiting the Update page, it does *not* mean that the the Single-step update (single update action allowed) case should have a restrictive browser Back button handling behavior. In other words, the the Single-step update (single update action allowed) case should not always forbid the Update action after the transaction is completed when the Update page is accessed using the browser Back button.

From an implementation perspective, the key difference between the Single-step update (single update action allowed) case and the Single-step update (Iterative Updates Allowed) case is that the Apply action in the Single-step update (single update action allowed) case also has to end the transaction unit because the user leaves (exits) the

transaction page after the Apply action. In other words, the user is finished with their edits.

We also need to manage the following two cases.

Updates Allowed After Back Button Navigation

Updating a record multiple times is considered a "benign" action, so you generally shouldn't prevent Back button access to this kind of flow. You do, however, want to ensure that the user isn't surprised by abandoned objects in the middle tier cache while working on a subsequent transaction. To do this: follow all the instructions for the Create case above with the following differences:

- You must pass the primary key for the object to update as a URL request parameter.
- There is no need to perform the `isBackNavigationFired()` check in the Update page's `processRequest()` logic. Simply start the update transaction unit.
- You don't need to worry about the `isFormSubmission()` check before finding your row for update purposes as this is needed only when creating new rows. However, do pay close attention to the View Object Initialization Guidelines to ensure that you don't inadvertently lose user data.

For a code example, see the Update lab in the ToolBox Tutorial.

Updates Not Allowed After Back Button Navigation

To disallow updates if the page is accessed in an "abnormal" page flow follow all the instructions for the multistep Create case above with the following differences:

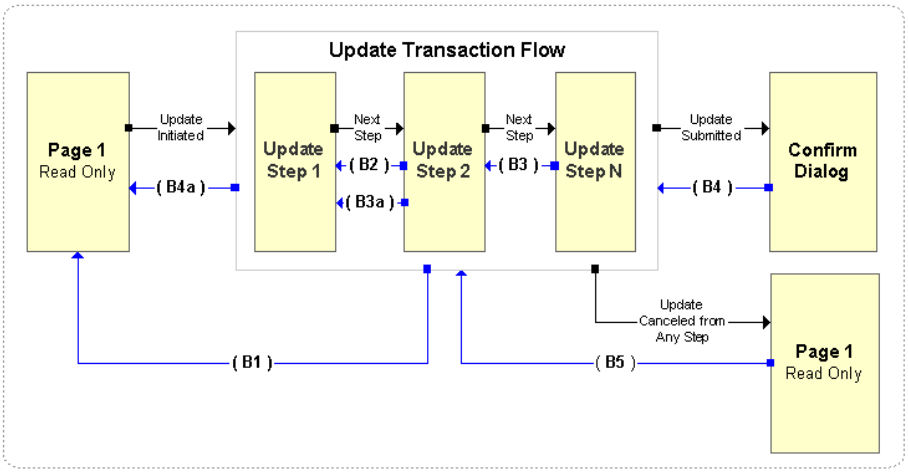
Example: Once a purchase order is updated it is submitted to an Approval workflow for processing so that subsequent changes cannot be made after the user commits.

- As mentioned above, you *must* pass the primary key for the object you want to update as a URL request parameter.
- You don't need to worry about the `isFormSubmission()` check before finding your row for update purposes as this is needed only when creating new rows. However, do pay close attention to the View Object Initialization Guidelines to ensure that you don't inadvertently lose user data.

In this case, you must still perform the `isBackNavigationFired()` check as in the Create case to disallow further updates once a change is committed.

Read-Only Page w/ Multistep Update

Note: The dotted line represents the retained application module boundary. The solid box around the "Update Transaction Flow" simply represents a grouping for the transaction pages to simplify the description of navigation events.



Scenario Description

In this case, the user steps through several pages to complete the Update transaction launched with selection of the Update form submit icon on a read-only Summary page. When the **Submit** or **Finish** button is selected in the final page to commit the transaction, a Confirmation dialog is displayed. When the user selects the **OK** button in the dialog, the originating Summary displays.

Key Assumptions

- All pages share the same root UI application module definition. We also retain the application module while the user works in this module; we don't release it after the transaction is committed. We made this decision because the UI guidelines stipulate that the user's search and sort state in the summary page should be preserved.
- We chose to use a different view object for the Update and Search pages. We did this because "in real life" summary view objects are often far smaller than view objects designed to include all the attributes of a typical Oracle E-Business Suite entity object. We recommend that you use two different view objects for these cases.
- Per the UI guidelines, we display a Confirmation dialog at the end of the multistep transaction, and then display the read-only summary page with the user's cached search state. We do not re-query.

  Note: The changes are not reflected in the result set.
- If the user does any non-standard navigation, or abandons a transaction before returning to the summary page, it's important that the user does not encounter partial (invalid) entity objects in the middle tier cache.

Browser Back Button Navigation Events

Navigation Event	Navigation Event Description	Post-Navigation Application Behavior
B1	User abandons the update flow and navigates back to the summary page (Page 1) using the browser Back button.	No visible impact to the user. They can perform whatever actions they want on this page. Active browser Back button protection is required to ensure subsequent transactions perform correctly. If, for example, the user presses the **Update <*Object*>** button again, the user should not encounter partial (invalid) objects in the middle tier cache.
B2, B3, B3a	User moves back to previous Update steps within the flow using the browser Back button.	No visible impact to the user. They can perform whatever actions they want on this page.
B4	User navigates back to the Update flow from Confirmation dialog using the browser Back button.	Case 1: Update transaction cannot be resumed: If the user tries to make changes here and submits the form, an error dialog displays. Active browser Back button protection is required to prevent subsequent save actions on the Update pages once a transaction is committed. Case 2: Update transaction can be resumed. No visible impact to the user. They can perform whatever actions they want on these pages.
B4a	Following the B4 navigation, the user presses the browser Back button multiple times (without selecting any other buttons on the Create pages) to navigate to the originating page (Page 1).	No visible impact to the user. They can perform whatever actions they want on this page.
B5	After selecting the **Cancel** button from within the Update flow to abandon the transaction, the user returns to the Update flow using the browser Back button and tries to resume the transaction.	Behavior should be as stated above for the B4 navigation event.

Implementation Instructions

Tip: This example assumes that you know how to implement a single step Update; it focuses exclusively on Back button handling. For a complete multistep Update example, see the Update lab in the ToolBox Tutorial.

The recommended solution for handling the Back button in the Update case depends on whether or not changes are allowed after the transaction commits.

Updates Allowed After Back Button Navigation

Updating a record multiple times is considered a "benign" action, so you generally shouldn't prevent Back button access to this kind of flow. You do, however, want to ensure that the user isn't surprised by abandoned objects in the middle tier cache while working on a subsequent transaction. To do this: follow all the instructions for the Create case above with the following differences:

- You must pass the primary key for the object to update as a URL request parameter.
- There is no need to perform the `isBackNavigationFired()` check in the Update page's `processRequest()` logic. You will simply start the update transaction unit.
- You don't need to worry about the `isFormSubmission()` check before finding your row for update purposes. This is needed only when creating new rows. However, do pay close attention to the View Object Initialization Guidelines to ensure that you don't inadvertently lose user data.

For step-by-step instructions, see the Updates Not Allowed example below. The only difference between these two scenarios is you don't the need to include the `isBackNavigationFired()` check as shown in Step 1.

Updates Not Allowed After Back Button Navigation

To disallow updates if the page is accessed in an "abnormal" page flow, for example, once a purchase order is updated it is submitted to an Approval workflow for processing so the user cannot make subsequent changes once she commits); you should follow all the instructions for the multistep Create case above with the following differences:

- As mentioned above, you must be certain to pass the primary key for the object you want to update as a URL request parameter.
- You don't need to worry about the `isFormSubmission()` check before finding your row for update purposes. This is needed only when creating new rows. However, do pay close attention to the View Object Initialization Guidelines to ensure that you don't inadvertently lose user data.

In this case, you must still perform the `isBackNavigationFired()` check as in the Create case to disallow further updates once a change is committed.

Step 1: Follow all the general coding standards as described in Supporting the Browser Back Button.

Step 2: Define your Update page flow.

Step 3: Add a controller to the first page in the multistep Update flow with the following `processRequest()` logic.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);
```

```
  // This test checks for valid navigation from the Update icon.
  // It also checks to see if the user navigated to this page
  // using the OANavigationBarBean Back button (in this case, when we have a
  // valid navigation back to this page from a subsequent page within the
  // multistep transaction, we don't want to reinitialize the employee
  // and lose the user's work).
  if (!pageContext.isBackNavigationFired(false) &&
       (!"goto"Equals(pageContext.getParameter(EVENT_PARAM))))
  {
    // We are indicating that we are at the state of the Update transaction.
    TransactionUnitHelper.startTransactionUnit(pageContext, "empUpdateTxn");

    // We'll use this at the end of the flow for a confirmation message.
    String empName = pageContext.getParameter("empName");
    pageContext.putTransactionValue("empName", empName);

    String empNum = pageContext.getParameter("empNum");
    Serializable[] params = { empNum };
    OAApplicationModule am = pageContext.getApplicationModule(webBean);

    // For the update, since we're using the same VO as the "Details" page, we
    // can use the same initialization logic.
    am.invokeMethod("initDetails", params);
  }
  else
  {
    // Guard against the user being able to perform a form submit on this page
    // after the transaction is completed. If the user navigates back to this
    // page using the browser Back button after successfully committing, the
    // OA Framework will detect this navigation and step through processRequest()
    // first when the user tries to perform any action that causes the
    // form to be submitted. This will display a state loss error dialog
    // if this happens.
    if (!TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "empUpdateTxn", true))
    {
      // Please use a custom message for the dialog page! The more explicit,
      // the better.
      OADialogPage dialogPage = new OADialogPage(STATE_LOSS_ERROR);
      pageContext.redirectToDialogPage(dialogPage);
    }
  }
} // end processRequest()
```

Step 4: Add a controller to the subsequent pages in the multistep Update flow with the following `processRequest()` logic.

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  // Guard against the user being able to perform a form submit on this page
  // after the transaction is completed. If the user navigates back to this
  // page using the browser Back button after successfully committing, the
  // OA Framework will detect this navigation and step through processRequest()
  // first when the user tries to perform any action that causes the
  // form to be submitted. This will display a state loss error dialog
  // if this happens.
  if (!TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "empUpdateTxn", true))
  {
    // Please use a custom message for the dialog page!
    OADialogPage dialogPage = new OADialogPage(STATE_LOSS_ERROR);
    pageContext.redirectToDialogPage(dialogPage);
  }
} // end processRequest()
```

Step 5: Add `processFormRequest()` logic as shown to handle the **Finish/Submit** and **Cancel** buttons in the flow.

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processFormRequest(pageContext, webBean);
  OAApplicationModule am = pageContext.getApplicationModule(webBean);

  // This button should only be displayed on the final page...
  if (pageContext.getParameter("Submit") != null)
  {
    am.invokeMethod("apply");

    // Indicate that the Update transaction is complete.
    TransactionUnitHelper.endTransactionUnit(pageContext, "empUpdateTxn");

    ...
  }
  else if (pageContext.getParameter("Cancel") != null)
  {
    // Note that the rollbackEmployee() method in the root UI application module
    // has an isDirty() check optimization so the rollback can be avoided
    // if it isn't required (this can be used for EO-based view objects only).
    // See oracle.apps.fnd.framework.toolbox.labsolutions.server.EmployeeAMImpl.java
    // in the ToolBox Tutorial for an example.
    am.invokeMethod("rollbackEmployee");

    // Indicate that the Update transaction is complete.
    TransactionUnitHelper.endTransactionUnit(pageContext, "empUpdateTxn");
```

```
    ...
  }
} // end processFormRequest()
```

Step 6: Add `processRequest()` logic to the Summary page as shown:

```
public void processRequest(OAPageContext pageContext, OAWebBean webBean)
{
  // Always call this first.
  super.processRequest(pageContext, webBean);

  OAApplicationModule am = pageContext.getApplicationModule(webBean);

  // Note that we pass "false" as the final parameter to the isTransactionUnitInProgress()
  // method because the current page is not part of the transaction flow.
  if (TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "empUpdateTxn", false))
  {
    am.invokeMethod("rollbackEmployee");
    TransactionUnitHelper.endTransactionUnit(pageContext, "empUpdateTxn");
  }
  ...

} // end processRequest()
```

Summary and Transaction Pages in Inter-Application (Cross-Product) Page Flows

For inter-application (cross-product) page flows, the Summary page and the drilldown transaction page may use different root application modules. Therefore, the Summary page cannot rollback the transaction for the drilldown transaction page as in other use cases. For this case, application teams should consider sharing a root application module for these flows by "nesting" in the other product's shareable region. However, if this is difficult and requires too much coordination, follow all the instructions for the read-only summary page with transaction use cases above. In other words, add the logic to start and end the transaction units and redirect to an error page when the transaction unit ends with the following differences:

- Before starting a transaction unit, perform a rollback as follows to clear any pre-existing transaction state in the drilldown transaction page's controller `processRequest()` method. In other words, move the rollback logic from the summary page to the drilldown transaction page to lazily handle the rollback. For multistep transaction case, the logic below would be only for the first page.

  ```
  // Perform this rollback right before and only before the
  // startTransactionUnit call to clear pre-existing transaction state.
  //
  // Note: Although the transaction page participates in a transaction unit,
  // note that we are passing "false" for the last parameter in
  // isTransactionUnitInProgress call for this particular workaround as an exception case.
  // The reason is that, as in the summary page, we want to trap any pre-existing translation unit
  // instance for the named logical transaction and not just the currently active transaction unit
  // instance.
  //
  // For other isTransactionUnitInProgress calls in the drilldown transaction page,
  // "true" value should be passed into this parameter as usual.
  if (TransactionUnitHelper.isTransactionUnitInProgress(pageContext, "empCreateTxn", false))
  {
    am.invokeMethod("rollbackEmployee");
    TransactionUnitHelper.endTransactionUnit(pageContext, "empCreateTxn");
  }

  TransactionUnitHelper.startTransactionUnit(pageContext, "empCreateTxn");

  ...
  ```

- Since the Summary page and the transaction page use different root application modules:
  - Summary page should *not* have any rollback logic for the transaction page's transaction unit in its controller `processRequest()` method.
  - The transaction page can just commit or rollback the changes.

Updateable Summary Page w/ Actions in Table (Duplicate & Remove)

This section describes a scenario that you can use as a template for how to address task-specific actions in an updateable page.

Scenario Description

An updateable summary page has a table of rows with the following characteristics:

- The table control bar section includes a **Duplicate <*action*>** button, which enables users to duplicate one or more selected rows. When the user clicks the **Duplicate** button, the duplicated rows show up at the end of the same table. Internally, these duplicated rows have new primary key values. The duplicate action is not committed until the user clicks the **Apply** button, and it does not involve any drilldown page to perform the action.
- The table control bar section also includes a **Delete <*action*>** button, which enables users to delete one or more selected rows. When the user clicks the **Delete** button, the selected rows are removed from the table. The Delete action is not committed until the user clicks the **Apply** button, and it does not involve any drilldown page to perform the action.
- The summary page has **Apply** and **Cancel** buttons, which control whether the duplicate and remove actions are saved or discarded. Therefore, users can duplicate and delete rows as many times as they wish, and changes are committed when they select **Apply**. Similarly, changes are discarded when they select **Cancel**.
- Assume the user performs a duplicate action, then navigates with the browser Back button and tries to duplicate the same row. This is a benign navigation; the duplicate action should perform without any problems.
- Assume the user performs a Remove action, then navigates with the browser Back button and tries to remove the same row. In this scenario, a "stale data" error displays because it follows the general rule that delete-type transactions should not be repeatable.

- If the user both duplicates a row and removes a row, and then navigates with the browser Back button, any form submit action on the page will cause the "stale data" error to display due to the removed rows.

Key Assumptions

- The **Remove** and **Duplicate** icons/buttons submit the form.

Implementation Instructions

This section does not describe how to implement a duplicate or remove action; it focuses exclusively on what you need to do for Back button support.

Follow all the general coding standards as described in Supporting the Browser Back Button. If your application code observes these guidelines, OA Framework will provide automated browser Back button protection.

Partial Rage Rendering (PPR) Page Refresh

See the Dynamic User Interface document for a description of partial page rendering Back button navigation behavior.

From a coding perspective, following the general Back button standards offers the best protection against unexpected behaviors when coupling Back button navigation with PPR events.

Advanced Java Entity Object Development Topics

Overview

This document describes scenarios and techniques for solving more advanced coding challenges in the model layer.

Content

- Polymorphic Entity Objects
- Coordinating Foreign Key Changes for Associated and Non-Composite Entities
- Posting Parent Entities Before Children (Non-Composition)
- Validating Parent Entities Before Children (Non-Composition)
- One-Time Entity Group Validation
- Validating Entities of the Same Type in a Prescribed Order
- Persisting entity-derived attribute values after a commit

Prerequisite Reading

This document assumes that you have read the following in the OA Framework Developer Guide:

- Chapter 3: Implementing the Model
- Chapter 5: Implementing Java Entity Objects

Polymorphic Entity Objects

This illustrates how to use a discrimator attribute in a view object to instantiate the right entity object. For example, you have a "document" superclass and you need to instantiate 1 of *n* different document types such as purchase order, requisition, blanket agreement and so on.

1. Define the superclass entity object with a discriminator column.
2. Define the subclass entity object with default values for the discriminator column.
3. Define the view object to map to the superclass entity object and add the subclass entity objects as "Subtypes".
4. Create a new row, using the `createAndInitRow` method to tell BC4J which entity object subclass to instantiate. Do this by specifying a value for the discriminator column.

Coordinating Foreign Key Changes for Associated and Non-Composite Entities

In this scenario, logical parents need to propagate foreign key changes to associate (non-composite) children. The easiest place to this kind of coordination is in the parent's `postChanges` method. Each logical parent knows about its children and the attributes it needs to propagate.

Assumptions

- You know the foreign key values on the parent key before modifying them. This allows you to navigate to the children to also update their FK values.

  Note: Usually the parent's FK values are populated in the `postChanges` cycle for new entities after returning from a stored procedure call, or a database sequence being populated.

- The parent is always posted before the children. Instructions for controlling the posting order are provided in the next section.

Example

In this example, we assume that the parent's foreign key is updated in the `postChanges` cycle:

```
/**
 * This is a private variable in the entity class to cache employees before posting Department.
 */
private RowIterator newEmpsBeforePost = null;
public void postChanges(TransactionEvent e)
{
   if (getPostState()==STATUS_NEW)
   {
   newEmpsBeforePost = getEmployees();

   // Please note that getEmployees() will do a DB roundtrip to execute a query on employees and
   // then merge the data with employees in the cache.
   // If you know for sure that all your employees are in the cache, then you can just iterate
   // the entity cache using the current deptNo which will be much faster. The code would be
   // slightly different if you decide to take this route.
   }  else
      {
      newEmpsBeforePost = null;
      }
   //Will call doDML() which will populate the actual FK values.
   super.postChanges(e);
}
```

```
/**
 * BC4J framework invokes this method to allow a newly inserted master entity
 * to update its new detail entities. Typically you do not have to override
 * this method because the base implementation in oracle.jbo.server.EntityImpl
 * will automatically handle this refreshing when detail entity instances
 * are related to new "master" entity instances by composition.
 */
protected void refreshFKInNewContainees()
{
Number newDeptno = getDeptno();
if (newEmpsBeforePost != null)
  {
  while (newEmpsBeforePost.hasNext())
    {
    EmployeeImpl curEmp = (EmployeeImpl)newEmpsBeforePost.next();
    curEmp.setDeptno(newDeptno);
    }
  newEmpsBeforePost = null;
  }
}
```

Updating the primary key of an existing entity and propagating changes to associations:

If the primary key is updated via the setter method of an entity (a common case), then cascade update should be implemented in the setter method itself:

- Get the row set iterator (RSI) for the association.
- Execute the RSI and cache it (keep a handle to it) in a local variable.
- Set the attribute value on the super entity.
- Run through the list of associated details in the RSI and cascade-update the primary keys to the new values.

If the foreign key update is done in the database, then you must to perform the above caching of detail iterators in the `doDML` method and perform cascade update after `super.doDML`. However, if the details are not fetched, leave it to the database to perform cascade update via some update procedure called from `doDML`. In such cases:

1. Post all the relevant details.
2. Post the master.
3. Call the stored procedure to do the cascade update.
4. Drop and re-query all the details.

Posting Parent Entities Before Children

For associated entities (non-composite), parent entities sometimes need to be posted before children because:

- There are database constraints requiring that parent records be inserted before the children.
- For PL/SQL entities, validation logic in the `insertRow` or `updateRow` procedures assumes that the parent is already posted to the database.

In each of the children's `postChanges` we access the parent and post it first:

```
/**
 * postChanges() in EmployeeImpl. Employee is a child for Department.
 */
public void postChanges(TransactionEvent e)
{
   DepartmentImpl parentDept = getDepartment();
   // If the associated Dept instance is modified, post it first
   if (parentDept!=null
   (parentDept.getPostState()==STATUS_NEW ||
   parentDept.getPostState()==STATUS_MODIFIED))
   {
     parentDept.postChanges(e);
   }
   super.postChanges(e);
}
```

Note: If your product entity object `postChange()` code explicitly invokes `postChange()` for other entity objects, then your `postChange()` code must also take care of `postChange` error handling. Overwrite the `handlePostChangeError` method, which also invokes `handlePostChangeError` for those other entity objects, so that the state of those entity objects revert back correctly after the `postChange` error is raised.

Validating Parent Entities Before Children (Non-Composition)

You can force parents to be validated before their children in a non-composite association by overriding `validateEntity` for the children as follows:

```
/**
 * validateEntity() of EmployeeImpl. Employee is a child of Department.
 */
```

```
public void validateEntity()
{
 DepartmentImpl parentDept = getDepartment();
 // If the associated Dept instance is modified, post it first
 if (parentDept!=null)
   (
   byte state = parentDept.getEntityState();
   if (!parentDept.isValid()
   (state==EntityImpl.STATUS_NEW || state==EntityImpl.STATUS_MODIFIED))
      {
      parentDept.validateEntity();
      }
   }
 // The Employee validation
 .....

 super.validateEntity();
}
```

One-Time Entity Group Validation

Developers often need to perform group validation on modified entities of the same type in a transaction. This validation can be expensive and needs to be executed once in the transaction's validation cycle, or when called explicitly.

Note: This applies to parent entity objects only. For children, the "group" logic belongs in the parent.

Step 1: Define Custom Validator Which Implements ValidationListener

```
public class CustomGroupValidater implements ValidationListener
{
 DBTransactionImpl mTransaction;
 EntityDefImpl mEntityDef;
 boolean mIsValid;
 boolean mBeingValidated;

public CustomGroupValidater (DBTransactionImpl trxn, EntityDefImpl entityDef)
   {
   mTransaction = trxn;
   mEntityDef = entityDef;
   mIsValid = false;
   mBeingValidated = false;
   }
public void validate() throws JboException
   {
   // So that only the first entity calls the group validation.
   if ( mBeingValidated )
   return;
      try
      {
      mBeingValidated = true;
      // The group validation might require the entities to be valid, so
      // it could loop the entity cache to validate invalid ones.
      Iterator iter = mEntityDef.getAllEntityInstancesIterator(mTransaction);
      while ( iter!= null  iter.hasNext())
         {
         EntityImpl cachedEntity = (EntityImpl)iter.next();
         byte state = cachedEntity.getEntityState();
         if ( state==EntityImpl.STATUS_DELETED || state==EntityImpl.STATUS_DEAD ||cachedEntity.isValid())
         continue;
         cachedEntity.validate();
         }
      // Do your group validation here
      ....
      // Mark as valid
      mIsValid = true;
      }
      finally
      {
      mDontEnter = false;
      }
   }
   public boolean isValid()
   {
   return mIsValid;
   }
   protected setInvalid()
   {
   mIsValid = true;
   }
}
```

Step 2: Add a Custom Validator Accessor to the Entity Expert

The Entity Expert has one instance for each entity definition per transaction. Therefore, if you store the custom validater in the entity expert then only one instance of the validator will be available per entity definition/transaction. With this approach, you can get the custom validator and explicitly call `validate` on it, or you can just leave the transaction validation cycle do its job.

```
// A local variable in the entity expert class to keep the custom validator.
private ValidationListener mCustomGroupValidater;

/**
 * This method always returns the same CustomGroupValidater Instance.
 */
public ValidationListener getCustomGroupValidater()
{
   if (mCustomGroupValidater == null)
   {
     mCustomGroupValidater = new CustomGroupValidater(getOADBTransaction(), getEntityDefImpl());
   }
    return mCustomGroupValidater;
}
```

Step 3: Associate the Validation Listener with validateEntity

```
public void validateEntity()
{
  //Call the group validation
  EntityDefImpl eDef = getEntityDef();
  OADbTransaction trxn = getOADBTransaction();
  EmployeeExpert expert = (EmployeeExpert)trx.getExpert(eDef);
  CustomGroupValidater groupValidater = expert.getCustomGroupValidater();

  if (!groupValidater.isValid())
  {
  groupValidater.validate();
  }
  else
  {
  super.validateEntity();
  // Your Entity Level validation goes here
  ....
  }
}
```

Step 4: Override Entity `setInvalid`

Finally, you need to override the entity's `setInvalid` method to invalidate the group validator

```
protected void setInvalid()
{
 super();

 EntityDefImpl eDef = getEntityDef();
 OADbTransaction trxn = getOADBTransaction();
 EmployeeExpert expert = (EmployeeExpert)trx.getExpert(eDef);
 CustomGroupValidater groupValidater = expert.getCustomGroupValidater();
 groupValidater.setInvalid();
}
```

Validating Entities of the Same Type in a Prescribed Order

Although rare, you may need to validate modified entities of the same type in a prescribed order (BC4J does not guarantee any particular validation order). In this case, the validator must be called separately from `validateEntity()` of other entities as follows:

Tip: This applies for parent entity objects only. For children entity objects, the association iterator can be ordered by setting the `ORDER BY` on the iterator before getting the first child.

Assumptions

- There is only one Entity Expert for each entity type in a transaction.
- This code assumes that all entities are already in the transaction entity cache.
- The server side validation in the UI is usually disabled.(Defer it until commit time).

```
public void validateEntity()
{
  EntityDef eDef = getEntityDef();
  OADbTransaction trxn = getOADBTransaction();
  EmployeeExpert expert = (EmployeeExpert)trx.getExpert(eDef);

  // Will loop entities of the same type in entity cache and validate them in a certain order.
```

```
  if (validateEntitiesInOrder())
      return;

  // Entity Validation logic goes here.
  ...
  super.validateEntity();
```

In the Employee Expert you'll add:

```
// A flag to indicate that we are in the ordered validation method
private boolean mInOrderedValidation;

/**
 * Loops entities of the same type in entity cache and validates them in a certain order.
 * This method will not return if called from within itself.
 * @returns true if the entity's validation needs to be skipped.
 */
public boolean validateEntitiesInOrder()
{
   try
   {
   // If called from within itself then just return.
   if (mInOrderedValidation)
      return false;

   //Set the flag to start validating in order.
   mInOrderedValidation = true;
   OAEntity entities[] = ...

   // Get entities from EntityCache
   while ( loop entities in specific order)
      {
         ...
      // Validate the entity
      entities[i].validateEntity();
      }
   }
   finally
   {
   //Reset the flag
   mInOrderedValidation = false;
   }
      return true;
}
```

Persisting Entity-Derived Transient Attribute Values After Commit

Entity-derived transient attribute values may become lost when a view object row set is refreshed after a commit. The reason being is that once `Transaction.commit()` posts and commits the pending changes in entity objects, it removes the processed entity objects from its listener list. If the view object row set gets refreshed to contain a different set of rows, the view object no longer holds references to these entity objects in an unmodified state. The only references to these entity objects are the *weak* references from their entity object caches.

To make entity-derived transient attribute values persist, insert the following code in your `EntityImpl` subclass. (For example, when you have a mutable entity-derived transient attribute and cannot calculate this attribute value in the attribute getter method).

Warning: This code pins the entity object in the entity object cache causing an increase in the size of the entity object cache.

```
import oracle.jbo.server.TransactionEvent;

public void afterCommit(TransactionEvent e)
{
  super.afterCommit(e);

  // This assumes that you have three transient attributes
  // named TransAttr, TransAttr2, and TransAttr3.
  populateAttributeAsChanged(TRANSATTR, getTransAttr());
  populateAttributeAsChanged(TRANSATTR2, getTransAttr2());
  populateAttributeAsChanged(TRANSATTR3, getTransAttr3());

  addToTransactionManager();
}
```

Controlling UIX Rendering Output

Overview

This document describes how to control the look-and-feel and/or facet used to render to your pages.

- A **Look-and-Feel (LAF)** controls the appearance of an OA Framework application. An LAF provides the rendering logic for each component along with a Look-and-Feel-specific style sheet. This means that the LAF controls both the content that is generated for each component and the visual properties (colors, fonts, borders, etc...) used to style the component.
- You can also implement optimized variations of the LAF for a specific output medium by setting a **facet**. Facets typically modify the implementation of a small set of components, but in general share the bulk of the LAF's rendering code. For example, the printable version of a general browser page excludes superfluous navigation and personalization controls that would consume space and clutter the printed output. The printable version of a page is implemented as a facet.

Contents

- Controlling the Look-and-Feel
  - Touch Device Considerations
- Controlling the Facet

Controlling the Look-and-Feel

OA Framework provides the following LAFs:

- **Swan Desktop Look and Feel** -- implements the OA Framework Swan LAF for Release 12.0 and Release 12.1.
- **Skyros Look and Feel** -- implements the OA Framework Skyros LAF for Release 12.2.3 and above.
- **Alta Look and Feel** -- implements the OA Framework Alta LAF for Release 12.2.5 and above.
- **Minimum Look and Feel** -- generates "minimal" content to reduce the size of HTML pages and overall network overhead (for example, this LAF uses fewer images than the other versions).
- **Simple Desktop Look and Feel** -- a Look-and-Feel built on top of Base LAF to offer more customization features, and to serve as an illustration of how to create LAF extensions on top of a base LAF.

The Look-and-Feel of OA Framework-based pages is controlled by the Oracle Applications Look and Feel profile option. By setting this centrally, customers can choose which seeded LAF is most appropriate for their uses cases, and they can quickly adopt a new, custom LAF.

Attention: Each release of Oracle E-Business Suite is associated with a standard LAF / skin as shown in the table below. Oracle only supports Look-and-Feel issues on the standard LAF for that release, and not for any other LAF, including sample skins such as **Minimal Look and Feel** and **Simple Look and Feel** that are provided as basic references for custom skin development.

The following table summarizes the LAF(s) supported by OA Framework for a given release of Oracle E-Business Suite.

Oracle E-Business Suite Release	Supported Look-and-Feel
11.5.10	Browser Look and Feel
12.0 to 12.0.6	Swan Desktop Look and Feel
12.1.1 to 12.1.3	Swan Desktop Look and Feel
12.2.2	Swan Desktop Look and Feel
12.2.3	Swan Desktop Look and Feel and Skyros Look and Feel
12.2.4	Skyros Look and Feel
12.2.5 and above	Alta Look and Feel

Figure 1 displays the Advanced Table Sample Library page with the default **Alta Look and Feel** for Release 12.2.5. This LAF appears when the Oracle Applications Look and Feel profile option is not set (null) or if you set the profile value to **Alta Look and Feel**.

Figure 1. Advanced Table Sample Library page shown with Release 12.2.5 Alta Look and Feel. (Rich Table Interactions enabled)

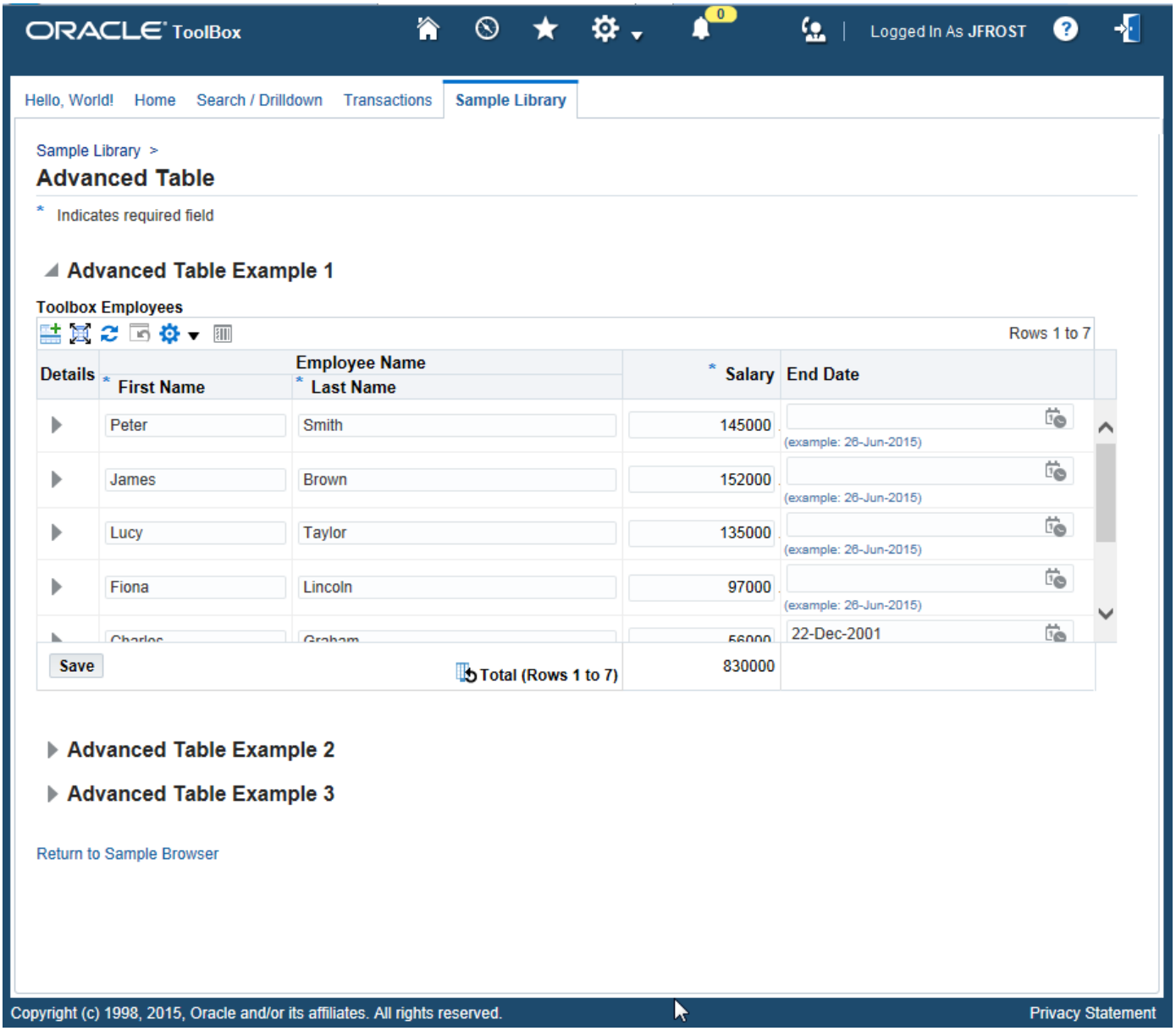


Figure 2 displays the Advanced Table Sample Library page with the default **Skyros Look and Feel** for Release 12.2.4. This LAF appeared when the Oracle Applications Look and Feel profile option is set to the profile value **Skyros Look and Feel** in Release 12.2.4.

Figure 2. Advanced Table Sample Library page shown with Release 12.2.4 Skyros Look and Feel. (Rich Table Interactions enabled)

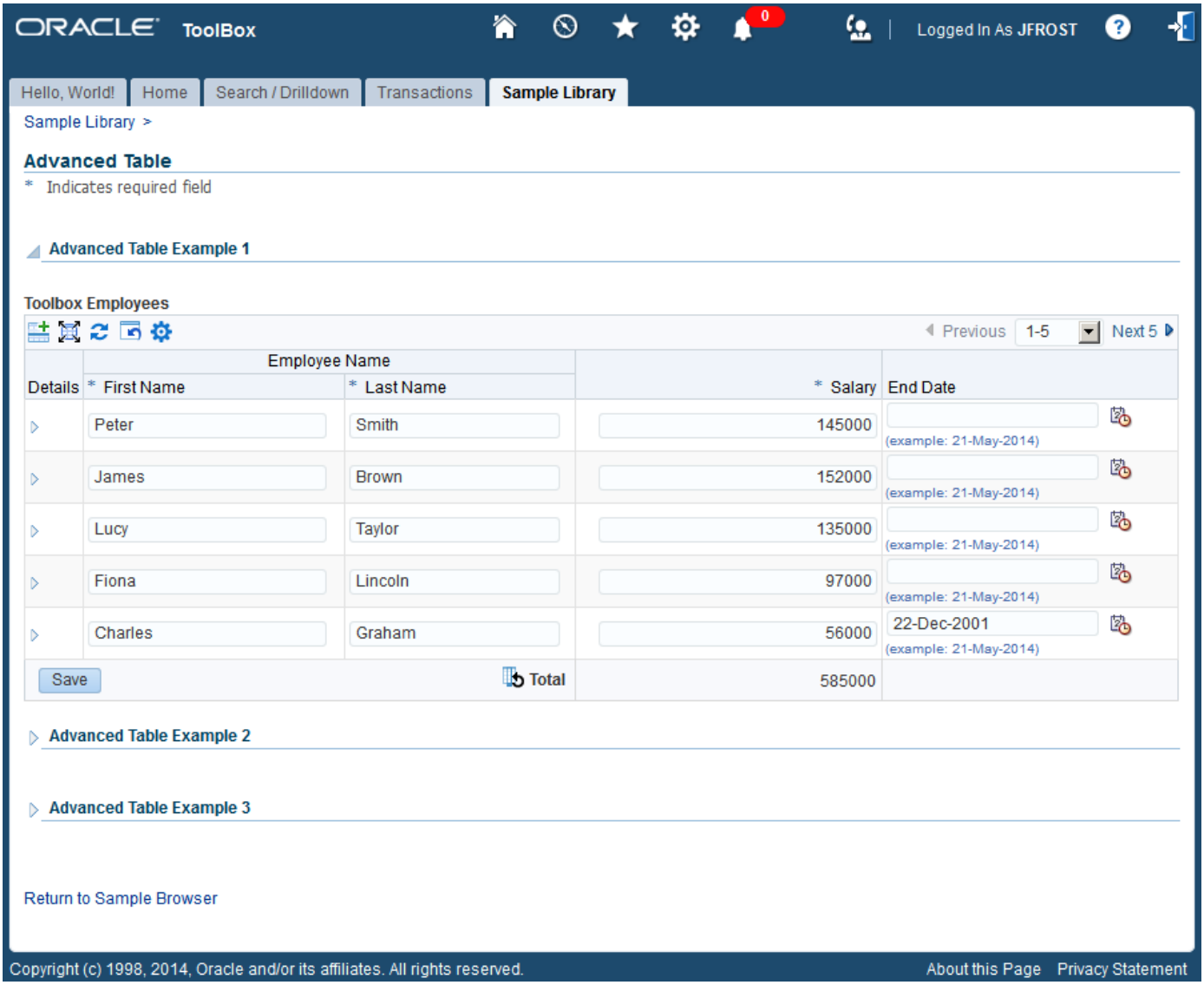
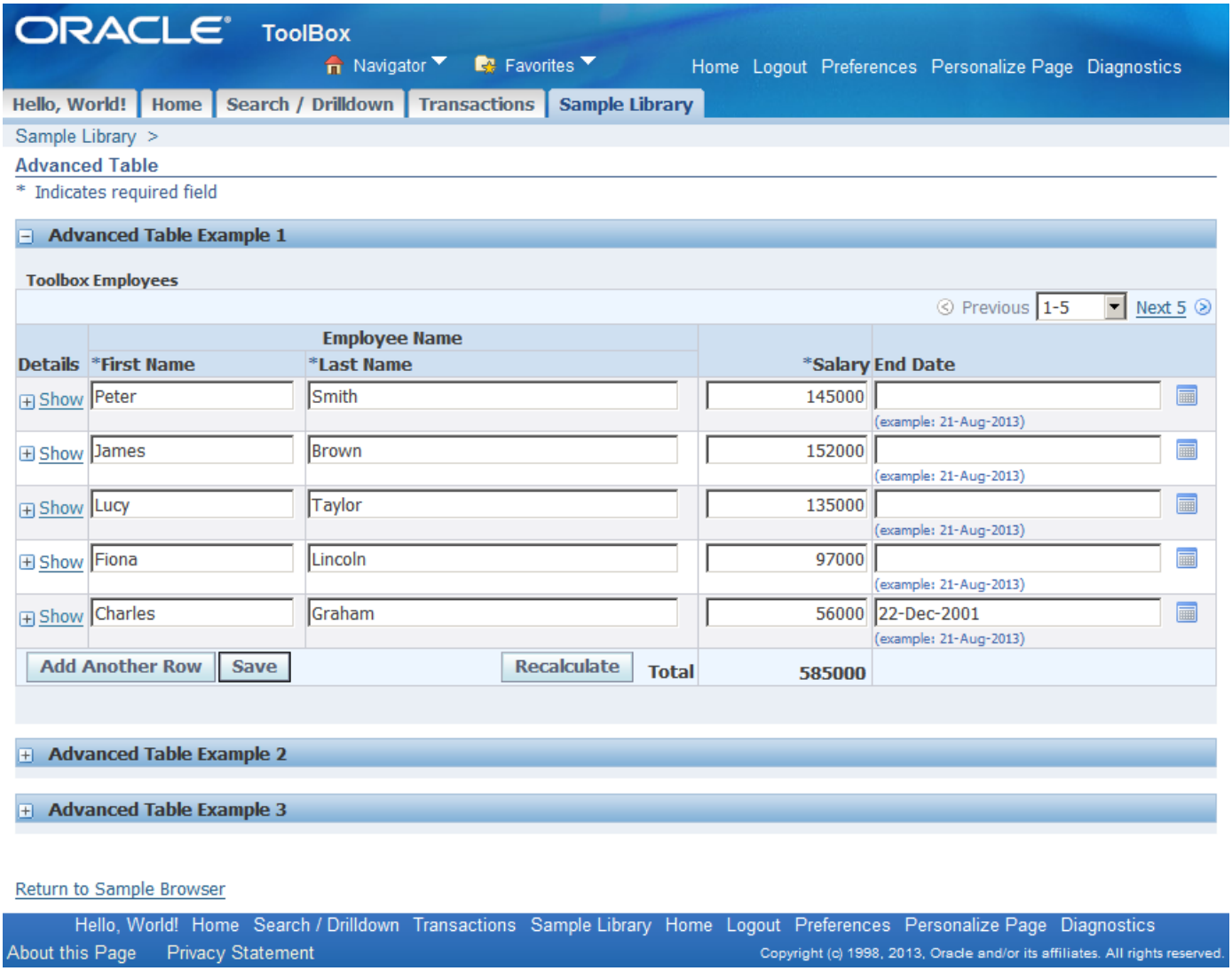


Figure 2. Advanced Table Sample Library page shown with Release 12.2 Swan Look and Feel.



To create a custom LAF or modify an existing custom LAF, use the Customizing Look and Feel UI.

Touch Device Considerations

See a summary of considerations for the Look-and-Feel on touch devices in the Mobile Applications topic.

Controlling the Facet

UIX (and the OA Framework) supports the following four facets for the **HTML** LAFs (when you use the plain text LAF, facets do not apply and are ignored if set).

- **Default** - renders HTML designed for a general purpose browser.
- **Email** - renders HTML suitable for a HTML-capable email client (it makes changes like deactivating JavaScript and inlining CSS styles as much as possible).
- **Portlet** - optimizes the imports of CSS style sheets and (where possible) Javascript libraries to improve performance when multiple portlets try to load the same style sheets and libraries in a single portal page.
- **Printable** - optimizes content for printing.

To enable these facets, set the `LafConstants.OA_RENDER_FACET` (or "`OARF`") to one of the following values as shown in the example below. The OA Framework ignores any invalid values for this parameter and uses the default facet for HTML LAFs.

Note: The `LafConstants` class is in the `oracle.apps.fnd.framework.webui.laf` package.

Constant	Constant Value	Description

`LafConstants.FACET_DEFAULT`	`default`	The default facet. **Note:** This is the default facet that is applied if no parameter value is specified, and the LAF renders HTML.
`LafConstants.FACET_EMAIL`	`email`	The email facet. Use this facet if you want to send HTML email (for example, the Oracle Workflow Mailer uses this facet for notifications).
`LafConstants.FACET_PORTLET`	`portlet`	The portlet facet. **Warning:** Do not use this facet. The OA Framework Web Portlet Provider will enable this facet for all OA Framework-based portlets.
`LafConstants.FACET_PRINTABLE`	`printable`	The printable facet. Set this parameter to identify a printable page. **Note:** The earlier approach of naming a "Printable Page" button "IcxPrintablePageButton" is still supported, but setting the facet is the preferred approach.

Example

See the Printable Page document for a complete example.

OA Framework and AOL/J Caching

Overview

You can leverage the AOL/J caching framework with your OA Framework applications. For detailed instructions, see the *Oracle E-Business Suite Java Caching Framework Developer's Guide*, My Oracle Support (formerly Oracle *MetaLink*) Knowledge Document 1108093.1.

Application Module and Connection Pooling

Overview

This document describes the OA Framework pooling architecture for BC4J application modules and AOL/J JDBC connections, including the mechanisms that you can use to monitor and tune your application module and JDBC connection pools for optimum performance.

Contents

- Architectural Overview
- Application Module Pooling
  - Pooling Process
    - Checkout
    - Checkin
    - Cleanup
  - Configuring the Application Module Pool
- Monitoring the Application module Pool
  - Accessing the AM Pool Monitor
  - Inspecting the AM Pool Monitor
- JDBC Connection Pooling
  - Pooling Process
    - Checkout
    - Checkin
    - Cleanup
- Monitoring the JDBC Connection Pool
    - Accessing the JDBC Connection Pool Monitor
    - Inspecting the JDBC Connection Pool Monitor
- Troubleshooting & Diagnostics
- Frequently Asked Questions
  - Application Module Pool
  - JDBC Connection Pool
  - System Configuration & User Load

Prerequisite Reading

- Chapter 2 : OA Framework State Management -> Application Module Pooling
- Chapter 6: Supporting the Browser Back Button

Architectural Overview

Pooling is a mechanism where cached objects kept in a runtime object pool can be used and reused as needed by an application. Instead of creating an object when requested by an application and destroying it when released by the application, you can pool these objects and reuse them when required. This saves the creation time associated with the object and improves performance and scalability.

A pool manager manages the objects in the pool. When an object is requested, it looks for an available object in the pool. If there isn't any, it creates one, up to the maximum allowable objects. If it can't create an object, it waits for one to be released and depending on the implementation may eventually display an error indicating that it was unable to retrieve the object.

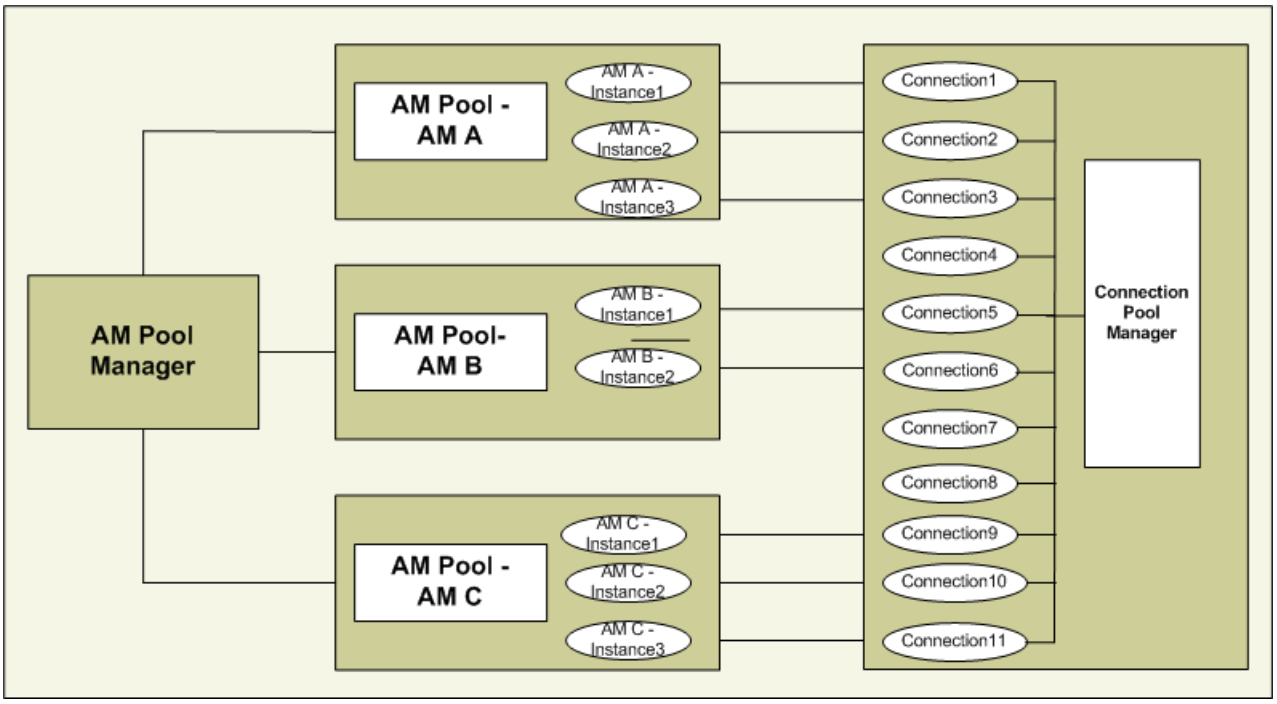
The OA Framework pools the application modules and the connections used by application modules. This chapter describes these two pools and the various options that configure each pool. You will also learn how to diagnose and troubleshoot performance and scalability issues associated with pooling in your application.

The Application Module (AM) instances are pooled in an Application Module Pool (also known as AM Pool). Each JVM has an application module pool manager that contains and manages individual application module pools. Each application module pool contains multiple instances of the same application module definition. In other words, a pool is created for each root application module definition (type) in your product. Application module instances within the pool are designated as being available for use, or unavailable (currently "checked out"). Only root application modules are pooled. Nested application modules are pooled as children of the root application module; they cannot be checked out or checked in independently of the parent.

In addition to pooling the AMs, the OA Framework relies upon AOL/J, which in turn uses the Oracle WebLogic Server (WLS) DataSource pool in Release 12.2 and above, to pool the connections used by them. The AMs reuse a pool of connections instead of creating a JDBC connection for every new instance and destroying it when the instance disconnects. A connection pool is assigned for each JDBC connection URL; currently in OA Framework, only one DBC file (and hence, only one JDBC connection URL) is allowed per installation of Oracle E-Business Suite.

Below is a pictorial representation of how the AM and the connection pools are connected to each other:

Figure 1: AM and Connection Pools used in the OA Framework



Application Module Pooling

Pooling Process

Checkout

When a page requires a root application module:

1. The OA Framework retrieves the pool manager for the JVM. If the pool manager doesn't exist, it will be created.
2. The OA Framework then retrieves the AM pool from the pool manager. If the AM pool doesn't exist, it will be created.
3. If the pool contains an unused (available) application module instance, it is "checked out" (meaning it is marked as being *unavailable (locked)* so others can't use it). If the pool doesn't have any available AM instances, a new instance is created and checked out. Note that the AM (via its transaction and the WebAppsContext object) obtains its jdbc connection from the AOL/J-managed WLS DataSource connection pool.
4. Upon checkout, BC4J ensures that the connection associated with an AM is in a usable state. If the connection is not usable, BC4J disconnects and reconnects the AM to request another connection from the AOL/J-managed WLS DataSource connection pool.

Note that the AM in the pool may not always have a connection associated with it. The AM pool determines whether the AM has a connection, and if it doesn't, gets one from the connection pool upon the AM checkout process.

OA Framework checks out a root application module for a page at the beginning of each request (in the JSP forward case, at the beginning of page processing).

Checkin

With Application Module Pooling enabled, after the page processing is finished, the root application module associated with the page can be checked back into the AM pool. When an application module is "checked in" to the AM pool, it is marked as being available so other user threads can use it. Application modules can be checked in with or without state.

Checkin Without State (Release)

When an AM is *released* (meaning it is no longer needed by a page), it is checked back into the pool *without state*. This means:

- View object data is cleared.
- The database transaction is rolled back.
- OA Framework clears any cached data on this AM and the transaction for the root AM. However, OA Framework does not clear any product-specific AM class member variables and you should avoid using them for this reason. If an existing product-

specific AM implementation already has class member variables defined, then you can clear them by overriding OAApplicationModuleImpl `beforeRelease()` method (or the deprecated OAApplicationModuleImpl `beforePoolCheckin()` method). See the OAApplicationModuleImpl `beforeRelease()` javadoc for more information. Additionally, the product-specific AM should also override the OAApplicationModuleImpl `afterReserve()` method to perform cleanups similar to those done in OAApplicationModuleImpl `beforeReserve()`. This ensures that any class member variable on this AM instance is cleaned up during AM pool check out again. Lastly, in the product implementation of the overridden method (`beforeRelease()` and `afterReserve()`), also be sure to invoke the base class method, that is, `super.beforeRelease()` or `super.afterReserve()`.

- The AM is marked as being available for use.

Checkin With State

An AM can be checked in with state managed. This means:

- All AM state (view object data, database transaction state, and so on) is managed without being cleared.
- The AM is marked as being available for use.

Checkin Without State vs. Checkin With State

- Checkin *without* state can be explicitly requested through `OAPageContext.releaseRootApplicationModule` call or retainAM parameter. (See Chapter 2 topic OA Framework State Management document for more details on the retainAM parameter.) Checkin *with state* cannot be explicitly requested through OAPageContext API or retainAM parameter.
- When the "Home" or "Logout" global button in a page is pressed, OA Framework implicitly checks in all the cached application modules *without state*.
- Upon servlet session timeout, OA Framework implicitly checks in all the cached application modules *without state*.
- At the end of each request (in the JSP forward case, at the end of page processing), OA Framework performs the following actions to the root application module of a page:
  - If an explicit AM request was requested through OAPageContext API, check in the AM *without* state.
  - If there was no explicit release request, as in the servlet session timeout case, leave the AM reserved (locked, unavailabled) for the current user thread.
- When an AM is checked out after a checkin *with state* for the same user session thread, the checked out AM will contain prior state. `OAApplicationModule.afterReserve()` hook method is entered for AMs checked out without state. `OAApplicationModule.beforeRelease()` is entered for AMs checked in without state.

How JDBC Connections Are Maintained for AMs

The JDBC connection stays with the AM until the connection is harvested (reclaimed back by WLS DataSource pool when WLS DataSource Connection pool harvester thread is initiated -- see JDBC Connection Pooling section below for more information on the harvester) or until the AM is destroyed, whichever comes first. You can override this default behavior of retaining a connection with the AM with the profile option, FND: Application Module Connection Pool Enabled to perform an eager release instead. By default, the connection is retained/dedicated with the AM for performance optimization.

For more information on the connection release strategy based on different AM retention levels, see the profile option document for FND: Application Module Connection Pool Enabled.

Cleanup

The OA Framework has an AM Pool Monitor cleanup thread that periodically destroys the idle AMs. Please read the section on Appendix B: Profile Options - Application Module Pooling on the different profile options that affect this.

Configuring the Application Module Pool

A special pool manager thread maintains the size of all the AM pools in a JVM by periodically destroying idle AMs. You can configure this process and other AM pool properties using the profile options described in OA Framework Profile Options: Application Module Pooling.

Monitoring the Application Module Pool

The AM pool monitor shows the active AM pools in the JVM.

The AM pool monitor shows several aspects of runtime and configuration information about the JVM. The five major aspects are listed below:

- **JVM Properties** - This includes information about JVM version, command line parameters, classpath information, BC4J

properties, and so on. The user is able to perform searches in properties, classpath or BC4J configuration. They can also export the classpath to a file. This is useful for both support and for cloning an environment.
- **JVM Memory** - This shows runtime information about heap usage in the JVM. It also allows the user to force a garbage collection cycle and see how much memory was collected.
- **JServ Sessions** - This shows a list of JServ sessions along with creation time and last access time. This can be used to see how much activity the JVM is handling. It also shows if users are leaving their sessions idle for a long time.
- **AM Pools** - This shows a list of the different AM pools. You can drill down to see full details of every AM instance in any pool. This is a very important monitoring function and is discussed in more detail below.
- **Versions** - This shows version information about the different technology stack components in your environment. This replaces the OAInfo.jsp functionality as OAInfo.jsp is no longer supported. You are also able to retrieve version information about any oracle.apps java class file. This is useful for development when debugging customer issues and gives a more accurate version number since it attempts to access the class version information at runtime in the JVM rather than on the file system.

For each AM pool type, you can get the following vital details:

- total number of AMs instantiated in the pool.
- number of AMs that are locked
- number of AMs that are available for checkout.
- Number of AMs created.
- number of AMs removed.
- number of AMs checked-in.
- number of AMs checked-out.

You can also run SQL queries against the database to monitor the AM pool and its connections. In Release 12, you can see this information in the AM pool monitor correlation between AMs and database connections.

If you are running OA Framework Release 12, this information is available in the pool monitor AM Pool Instance Details page along with several other pool parameters described below:

- Availability - whether this AM instance is available for checkout or locked by an active user.
- Username - the username that is currently locking the AM or the one that last had it locked.
- Creation Time - time AM was created.
- Lock Time - time AM was last locked.
- Release Time - time AM was released back into the pool.
- Timeout Time - time AM was timed out; this happens when the user is inactive for a period of time.
- Time To Create - time it took to create the AM instance.
- View Links - number of view links instantiated in AM.
- View Objects - number of view objects instantiated in AM.
- Nested AMs - number of nested AMs instantiated in this root AM.
- SID - session ID for database connection used by this AM.
- Connection Logon Time - time connection used by this root AM was created on the database.
- Jserv Session - correlation between this AM and the JServ session the user is using.

The information shown above is instance-specific within a certain pool type. The page also displays additional information for the pool type that is pool-specific. This is listed below:

- Available - total number of available AMs in the pool for checkout.
- Unavailable -total number of unavailable AMs in the pool. AMs already checked out and locked by other users.
- Creations - total number of application modules (AMs) created in this pool.
- Removals - total number of application modules (AMs) removed from this pool.
- Checkins - total number of AMs checked in.
- Checkouts - total number of AMs checked out.
- Activations - total count of activations that occurred.
- Referenced AMs Reused - total number of AMs that the session had affinity to that got checked out.
- Referenced AMs Reused Recycled - total number of AMs that had affinity to another session that got checked out.
- Unreferenced AMs Reused Recycled - total number of AMs that had no affinity to any session that got checked out.
- Pool CheckOut Failures - total number of AMs that errored out upon checkout.
- Total AMs In Pool - total number of AMs currently in the pool.
- Max AMs In Pool - maximum number of AMs that were in this pool for the lifetime of the pool.
- Average AMs In Pool - average number of AMs that were in this pool for the lifetime of the pool.
- Total Available AMs In Pool - total number of available AMs in the pool for the lifetime of the pool.
- Average Unavailable AMs In Pool - average number of available AMs in the pool for the lifetime of the pool.
- Total Referenced AMs In Pool - total number of AMs that have affinity to some session.
- Sessions Registered With Pool - total number of SessionCookies that are registered with the pool. SessionCookies represent an ApplicationPool session. An HttpSession may reference more than one SessionCookie.
- Average Sessions with Referenced State - average # sessions with affinity to AMs.

Accessing the Application Pool Monitor

The new version of pool monitor is now accessible from the Diagnostics global button. To gain access, you need to set the FND: Diagnostics profile to Y for the user.

Once you log in, the Diagnostics button "Show Pool Monitor" option gives you access to the pool monitor functionality. The pool monitor interrogates iAS to get a list of load-balanced JVMs and if successful gives you the opportunity to access the different JVM pools. This functionality is only available with iAS 1.0.2.2.2 or higher. If you are using an earlier version or the JVM list can not be obtained, then you are only be able to inspect the pool in the current JVM you are connected to.

Inspecting the AM Pool Monitor

Here is a real world example (that is OA Framework 5.7-based, but is still applicable to OA Framework 12) that explains how you can inspect the AM pool monitor. Let us say that the monitor shows the following statistics:

- `AM Pool type 1: agsidbs3gsi3ap1521oracle.apps.icx.icatalog.shopping.server.ShoppingAM`
  `Number of AM instances: 4.`
- `AM Pool type 2: agsidbs3gsi3ap1521oracle.apps.ap.oie.server.WebExpensesAM`
  `Number of AM instances: 3.`

Each AM Pool type has a list of number of AM instances that are available within its pool. Each AM instance has a dedicated JDBC connection associated with it. So, in the example above you have 7 JDBC connections that are used by the different AMs.

You can click on a specific AM Pool type to determine the number of AM instances that are locked Vs the number of AM instances that are available for use. In our example above, when you click on the AM Pool type 2, you can see the following output:

```
Details for Pool:agsidbs3gsi3ap1521oracle.apps.ap.oie.server.WebExpensesAM

Connect String jdbc:oracle:thin:applsyspub/pub@(DESCRIPTION=(ADDRESS_LIST=(ADDRESS=
(PROTOCOL=TCP)(HOST=agsidbs3)(PORT=1521)))(CONNECT_DATA=(SID=gsi3ap)))

Details of User Data in App Module Pool
User Data No User data present

Details of Application Module instance availabilty
Total number of App Module Instances in Pool 3
Total number of Available App Module Instances 1

Name: agsidbs3gsi3ap1521oracle.apps.ap.oie.server.WebExpensesAM (#1)
Availability=YES
Application UserName=user1
Creation time=1077721829221
Lock time=1077733716146
Release time=-1
Timeout time=-1
Time taken to create(ms)=9

Name: agsidbs3gsi3ap1521oracle.apps.ap.oie.server.WebExpensesAM (#2)
Availability=NO
Application UserName=user2
Creation time=1077731669625
Lock time=1077733949086
Release time=-1
Timeout time=-1
Time taken to create(ms)=96

Name: agsidbs3gsi3ap1521oracle.apps.ap.oie.server.WebExpensesAM (#3)
Availability=NO
Application UserName=user3
Creation time=1077731827107
Lock time=1077733367039
Release time=1077733557993
Timeout time=1077733557993
Time taken to create(ms)=8
```

The output above gives the following information about every AM instance in the pool:

- If the **Availability** is set to **YES**, then it means that the AM is currently free and can be used by the next user requesting it. If set to **NO**, then it means that the AM is currently used by a user.
- **Application User Name** specifies the user name of the user that is using the AM.

Detecting an AM leak

You can detect if AMs are freed proactively by monitoring the **Availability** flag of the AM pool at runtime. You can detect for AM leaks

using the following process.

Step 1: Identify AM instances that are suspects for AM leaks -- any AM type that has more used AMs than available ones fall under this category.

Step 2: Identify the pages, user flows and transactions that use such instances.

Step 3: Run through the above user flows and transactions in a dedicated environment and inspect the availability of the AM at the same time using the pool. Note that you should use a dedicated environment and not a production environment for this test.

Inspecting the AM Pool using SQL

You can find out database connections associated with a specific AM by running the following query against the v$session table (Note, in Release 12, this is not needed since the information is also available in the pool monitor):

```
select s.sid, s.serial#,
s.module, p.spid,
s.program program, s.status
from v$session s, v$process p
where s.username is not null
and s.paddr = p.addr
and s.program like 'JDBC%'
```

The above SQL query results in the following sample output:

```
SID Serial# Module SPID Program Status
156 541 icx.icatalog.shopping.server.ShoppingAM 22441 JDBC Thin Clien INACTIVE
83 3873 icx.icatalog.shopping.server.ShoppingAM 23403 JDBC Thin Clien INACTIVE
91 871 icx.icatalog.shopping.server.ShoppingAM 22261 JDBC Thin Clien INACTIVE
17 761 icx.icatalog.shopping.server.ShoppingAM 23407 JDBC Thin Clien INACTIVE
46 667 icx.icatalog.shopping.server.ShoppingAM:R 22296 JDBC Thin Clien INACTIVE
68 1094 ap.oie.server.WebExpensesAM 22263 JDBC Thin Clien INACTIVE
144 408 ap.oie.server.WebExpensesAM 23405 JDBC Thin Clien INACTIVE
45 647 ap.oie.server.WebExpensesAM 23496 JDBC Thin Clien INACTIVE
```

The output above gives the following information about every AM instance in the pool:

- The number of connections that are used and the AM using the connection (the module contains the truncated AM name).
- The number of harvested connections -- If the AM name has a trailing ":R", then it means that the connection associated with the AM is available for use (or no longer locked). This will happen if the connection is harvested.
- The status column -- This column is set to ACTIVE only when your application is actively doing something with the database - like opening and reading from a cursor. Through the lifetime of an application using the OA Framework, the connection will be in the ACTIVE status for a very small percentage of time. In other words, even though the AM is holding the connection, the v$session table will not show it as being ACTIVE unless there is some database activity.

As you can see from the above output, this indicates that the WebExpensesAM has 3 connections used by 3 different instances (same as observed from the AM Pool Monitor).

JDBC Connection Pooling

As we stated above, the OA Framework pools JDBC connections used by AMs by using AOL/J WLS DataSource connection pool, in addition to the AMs themselves. The JDBC connection pool is a list of available connections, a list of locked connections, and an API that allows clients to borrow/return connections from/to the pool. The pool also maintains information about the current database session state of each available connection. The WLS DataSource Connection pool maintains the pool by shrinking (removing the extra connections from the pool when there is low load), harvesting (reclaiming the unused harvestabale connections from the client code back to the pool when the number of available connections in the pool falls below a certain trigger value), and creating new connections when requested. The harvester allows the marked connections from these AMs to be checked back into the connection pool (without destroying them) for reuse by other threads. All these pool operations are controlled by configuring pool properties with appropriate values.

Each JDBC connection in the pool is really a meta-connection object that wraps the connection and additionally records the current AOL and NLS database session state of the connection. The WLS DataSource Connection pool is managed by several administrator-configurable pool parameters. These parameters govern the pool size, cleanup, and types of safety checks the pool performs on connections. All parameters have default values that can be overridden and configured from the WLS Administrator Console for an Oracle E-Business Suite-defined DataSource.

Out of the box, Oracle E-Business Suite creates a dataSource with proper and well tested values defined for all the pool properties. Please DO NOT change the default values defined for these properties. You may only modify the Pool size controlling parameters (Initial Capacity, Maximum Capacity, Minimum Capacity) and harvesting parameters (Connection Harvest Max Count, Connection Harvest Trigger Count) if the default values do not meet your needs and load. For more information about WLS DataSource pool

properties, refer to the Oracle WebLogic Server's Administration Console Online Help topic JDBC Data Source: Configuration: Connection Pool.

Pooling Process

Checkout

- Every active AM has a dedicated JDBC connection. When the AM is created, it requests a JDBC connection from the WLS DataSource Connection pool.
- The WLS DataSource Connection Pool tries to select a usable connection from the list of available connections and returns one if found.
- If the selected connection is not usable, or if there are no available connections, then the pool checks to see if it has already reached its maximum size (governed by the Maximum Capacity parameter). If the pool is not at maximum size, it tries to create a new connection for the client.
- If the pool is already at maximum size, the JDBC connections are detached (harvested) from the inactive AMs and released back to the pool for reuse. WLS Connection pool triggers the harvesting mechanism as soon as it detects that the number of available connections in the pool is less than or equal to the value of the Connection Harvest Trigger Count defined for the datasource.
- If it cannot create a new connection, it records an error and returns null.

Checkin

Once a connection is checked out by the AM, it remains locked. Note that checking an AM into the AM pool does not check in the connection back to the connection pool by default. Instead, connections are lazily checked back into the connection pool in cases as below:

- when the AM is destroyed by the AM Pool Monitor cleanup thread
- when the AM needs to disconnect and reconnect upon checkout to acquire a usable connection in place of an unusable (closed or timed out) connection
- when the profile option FND: Application Module Connection Pool Enabled is set to Yes. -- see the profile option document for FND: Application Module Connection Pool Enabled for information on which connections are checked in when
- when the WLS DataSource connection harvester requests a connection from the AM.

The WLS DataSource Connection harvester is invoked only when the number of available connections in the pool are less than the Connection Harvest Trigger Count defined for the pool. However, there is a possibility of connection harvesting at the beginning of the pool life cycle when the total number of connections and hence the available connections in the pool itself are less than the harvest trigger count. When connection harvesting is triggered, the pool reclaims all connections marked as harvestable back to the pool. If there are no connections marked as harvestable, the harvester waits and then re-checks for connections that can be harvested. The pool manager creates a new connection if there is a scope to create a new connection to serve the client request.

Cleanup

The connection pool tries to maintain a number of available connections between the Minimum Capacity and Maximum Capacity values. When the usage is high, the pool size may grow considerably, and hence, the cleanup thread removes (destroys) connections. When the usage is low, the pool shrinks down below the buffer max size, and hence, the cleanup thread creates connections.

The cleanup happens over a period of time controlled by the Shrink Frequency and the Minimum Capacity. The value set for Minimum Capacity is used while making connection pool shrinking calculations. This is to ensure the pool contains the minimum connections as configured under 'Minimum Capacity' even when it is shrinking the pool.

Monitoring the JDBC Connection Pool

Accessing the Connection Pool Monitor

Oracle WebLogic Server provides an built-in tool to view and monitor the current statistics of a JDBC DataSource Connection Pool. You can access this statistics tool from the WLS Administration Console as defined under the Monitor DataSource Statistics page.

Inspecting the Connection Pool Monitor

You can inspect the connection pool monitor for the following metrics:

- Available Connections: The number of connections that are available for checking out.
- Locked Connections: The number of connections that were checked out by clients that use the JDBC Connection Pool. This includes the OA Framework AMs, other JSPs, servlets and so on.
- Pool Size Counter: Total number of available and used connections in the pool.
- Leaked Connection Count: Total number of connections that are leaked currently by the client code.

Troubleshooting & Diagnostics

Detecting a connection leak

The application modules only return connections back to the pool under certain conditions even if they are not actively using the connection as described above. You should hence check for any leaked connections.

The DataSource monitoring statistics page shows the number of leaked connections, if any. If there are leaked connections, you can use profiling to determine which application is the source of leaks. Enable connection leak profiling to collect information about threads that have reserved a connection from the data source and the connection leaked (was not properly returned to the pool of connections). Use this profile information to help determine which applications are not properly closing JDBC connections. The record contains the following information:

- PoolName - name of the data source to which this connection belongs.
- ID - connection ID.
- User - stack trace of the thread waiting for the connection.
- Timestamp - time stamp showing when the connection leak was detected.

Refer to the "Monitoring WebLogic JDBC Resources" chapter of *Configuring and Managing JDBC Data Sources for Oracle WebLogic Server 11g Release 1* for more information on leak profiling to detect source of leaked connections.

Tuning

When tuning your pools, you should note that there will be a tradeoff of number of connections and memory available vs. the performance of your system. You should try to minimize the number of expensive events given the existing resources.

The events are listed below in order of expense:

- WLS DataSource pool is unable to allocate connection (MAX_JDBC_CONNECTIONS exceeded) -- If this is the issue, then you should first try turning on connection pooling using the profile option FND: Application Module Connection Pool Enabled. Alternatively, you can try turning off your AM pool using the profile option FND: Application Module Pool Enabled. This will allow the connections to be eagerly released as well, but it will destroy the AM instance after use so it can't be reused again.

  You can increase the number of available connections and/or decrease the JDBC session timeout to minimize the events above. However the expense with this approach is that there will also be an increased expense of maintaining the system with more connections. A decrease in the JDBC session timeout will also affect the user experience.

- Harvesting of Released AMs (harvesting of 'high' rated connections) - how much idle time before harvesting?
- AOL/J context switch when new connection is requested

Please refer the "Tuning Data Source Connection Pools" chapter in *Configuring and Managing JDBC Data Source for Oracle WebLogic Server 11g Release 1* for details on tuning the pool using pool properties.

Detecting AM State Corruption

Introduced in Release 12.2.6, the `OAPageContext.getRootApplicationModule()` API now throws an `FND_STATE_LOSS_ERROR` with reason code `'Root Application Module State Corruption ... (AM) has already been released'` when a product page flow potentially accesses an AM that has already been checked back into the AM pool. Refer to Frequently Asked Questions (How do I know if the Application Modules are being released?) and the Application Module Pooling > Pooling Process > Checkout / Checkin sections for more information on how to correct this error.

Frequently Asked Questions

Application Module Pool

- How do I know if the Application Modules are being released?
- What is the difference between releasing an AM and destroying an AM - how do these affect the performance of a system?
- How do AM releases work with the "Logout" , and "Home" global buttons?
- How do AM releases work when I close the browser window?

- **How do I know if the Application Modules are being checked in?**

  Look for the following diagnostics in the error_log file:

  ```
  BC4J HTTP Container was timed out
  The binding listener for<:your_Application Module_name> was timed out
  ```

  This is an indicator that servlet session (HTTP session) is timed out and application module was checked in. If you would like to print a product specific message when your application module times out and gets checked in without state

(released), then you can do so by overriding the `beforeRelease()` method on your application module.

- **What is the difference between releasing an AM and destroying an AM - how do these affect the performance of a system?**

  When an AM is checked in without state, an AM will be destroyed if AMPOOLENABLED=No. If AMPOOLENABLED = Yes then the AM will be released (not destroyed). When AMPOOLENABLED=Yes, an unused (released) AM can be destroyed by an AM pool monitor cleanup thread. The AM pool monitor thread determines how to destroy the unused AMs using the following four profile options:

  - AMPOOL_MONITOR_SLEEP_INTERVAL
  - AMPOOL_MIN_AVAIL_SIZE
  - AMPOOL_MAX_AVAIL_SIZE
  - AMPOOL_MAX_INACTIVE_AGE

- **How do AM checkins work with the "Logout" , and "Home" global buttons?**

  When you click on the "Home" or "Logout" buttons, all AMs will be destroyed only if AMPOOLENABLED=No (AMs are not pooled). If AMPOOLENABLED=Yes (AMs are pooled) then the AMs will be released (not destroyed). The four profile options described about determine how to quickly destroy these released AMs.

- **How do AM checkins work when I close the browser window?**

  When you click on the close button, the AM is neither released nor destroyed. Eventually when the session times out AMs are checked in (not destroyed). The four profile options described above determine how to quickly destroy these released AMs.

JDBC Connection Pool

- When does a user get a a connection max-ed out error? What happens when the AM pool maxes out? How do AM reuse and JDBC connections work under high user load?
- What are the overhead connections that I should always expect?

- **When does a user get a a connection maxed out error? What happens when the AM pool maxes out? How do AM reuse and JDBC connections work under high user load?**

  If the AM pool remains maxed out it means that as soon as an AM becomes available, it gets immediately reused by the next incoming user. If you have a large number of users connecting to your system, then this can happen over and over again, until the user load decreases. Since the AMs get reused, if the connections are maxed out it doesn't mean that every user who tries to get on the system will get an error. The user will see the error only when the jdbc connections are maxed out and there are no AMs available that can be reused.

  Since there are no AMs to be reused any new user who tries to get in will try to create a new AM. A new AM means a new jdbc connection. But if the connections are maxed out then it is not possible to obtain a new connection. You will get an error in this case. Now if the jdbc connections are maxed out and there is one AM in the pool waiting to be reused - the next user who gets in will not see the error because he/she will reuse that one AM.

- **What are the overhead connections that I should always expect?**

  You should expect the following 'overhead' OA Framework specific connections in your system.

  - A connection dedicated to logging in (GWYUID connection).
  - If you have any AK based applications, then you will have a static AM created for fetching its metadata. This static AM will have a connection associated with it.
  - An Oracle Workflow Business Events Control Connection. There is one connection per JVM.
  - Transportation servlet inbound and outbound connections, that are configurable. By default, there are 5 outbound and another 5 inbound connections. The defaults can be changed by setting one or both of the following JVM parameters in jserv.properties:
    - `wrapper.bin.parameters=-DOXTAOutThreads=1 # for outbound connections`
    - `wrapper.bin.parameters=-DOXTAInPoolSize=1 # for inbound connections`

System Configuration & User Load

- How do I find out how many users are accessing my system at any given time?

- **How do I find out how many users are accessing my system at any given time?**

  There is no way to accurately determine the number of active OA Framework users on a system. The best you can do is determine the number of active Oracle E-Business Suite users. You can do that by querying the icx_sessions table where disabled_flag = 'N'. Also note that for one user entry in the icx_sessions table there could be many active AMs.

Advanced View Object Development Topics

Overview

This document describes scenarios and techniques for solving more advanced coding challenges pertaining to view objects. You may also wish to refer to View Objects in Detail for additional information.

Contents

- Entity Event Notification
- BC4J Native JDBC Statement Management
- Miscellaneous Implementation Techniques
  - Iterating View Object Rows
  - Copying Rows
  - View Object with Custom Data Source
  - Updating "Expert Mode" Rows with Multiple Outer Joins
  - Dynamic View Object Definitions
  - Setting View Object Where Clause
  - Master-Detail View Object Relationships

Entity Event Notification

This section explains how view objects are notified of entity events and what effects the event notification has on the notified view objects. This is an advanced topic that is included for informational purposes only. If you do want to understand this content, read the following documents first. Also read the above section on View Object Attribute Types and Caching.

- Java Entity Objects
- Entity Object and View Object Attribute Setters

Entity Event Notification Process

The event notification from entity objects to view objects is a two-step process involving entity events and view events.

- An entity event is an event generated by an entity object. A view event is an event generated by a view object.

  Note: View event is sometimes termed *row set event* or *row set iterator event* because it is associated with a particular view row set iterator (`oracle.jbo.server.ViewRowSetIteratorImpl`) instance. A view object owns a default row set, and the the default row set owns a default row set iterator.

  A view object can own multiple row sets and a row set can own multiple row set iterators.

- An entity event is represented by `oracle.jbo.server.EntityEvent` Java class. A view event is represented by the following classes in the `oracle.jbo` package:

  ```
  DeleteEvent
  InsertEvent
  NavigationEvent
  RangeRefreshEvent
  ScrollEvent
  UpdateEvent
  ```

- An entity cache sends out entity events to view objects by invoking the `oracle.jbo.server.ViewObjectImpl.sourceChanged(EntityEvent)` method with a specific entity event object. The `sourceChanged()` method then converts the entity event into a corresponding view event, and sends it to view row sets. Each view row set then sends the view event to view row set iterators. During this process, `ViewObjectImpl.notifyRow*()` methods and hook methods in the `oracle.jbo.RowSetListener` interface are invoked.

  `ViewObjectImpl.notifyRow*()` methods can be overridden by advanced BC4J users to respond to view events. A more common and public way to respond to view events is via `RowSetListener` objects. You can register your customized `RowSetListener` object with any row set iterator using the `RowSetIterator.addListener()` call to respond to view events associated with the row set iterator instance. `ViewObjectImpl.addListener()` delegates to `RowSetIterator.addListener()`.

  A view event can also be generated by a row set iterator when rows are navigated without any changes in the underlying entities.

- All view objects based on particular entity objects listen to entity events. (`ViewObjectImpl.sourceChanged(EntityEvent)` is called for all view objects that are based on the changed entity), but not all entity events produce corresponding view events. The following lists how entity events are converted to view events:

  - For an *attribute change/update* entity event, BC4J optimizes how view events are produced because attribute updates happen frequently for both newly created view rows and queried view rows:

    If any of the following conditions are met for a view object, and if the view object contains view rows that actually reference the changed entity (event source), BC4J converts the entity event to a corresponding view event and sends the view event to the view object:

    i. You explicitly add a `RowSetListener` instance to your view object with a call to `ViewObjectImpl.addListener()`.

    ii. The changed attribute is a foreign key joining two entity objects for a view object. For example, assume a view object `EmpDeptVO` joins `EmpEO` and `DeptEO`. If the `EmpEO.Deptno` foreign key attribute changes, `EmpDeptVO` receives and processes the view

event.

iii. The changed attribute is a foreign key attribute in a detail view object participating in a master - detail relationship via a view link. For example, assume an `EmpEO.Deptno` attribute was changed for an `EmpVO` which is linked to a `DeptVO` via a view link. `EmpVO` receives and processes the view event.

- For an *entity removal* event, all view objects with view rows that actually reference the removed entity (event source) will receive and process the view event. Since entity removal does not occur often, there is no further optimization in view event production behavior as in the case of attribute changes.
- For an *entity insertion* event, a new view row (`ViewRowImpl` instance) referencing the new entity (`EntityImpl` instance) will be added to a view object if the view object directly triggered the entity insertion event by a `ViewObjectImpl.insertRow(Row)` call or if the view object's association consistency was enabled (`ViewObjectImpl.isAssociationConsistent()` flag value was `true`). Additionally, for these view objects, if a `RowQualifier` was specified with a call to `ViewObjectImpl.setRowQualifier(RowQualifier)`, BC4J checks to see if the new row meets the `RowQualifier` conditions before inserting it into the view object. Only those view objects that ultimately insert the new view row receive and process the view event.

OAViewObjectImpl.isDirty( ) Flag Behavior

The `OAViewObjectImpl.isDirty()` method uses the view object's query collection's dirty flag (it delegates to `QueryCollection.isDirty()`).

Tip: A *query collection* is a collection of view rows. A view object centrally owns query collections and allows the query collections to be shared by view row sets that have the same query criteria values.

This flag is marked as being "dirty" in the following cases:

- An entity event was converted to a view event and was sent to a view object. The query collection of the view object that received the view event becomes dirty, thereby effectively making the `OAViewObjectImpl.isDirty()` flag value `true`.
- A view object performed insert, update, or delete operations by invoking `ViewObjectImpl.insertRow(Row)`, `ViewRowImpl.setAttribute()` / `set<AttributeName>()` / `setAttributeInternal()`, or `ViewRowImpl.remove()`. In this case, the query collection of the view object that directly triggered the insert, update, or delete operation is marked dirty.

There are some limitations with the `OAViewObjectImpl.isDirty()` flag behavior. See the Javadoc for `OAViewObjectImpl.isDirty()` for more information.

BC4J Native JDBC Statement Management

This discussion lists key points with regards to how BC4J closes JDBC statements for view objects. This information is helpful if you have a situation where you need to tune your memory usage or fix a statement leak.

BC4J caches the JDBC prepared statements by default in its own internal statement cache. This BC4J statement caching is separate from Oracle JDBC statement caching. The Oracle JDBC statement caching releases the statement to the JDBC statement cache when `Statement.close()` is called as opposed to actually closing the statement.

The following list identifies how BC4J closes these cached prepared statements stored in its own cache.

1. `oracle.jbo.server.ViewObjectImpl.clearCache()`:

- Closes the QueryCollection's JDBC ResultSet (cursor) - this action is accompanied by the resetting of the `QueryCollection.isExecuted()` flag to **false** to force query re-execution (and hence opens a new valid JDBC ResultSet (cursor)) upon subsequent row navigation methods.
- *Releases* the QueryCollection's prepared statement to the BC4J prepared statement cache - this action does not close the prepared statement.

2. `oracle.jbo.Transaction.clearEntityCache()`:

Neither releases nor closes the entity cache statements (such as insert, update, delete DML statements or lock statements).

3. `oracle.jbo.Transaction.rollback()`:

- A*lways* clears the view object caches in `oracle.jbo.server.ViewObjectImpl.afterRollback()`. Hence, the QueryCollection's ResultSet gets closed and the QueryCollection's prepared statement is released to the statement cache.
- Determines whether to clear the entity object cache based on `Transaction.isClearCacheOnRollback()` flag value (default is **true**), however, clearing the entity object cache does not release the entity cache statements.
- *Closes* cached lock statements in entity object cache only.

4. `oracle.jbo.server.ViewObjectImpl.isFetchComplete()`:

- Returns **true** when all the rows have been fetched from the database. Note that `ViewObjectImpl.setFetchMode(FETCH_ALL)` allows all the rows to be fetched at once.
- When the view object fetch is complete, the QueryCollection's ResultSet gets closed and the QueryCollection's prepared statement is released to the statement cache. It does not matter whether `ViewObjectImpl.isForwardOnly()` is set to true or not.

5. The view object QueryCollection's prepared statements are *released* to the cache in the following cases:

- `ViewObjectImpl.clearCache()` is called.
- `Transaction.rollback()` is called.
- The view object completes fetching all the rows and hence, `ViewObjectImpl.isFetchComplete()` returns **true**.
- `ViewObjectImpl.remove()` is called - all the QueryCollection's prepared statements are released back to the cache and then closed.

- Application module is released with its state reset, that is `oracle.jbo.server.ApplicationModuleImpl.resetState` is called. The method `resetState` calls `Transaction.rollback()`. All the QueryCollection's prepared statements are released back to the cache and then closed.
- `Transaction.disconnect()` is called (with either **true** or **false**) - all the QueryCollection's prepared statements are released back to the cache and then closed.

6. The entity object cache statements (insert, update, delete DML statements and lock statements) are *released* to the cache in the following cases:

- When the view object Entity listeners are removed during:
  - `ViewObjectImpl.setListenToEntityEvents(false)`
  - Application module release with state reset (in `ViewObjectImpl.resetSession`) - all the entity object cache statements are released back to the cache and then closed.
  - `ViewObjectImpl.remove()` -- all the entity object cache statements used by the view object are released back to the cache and then closed.
- `Transaction.disconnect()` (set to **true** or **false**) is called -- all the entity object cache statements are released back to the cache and then closed.
- The only **exception** to this is when the lock statement is closed upon `Transaction.rollback()` as well.

7. With the exception of the entity object lock statement, all the cached prepared statements in view object QueryCollections and entity cache are *closed* in the following cases:

- `ViewObjectImpl.remove()` is called - for entity object cache statements, all the entity object cache statements used by the view object are released back to the cache and then closed.
- Application module is released with its state reset when `oracle.jbo.server.ApplicationModuleImpl.resetState` is called).
- `Transaction.disconnect()` (set to **true** or **false**) is called.
- **Special case** - `oracle.apps.fnd.framework.server.OAPlsqlEntityImpl` statements cached in `oracle.apps.fnd.framework.server.OAEntityCache`:
  - All the cached prepared statements in view object QueryCollections and entity cache are closed by overriding `oracle.jbo.server.EntityCache.closeStatements()` so that these statements can be closed whenever entity cache statements are supposed to be closed.
  - The cached prepared statements in view object QueryCollections and entity cache are also closed in the following cases to reduce memory usage:
    - Entity cache is cleared.
    - `Transaction.rollback()` is called (in `OAPlsqlEntityImpl.afterRollback()`).
    - Application module is released with its state reset (in `OAPlsqlEntityImpl.afterRollback()`).

8. `oracle.jbo.server.ViewObjectImpl.closeFreedStatements()`:

This public API allows you to close the freed (released) statements associated with the view object QueryCollection (the prepared statements that were released back to the statement cache and is currently in an unused/unlocked status).

In case of *`findByKey`* operations, BC4J may use an internal view object. Therefore, to close freed statements associated with the `findByKey` call, you need to use `ViewObjectImpl.getByKeyFinderRS()`.

Note: Even though the Javadoc states that it is for internal use, this is a BC4J-approved public use of this method.

```
ViewRowSetImpl vrs = getByKeyFinderRS();
  if (vrs != null)
  {
    ((ViewObjectImpl)vrs.getViewObject()).closeFreedStatements();
  }
```

9. For application modules pooled by OA Framework, the application module release (with a state reset) takes care of closing the JDBC statements:

- Selecting the global **Home** link forces an application module release with a state reset.
- A JServ session timeout forces application module release.

10. For standalone application modules created with `oracle.apps.fnd.framework.OAApplicationModuleFactory.createRootApplicationModule` that are not pooled by OA Framework, to prevent a JDBC statement leak or OutOfMemoryError, perform one of the following:

- Call `ViewObjectImpl.clearCache()` (if caching is not required) or fetch all the rows to cause the cached statements to be released. Then at the end of the request, call `ViewObjectImpl.closeFreedStatements()` to close the released statements. Perform a reconnect as necessary when a SQLException is encountered or check the `Transaction.isConnected()` flag at the beginning of the request.

or

- Call `Transaction.disconnect(true)` (if caching is required) or `Transaction.disconnect()`/`Transaction.disconnect(false)` to release and close the statements at the end of each request. Then at the beginning of each request, call `Transaction.reconnect()`. Note, however, that if the application module is a static application module, this would require proper synchronization, and the synchronization cost could degrade performance and scalability. There is also more performance overhead in disconnecting (such as the need for database rollback and cleanup) and then reconnecting (minimal database session initialization, cursor and QueryCollection synch-up after reconnect).

Miscellaneous Implementation Techniques

Iterating View Object Rows

To iterate over view object rows without changing row state / currency, create a separate iterator as shown in the following example from the ToolBox

Tutorial (this code can be found in `oracle.apps.fnd.framework.toolbox.tutorial.server.PoSummaryAMImpl`).

```
public Boolean deletePurchaseOrder(String poHeaderId)
{
  boolean rowFound = false;
  RowSetIterator deleteIter = null;

  try
  {
    // First, we need to find the selected purchase order in our VO.
    // When we find it, we call remove( ) on the row which in turn
    // calls remove on the associated PurchaseOrderHeaderEOImpl object.
    int poToDelete = Integer.parseInt(poHeaderId);

    OAViewObject vo = getPoSummaryVO1();
    PoSummaryVORowImpl row = null;

    // This tells us the number of rows that have been fetched in the
    // row set, and will not pull additional rows in like some of the
    // other "get count" methods.
    int fetchedRowCount = vo.getFetchedRowCount();

    // We use a separate iterator -- even though we could step through the
    // rows without it -- because we don't want to affect row currency.
    // This deliberately illustrates the use of an iterator. There are also
    // convenience methods for getting rows that have certain values. See
    // the "approve" method below for an example.
    deleteIter = vo.createRowSetIterator("deleteIter");
    if (fetchedRowCount > 0)
    {
      deleteIter.setRangeStart(0);
      deleteIter.setRangeSize(fetchedRowCount);
      for (int i = 0; i < fetchedRowCount; i++)
      {
        row = (PoSummaryVORowImpl)deleteIter.getRowAtRangeIndex(i);

        // For performance reasons, we generate ViewRowImpls for all
        // View Objects. When we need to obtain an attribute value,
        // we use the named accessors instead of a generic String lookup.

        // BAD CODE EXAMPLE: Number primaryKey = (Number)row.getAttribute("HeaderId");
        // GOOD CODE EXAMPLE:
        Number primaryKey = row.getHeaderId();
        if (primaryKey.compareTo(poToDelete) == 0)
        {
          row.remove();
          rowFound = true;
          getTransaction().commit();
          break; // only one possible selected row in this case
        }
      }
    }
  }
  finally
  {
    // Always close your iterators even if there are errors. So, enclose
    // this in a finally block.
    deleteIter.closeRowSetIterator();
  }

  return (rowFound? Boolean.TRUE : Boolean.FALSE);
} // end deletePurchaseOrder()
```

To simply locate a row with a certain attribute value, you can also safely use the `OAViewObject getFirstFilteredRows()` methods as shown in the following example:

Note: There is a `getFilteredRows()` method that also returns an array of rows with matching attribute values.

```
public void approvePurchaseOrder()
{
  // To call a custom method on an Entity Object you should add a wrapper
  // in the VO's *RowImpl class (see PoSimpleSumaryVORowImpl).

  PoSummaryVOImpl vo = getPoSummaryVO1();
  Row row = vo.getFirstFilteredRow("SelectFlag", "Y");
  if (row != null)
  {
    row.approve();
  }
  else
  {
    // The user did not select a row
    throw new OAException("AK", "FWK_TBX_T_SELECT_FOR_APPROVE");
  }
```

```
  getTransaction().commit();
} // end approvePurchaseOrder()
```

Copying Rows

To copy rows from one view object to another, use the following code as an example:

```
Row originalRow = .....
Row newRow = viewObject.createRow();
newRow.setAttribute(....); //Set new row's primary key
newRow.setAttribute(....); //Set new row's primary key
newRow.setAttribute(....); //Set new row's primary key

// Get the attribute values from the originalRow and set them on the newRow
// using setAttribute.
...
```

Generic Copy Routine

For cases where you need to code a centralized, generic routine to copy row contents from one view object to another, the following example code shows you how to code such routine in a safe manner.

By "safe," we mean that you must make sure that your code observes the following guidelines as illustrated in the code example below:

- Copy the attribute values based on *attribute names* rather than index position. Also make sure that the attribute exists in the target view object.

  This way, the copy operation works safely even if the attribute orders differ, or even if customers extend your view object and change the attribute order or include more attributes.

- Bypass unknown internal attributes that may have been dynamically added by BC4J or the OA Framework.

  The following code example show how to do this by examining the attribute kind. If you still want to copy your own known dynamic attribute values, then you must explicitly copy those values using hardcoded attribute names.

```
/**
 * Copies row contents from a source view object to an empty destination view
 * object.
 *
 * Key Assumptions:
 *
 * 1. The destination view object is empty and you have prepared the destination
 *    view object for row insertion prior to calling this method as stated in
 *    "View Objects in Detail - Initialization Guidelines" developer guide.
 * 2. The destination view object is a distinct instance from the source view object.
 * 3. The destination view object will reuse the primary key values of the
 *    source view object -- if you do not want this behavior, use a modified
 *    routine that skips copying the primary key values.
 * 4. Row.setNewRowState(Row.STATUS_INITIALIZED) is not called on the copied
 *    rows -- if you want the copied rows to be in STATUS_INITIALIZED state,
 *    use a modified routine that allows you to set the copied row state to
 *    STATUS_INITIALIZED state.
 *
 * Note: For an entity-based view object, if you want to create a new row
 * that reuses existing entity instances, you can create a new row using
 * ViewObject.createRow() and then call ViewRowImpl.setEntities to set the entities
 * of the new row to reference existing entities.
 *
 * @param sourceVO a source view object that contains rows to be copied
 * @param destVO an empty destination view object that will accept the
 *        copied rows
 */
public static void copyRows(ViewObjectImpl sourceVO, ViewObjectImpl destVO)
{
  if (sourceVO == null || destVO == null)
  {
    return;
  }

  AttributeDef[] attrDefs = sourceVO.getAttributeDefs();
  int attrCount = (attrDefs == null)? 0 : attrDefs.length;
  if (attrCount == 0)
  {
    return;
  }

  // Create a row set iterator on the source view object to use for copy operation.
  RowSetIterator copyIter = sourceVO.findRowSetIterator("poCopyIter");
  if (copyIter == null)
  {
    copyIter = sourceVO.createRowSetIterator("poCopyIter");
  }
```

```
  // Indicates whether a copied row has been inserted into the destination
  // view object or not
  boolean rowInserted = false;

  while (copyIter.hasNext())
  {
    Row sourceRow = copyIter.next();

    // If a copied row was inserted into the destination view object in the
    // previous iteration cycle, then advance the row pointer so that the
    // new row gets inserted in the next row position.
    if (rowInserted)
    {
      destVO.next();
    }

    // Create a row for the destination view object to accept the copied
    // contents.
    Row destRow = destVO.createRow();

    for (int i = 0; i < attrCount; i++)
    {
      byte attrKind = attrDefs[i].getAttributeKind();

      // Bypass copying attributes internally created by BC4J or OA Framework.
      // If you want to copy your own known dynamic attribute values from the
      // source VO to the destination VO, then copy those attribute values
      // explicitly using hardcoded attribute names.
      if (!(attrKind == AttributeDef.ATTR_ASSOCIATED_ROW ||
            attrKind == AttributeDef.ATTR_ASSOCIATED_ROWITERATOR ||
            attrKind == AttributeDef.ATTR_DYNAMIC))
      {
        String attrName = attrDefs[i].getName();

        // Make sure that the attribute exists in the destination view object.
        if (destVO.lookupAttributeDef(attrName) != null)
        {
          Object attrVal = sourceRow.getAttribute(attrName);

          if (attrVal != null)
          {
            destRow.setAttribute(attrName, attrVal);
          }
        }
      }
    }

    // Insert the copied row into the destination view object.
    destVO.insertRow(destRow);
    rowInserted = true;
  }

  copyIter.closeRowSetIterator();
  destVO.reset();
}
```

Tip: To insert the copied rows at the end of the same view object instead of at the beginning, use the following code:

```
try
{
  // Set the row validation to false, if you don't want the current row
  // change to trigger row validation.
  vo.setRowValidation(false);

  // Save the original current row and range start.
  Row savedCurrentRow = vo.getCurrentRow();
  int savedRangeStart = vo.getRangeStart();

  // Copy the source row contents into new rows.
  Row[] sourceRows = vo.getFilteredRows("SelectFlag", "Y");
  if (sourceRows != null && sourceRows.length > 0)
  {
    int numRows = sourceRows.length;
    for (int i = 0; i < numRows; i++)
    {
      Row newRow = vo.createRow();

      // Get the attributes from the sourceRow and set them on the newRow
      // -- expect the primary key.  Use the primary key generated from
      // the database sequence in createRow() for the new row.
      newRow.setAttribute(...);
      ...
```

```
      // Insert the new row at the end of the VO.
      // These new rows may appear in the next table range.
      vo.last();
      vo.next();
      vo.insertRow(newRow);
      newRow.setNewRowState(Row.STATUS_INITIALIZED);
    }
  }

  // Restore original current row and range start.
  vo.setCurrentRow(savedCurentRow);
  vo.setRangeStart(savedRangeStart);
}
finally
{
  vo.setRowValidation(true);
}
```

View Object with Custom Data Source

To formulate a view object data from a custom data source, create a placeholder view object that is not based on any entity object with transient attributes and designate one or more attributes as primary key attributes. In your code, initialize your view object with a call to `setMaxFetchSize(0)` to suppress any efforts by BC4J to execute a query on your behalf and then insert the rows with custom data set on the view object attributes. See Poplist Custom Data Example for a short code example.

Updating "Expert Mode" Rows with Multiple Outer Joins

Scenario

In this scenario, an expert mode view object includes an outer join to two entity objects (EO1 and EO2). Values exist for EO1 but not for EO2, which has not yet been created. If the caller tries to set attribute values on EO2, BC4J assumes that this is a modify operation instead of a create, and throws an exception because a row for EO2 does not exist in the database.

Solution

When setting attribute values in EO2, you *must* first check to see if the entity object needs to be created and marked for insertion. You can then set the attribute value.

In your `OAViewRowImpl` override `setAttributeInternal()` to handle these outer joins as shown:

```
protected void setAttributeInternal(int index, Object val)
{
  // Check to see if the primary key is null.  If it is, you need to
  // create the entity instance in an "insertable" state.

  if (getForecastId() == null && ....)
  {
    // Get transaction and create the attribute list of values to be
    // passed to the create() method in the EO.
    DBTransaction transaction = getEntity(0).getDBTransaction();

    AttributeListImpl attributeList = new AttributeListImpl();
    attributeList.setAttribute("ForecastId", <some value>);
    ForecastEOImpl forecastEO = transaction.createEntityInstance(
      "<full_path_of_EO_definition_in_the_form - oracle.apps.....schema.server.ForecastEO>", attributeList);
    //Override the existing entity which will eventually be garbage collected.
    setEntity(0, forecastEO);
  }

  super.setAttributeInternal(index, val);
}
```

Dynamic View Object Definitions

Per the OA Framework Model Coding Standards, view objects should always be created declaratively whenever possible. If you must define a view object dynamically at runtime, see the detailed instructions in the `oracle.apps.fnd.framework.server.OAViewDef` Javadoc for creating a thread-specific view instance.

Setting View Object Where Clause

Upon calling the `ViewObject.setWhereClause(String)` method, be aware that the string supplied to the `setWhereClause` method should reference the proper column alias if the database column is aliased in the view object.

For instance, suppose your view object's query statement contains the following SQL in the View Object wizard - Query section:

```
SELECT EmployeeEO.EMPLOYEE_ID,
  EmployeeEO.FIRST_NAME,
  EmployeeEO.LAST_NAME,
  EmployeeEO.FULL_NAME AS EMPLOYEE_NAME,
```

```
  EmployeeEO.EMAIL_ADDRESS AS EMPLOYEE_EMAIL,
  EmployeeEO.MANAGER_ID,
  EmployeeEO.POSITION_CODE,
  EmployeeEO.SALARY,
  EmployeeEO.START_DATE,
  EmployeeEO.END_DATE,
  FROM FWK_TBX_EMPLOYEES EmployeeEO
```

To set a WHERE clause for the employee's full name, use `setWhereClause("EMPLOYEE_NAME = :1")` instead of `setWhereClause("FULL_NAME = :1")` to avoid SQL exceptions.

Master-Detail View Object Relationships

This section discusses Data Model Master-Detail (DM MD) and Row-Level Master-Detail (RL MD) master-detail relationship styles that are supported by BC4J Framework. Also included in this section is information about using Association/View-Link Accessors with both of these styles of master-detail relationships.

Data Model Master-Detail (DM MD)

The Data Model Master-Detail style is where the user adds both the master VO and the detail VO instances into the Application Module's Data Model. The two VO instances are related by a view link. The master and detail VOs are fixed with the view link managing the data flow events between the two VOs.

In DM MD, the detail VO is stable. The data changes but the VO remains the same object. The data rows in the detail VO are refreshed as the current master row changes. Therefore, UI objects can bind themselves to the stable VO and respond to the `refreshRange` event, which is triggered when the current master row changes.

Note: DM MD is appropriate for a UI form where the master is shown in text boxes and the detail is shown in a grid. As the master VO is navigated, the grid will be refreshed to show data related to the current master row.

In DM MD, the detail VO can have multiple master VOs. For example, the EmpView VO is a detail to both the DeptView VO and the ProjectView VO. The current master row for DeptView shows Manufacturing and the current master row for ProjectView shows Project 100. In this case, the EmpView will show only those employees that belong to the Manufacturing Dept and are working on Project 100.

Note: In RL MD, the detail VO can only be associated to one master VO.

Row-Level Master-Detail (RL MD)

The Row-Level Master-Detail style is where the user takes a row from the master VO and calls an association/view-link accessor method (`get<association or view-link end name>()`) on it. For example, suppose there is a view link between DeptView and EmpView. The view-link accessor method on DeptView would be `getEmpView()`.

Unlike the Data Model MD, RL MD does *not* require an Application Module or Data Model context because the entire master context provides the master row. RL MD allows the user to make a random number of association/view-link accessor calls with each call returning a RowSet that contains a set of detail rows related to the master row.

Note: In RL MD, the data rows are stable. The detail data rows do not change as long as the association/view-link attribute value(s) in the master row remain unchanged. Because the detail row set is attached to the master row and unaffected by a change, RL MD is appropriate for a UI object such as the tree control.

In RL MD, the detail VO can only be associated to one master VO. The result of calling an association/view-link accessor is a RowSet whose data meets the criteria set by the current master row. In other words the resulting RowSet belongs exclusively to just one view row instance.

Association/View-Link Accessors

Important: In all cases, the row set returning from a call to an accessor (RL MD) originates from an internal view object, which is created in the root application model.

Suppose, a user builds a DM MD and then retrieves a row from the master VO of this data model. The user calls an association/view-link accessor from the master row and a row set is returned (RL MD). The user expects the row set to have originated from the detail VO in the data model but this is *not* the case. The user made a call to an accessor therefore, the resulting row set originated from an internal object.

The BC4J Framework does *not* use any user created view objects, including the ones in the data model, for accessor results (row sets). This is because the the VO used for the association/view-link accessor *must* be stable and unaffected by a user's unrelated changes to the detail view object of the data model.

Suppose the detail VO in the data model *is* used for accessor result row sets. The results will be affected by changes such as:

- The user takes the detail VO in the data model and sets its `WHERE` clause with parameters. (This is acceptable as long as the user supplies parameter values). However, if the detail VO was used for accessor result row sets, an error will occur because the `WHERE` clause parameter values were supplied *only* for the default row set of the detail VO and not the accessor result row set.
- The user adds another master VO to the detail VO in the data model causing the semantics of the accessor to be changed. For example, instead of an accessor returning details for the original master VO, it may start returning details for both the original and the new master VO.
- The user removes the detail VO or the containing Application Module causing *all* detail rows to become invalid.

Note: BC4J Framework *must* be able to differentiate between DM and RL detail VOs for certain operations. For example, when you create a new row in a detail VO, BC4J automatically populates the association/view-link attributes from the corresponding values of the master. The DM MD gets these values from

the current row of the master VO. The RL MD gets the values from the owning master row.

Security

Overview

This document centralizes the resources relevant to OA Framework web application security for authentication, authorization, data security and validation.

Contents

- Menus
- Page Security
- Component-Based Function Security
- Accelerated Validation
- Cross-Site Scripting
- Frame Busting
- Server Security
- Data Security
- Oracle Application Object Library Security
- Secure Configuration
- AntiSamy Check
- File Streaming
- Enabling Virus Scanning

Accelerated Validation

As of Oracle E-Business Suite Release 12, OA Framework-based pages in Oracle E-Business Suite run in accelerated validation mode. Accelerated validation provides a set of event points and checks that are specifically designed to prevent the tampering of parameters and URLs used by a page. Programmatic validation may be skipped if OA Framework can automatically ensure the validity of the URL parameters as coming from a trusted source. This functionality is enabled by default in Release 12 and is the recommended configuration, providing additional security to a page.

Your OA Framework-based pages can take advantage of the new accelerated validation as long as they meet existing coding standards. Overall, the validation is a transparent process, except for the following exception cases where you must follow the corresponding instructions to ensure correct page behavior:

- You have bookmarkable pages (any page that can be accessed *across* user sessions), where an OA Framework URL generated in one session must be accessible in another.
- You have custom JavaScript on the browser that reads from or writes to hidden form fields, picklists, radio groups, lists or shuttles.
- You programmatically configure a URL in the middle tier (using java code).

Contents

- Enabling Accelerated Validation
- Bookmarkable Pages
  - Parameter Validation in Bookmarkable Pages
  - Forwards or Redirects in Bookmarkable Pages
- Using an Untrusted URL within an OA Framework Page
- Handling JavaScript
- Preparing URLs Embedded in JavaScript/Text
- Skipping Programmatic Validation
- Fixed Key Support
- Related Information

Enabling Accelerated Validation

Accelerated validation is enabled by three Validation-related profile options:

- FND Validation Level / FND_VALIDATION_LEVEL
- FND Function Validation Level / FND_FUNCTION_VALIDATION_LEVEL
- Framework Validation Level / FRAMEWORK_VALIDATION_LEVEL

You may also override these profiles using their corresponding Java Virtual Machine (JVM) settings.

Refer to "Secure Configuration of Oracle E-Business Suite Profiles", My Oracle Support (formerly Oracle *MetaLink*) Knowledge Document 946372.1, for details on the recommended settings for these profiles.

Important: The Oracle E-Business Suite Applications Technology (TXK) Release 12.2.2 removes the ability to turn these profiles off. Oracle E-Business Suite Release 12.2.2 always runs as if these profiles are set to **ERROR**.

Bookmarkable Pages

To ensure valid access to a bookmarkable page, follow these instructions:

Step 1: Identify your "bookmarkable" page candidates. Generally, they fall into the following categories:

- (Required) any page accessed from a Workflow Notification (or email) link. Note that any page that could be accessed in this way prior to 12 *will not work correctly* unless you complete the following steps.
- (Required) any logical application entry points (like a "Home" page).
- (Required) pages to which you link from some non-OA Framework technology (JTT, Forms, and RF excepted).
- (Required) pages whose URL will be manipulated with JavaScript.
- (Optional) any page you think customers will expect to bookmark.

Step 2: For a bookmarkable page, select the **pageLayout** region in the Oracle JDeveloper Structure pane to open the Property Inspector. Set the `securityMode` attribute to **selfSecured** (the default value is **standard**, which means that the URL is prepared for accelerated validation). When you set the value to **selfSecured**, OA Framework lets you bypass accelerated validation to inspect the parameter values to determine whether page access should be granted.

Note: For shared regions that might be used in a bookmarkable page, you should also set the `securityMode` parameter to **selfSecured**.

Step 3: Any pages that are marked as **selfSecured** must have an associated function. This can be achieved either by setting the `Function Name` property on the **pageLayout** region, or by binding the `Rendered` property of the page to a function (see Dynamic User Interface: Component-Based Function Security for implementation instructions)

Note: If you set a function, OA Framework always checks it before rendering the page, regardless of whether the page is **selfSecured** or not.

Step 4: On any controllers in this page that call `getParameter` (or in a controller associated with a parent region that is marked as **selfSecured**), implement the `validateParameters` method as shown below.

If your **pageLayout** region's `securityMode` parameter is **standard**, and validation of the page's URL fails, processing stops. If validation succeeds, OA Framework enters the `processRequest` processing phase so it can render the page.

If the pageLayout region's `securityMode` parameter is **selfSecured**, OA Framework tests the function associated with the page, and then calls the `validateParameters` method in every page controller whose region is **selfSecured** before calling `processRequest`. Within this method, you must write code to inspect the relevant parameter values and indicate whether or not they are valid by including them in an array list that you return to OA Framework, which maintains a running list of valid parameters for the page. Whenever you call `pageContext.getParameter`, OA Framework checks the validated parameters list to see if it includes the given parameter. If so, it returns the parameter value to you. If not, it throws an exception.

For example:

```
public String[] validateParameters(OAPageContext pageContext,
                                   Vector alreadyValidatedParameters)
{
   // Get the parameter(s) that you're interested in.  Note that this call
   // to getParameter() does NOT raise exceptions if you access unvalidated
   // parameters.

   String paramToValidate = pageContext.getParameter("<ParameterToValidate">);

   // Add code here to do the parameter validation. This step may be
   // skipped if the parameter is actually validated later in the
   // processRequest call.
   //
   // Add the validated parameter names to a String array, and return.  The
   // OA Framework adds this list of validated parameters to its internal list.

   String[] list = {.....};
   return list;

} // end validateParameters()
```

Step 5 (optional): If you need to access your page outside an OA Framework runtime context (from a test JSP, for example), but do not wish to or cannot mark the target page as **selfSecured**, you need to generate your URL using the method `URLMgr.processOutgoingUrl(String url, WebAppsContext wctx)`. The private `oracle.apps.fnd.framework.CreateIcxSession` class includes a new convenience method called `getWebAppsContext` to obtain the WebAppsContext instance associated with the page. If you fail to do this, you will not be able to access your page from this launching context.

The following excerpts from a test JSP illustrate how to use these methods:

```
<%@ page
```

```
   language = "java"
   errorPage = "OAErrorPage.jsp"
   contentType = "text/html"
   import = "java.io.File"
   import = "oracle.apps.fnd.framework.webui.OAJSPHelper"
   import = "oracle.apps.fnd.framework.webui.URLMgr"
   import = "oracle.apps.fnd.common.WebAppsContext"
%>

...

<jsp:useBean
 id = "sessionBean"
 class = "oracle.apps.fnd.framework.CreateIcxSession"
 scope = "request">
</jsp:useBean>

...

WebAppsContext wctx = sessionBean.getWebAppsContext();

...

<a href="<%=URLMgr.processOutgoingURL("OA.jsp?OAFunc=...", wctx) %>">Lesson 1: Hello, World!</a><br>

...
```

Note: You can still run your XML page definitions directly from JDeveloper (by selecting the page in the JDeveloper Structure pane, right-clicking and selecting **Run *<page>*** or **Debug *<page>***) without generating your page URL with the `processOutgoingUrl` method. This is necessary only if you are using a test JSP to run your pages.

Usage Notes

- Once a URL is constructed with the `processOutgoingUrl` method, it should not be manipulated as this will lead to errors (note that it's okay to add parameters when doing a JSP forward or a redirect, and calling `pageContext.putParameter` is fine also).
- Forms, JTT, and RF.jsp (Java and PL/SQL versions) can call non-bookmarkable OA Framework pages with the accelerated validation mode enabled.
- As of Release 12.2.2, any page that is not bookmarkable, because it requires its AM state to be retained while navigating, is no longer accessible from the global header Favorites pull-down link. You may only add bookmarkable pages as favorites from the Favorites pull-down link.

Parameter Validation in Bookmarkable Pages

The steps above describe how to ensure valid access to a bookmarkable page, but you do not have to validate a bookmarkable page's parameters explicity in the `ValidateParameters()` routine itself. Architecturally, it is often better to validate the parameters in the Model code, that is, in the AM or VO for queries, but you should still indicate where the validation takes place in the `ValidateParameters()` routine. For every parameter that is asserted valid, either:

- Explicitly validate the parameter in `ValidateParameters()`.
- Explain, via a comment, where this page leverages validation.
- Explain, via a comment, why a lack of validation would not cause a security issue.

Use the following guidelines and testing considerations to ensure validation:

- **Validate Data Security** - Generally, for a parameter with data security, validate with an explicit query to confirm the user has access. You should have an explicit test to validate that users cannot view parameter values outside of their data security context.
- **Validate Data Types** - Validate data types with a list of allowed characters. If the parameter value is a generic numeric ID, you should validate that it is numeric. If the parameter value is a character ID, validate that it only has characters you expect. You need an explicit test to accept only allowed characters.
- **Validate IDs** - If you have an ID that is already used in a query and it is implicitly validated via the query, be sure it is done via bind variables (parameterized queries) and it enforces any data security. You need an explicit test to accept only valid values.

For example:

```
   ...
   String postingId =  pageContext.getParameter(IRC_PARAM_SELECTED_POSTING);
   String vacancyId =  pageContext.getParameter(IRC_PARAM_SELECTED_VACANCY);


   String[] list ={ };
```

```
  if (action != null )
  {
   if( action.equals(IRC_APPLY_FOR_POSTING) || action.equals(IRC_REFER_POSTING) ||
      action.equals(IRC_VIEW_POSTING) || action.equals(IRC_ADD_POSTING_TO_BASKET))
   {
    /*
        postingId - additional data security for postingId is in the AM method queryUserProjects()
        vacancyId - any vacancyId in the system is allowed for any action in the page - there is
           no additional data security.  The values are bound in to the page.
    */

    if (postingId != null &&  toNumber(postingId) != null &&
       vacancyId !=  null && toNumber(vacancyId))
       list = new String[]
          { IRC_PARAM_ACTION, IRC_PARAM_SELECTED_VACANCY, IRC_PARAM_SELECTED_POSTING };
   }
  }
```

Forwards or Redirects in Bookmarkable Pages

Be especially careful of redirecting URLs (client roundtrip) or forwarding (server-side chaining) from a bookmarkable page. Many mechanisms of forwarding or redirecting within OA Framework assume that the URL is trusted and OA Framework will sign the URL. This effectively turns every page that can be forwarded or redirected to into its own bookmarkable page. If that page or its parameters on the URL are unvalidated, a trust violation may occur, and provide a means to bypass accelerated validation security.

If you require a redirect or forward from a bookmarkable page, understand that the bookmarkable page must provide all validation. In particular:

- Only redirect or forward to a explicit list of pages. Testing that the user has access to the page being forwarded to is not sufficient.
- Always validate parameters being used in the redirect or forward.
- Build up the forward or redirect URL from scratch, using the validated page and parameters from the previous two bullet points.

Any parameters not explicitly validated will still raise errors if you use them in subsequent pages.

You can legitimately redirect to another bookmarkable page without validating if you mark that URL as untrusted. See Using an Untrusted URL within an OA Framework Page below for details on how to redirect a URL containing untrusted information.

Using an Untrusted URL within an OA Framework Page

Generally, if you put a URL reference into a page, it will be treated as a trusted URL. However, if you know that the URL is from an untrusted source (for example, it was input by a user) or if you put a URL reference into a non-OA Framework page, you should always validate that URL target. You can use either of the two options below to force programmatic validation for a target page:

Option 1:

Set an attribute on the web bean component that has the `Destination` attribute. You should include code similar to the following in the `processRequest` method of the controller:

```
OAStaticStyledTextBean item =
  (OAStaticStyledTextBean)createWebBean(pageContext,
    STATIC_STYLED_TEXT_BEAN);
// determine if its destination is untrusted.
boolean unTrustedURL= ...;

// set the destination as you would normally do
item.setDestination("{$"+viewAttributeName+"}");
// now set the UNTRUSTED_DESTINATION_URL_ATTR attribute to the
// constant value of UNTRUSTED_URL.
// Note that both these constants are from
// oracle.apps.fnd.framework.webui.OAWebBeanConstants
// and all controllers implement this class by default.
if(unTrustedURL)
{
  item.setAttributeValue(
    UNTRUSTED_DESTINATION_URL_ATTR, UNTRUSTED_URL);
}
```

Option 2:

To disable accelerated validation for a URL, in the `processRequest` method of the controller for the page that generates the URL, call the `createURL` API on `oracle.apps.fnd.framework.webui.OAUrl` with the following signature:

```
public String createURL(RenderingContext context,
  boolean trusted)
```

The code example below shows how to use this API:

```
String redirectURL = new String ("someRedirectURL");
OAUrl url = new OAUrl(redirectURL);
redirectURL = new StringBuffer(url.createURL(
  pageContext, false));
// false indicates the URL is coming from an untrusted
// source.
```

Handling JavaScript

Warning: Although the following section describes how to handle existing JavaScript, no new code should contain any JavaScript as per the OA Framework Controller Coding Standards and the OA Framework View Coding Standards.

As noted above, if you have JavaScript that reads from or writes to hidden fields, picklists, radio groups, lists or shuttles, you must follow these specific instructions to ensure proper page behavior in the context of accelerated validation. For simplicity, since access to the parameters being passed to and from the above styles must be validated, they are referred to as validation-required input fields within the rest of this document. To take advantage of accelerated validation, you can use the bound value `oracle.apps.fnd.framework.webui.OABoundValueMAC` while generating the JavaScript to ensure that the values for these input fields are prepared for validation.

Following is an example of how you would use this bound value with a hidden field. Suppose you have a JavaScript function as a link on your page and when you click on the link, the JavaScript function populates a hidden field on the page with a specific value.

Lets say the JavaScript link looks like:

```
putValue('myField1': 'myValue1');
```

The `putValue` JavaScript function does some checking, and ultimately populates the hidden field `myField1` with the value `myValue1`. Since `myField1` is a form value, you should make sure that the value you populate in it (`myValue1`) is prepared for accelerated validation.

You can do this using the following steps:

Step 1: Create the text web bean and set its text to your bound value. Let us call this `<Prod>BoundValueTextAttr` . Make sure you pass in the `formName` of your enclosing form to this bound value.

Refer to Bound Values for more information.

Step 2: Set this bound value on the text attribute of your raw text web bean as follows:

```
textBean.setAttributeValue (TEXT_ATTR,
  new <Prod>BoundValueTextAttr (formName));
```

Step 3: Define constants for the static parts of the strings in the `<Prod>BoundValueTextAttr` bound value:

```
private static final String TEXT_BEGIN = "putValue('myField1':"
private static final STRING TEXT_END = ");";
```

Step 4: Implement the `getValue` method of `<Prod>BoundValueTextAttr` as follows:

```
public Object getValue(RenderingContext context)


{
  // Get a new bound value key for the parameter that needs
  // to be prepared for accelerated validation. The inputName
  // is the name of the parameter or formValue that needs
  // this key.

  OABoundValueMAC boundValueMAC = new OABoundValueMAC
    ("DefaultForName", "myField1", "myValue1");

  // Now create a concat bound value that concatenates all
  // the above strings. See the javadoc corresponding to
  // these bound values.

  ConcatBoundValue concatBoundValue = new ConcatBoundValue (
          new BoundValue[] {
            new FixedBoundValue (TEXT_BEGIN);
            boundValueMAC,
            new FixedBoundValue (TEXT_END);
          });
  return concatBoundValue.getValue(context);
}
```

If for any reason, you cannot prepare the values for the validation-required input fields for accelerated validation, you must perform the following steps instead:

Step 1: This step is valid only if your validation-required input field is a hidden field. Proceed to Step 2 otherwise.

Identify how the hidden field is accessed or manipulated in your JavaScript function. If the JavaScript is used to simply clear all fields on the page including the hidden fields, then set the `scriptClearAllowed` property to **true** for the hidden fields using the programmatic method on the `oracle.apps.fnd.framework.webui.beans.form.OAFormValueBean`:

```
formValueBean.setScriptClearAllowed(true);
```

If your hidden field's `scriptClearAllowed` is set to **true**, then OA Framework skips its accelerated validation when its value is posted as null. OA Framework continues accelerated validation if the posted value is non-null.

If the JavaScript manipulates the hidden field in any other way, then follow the steps below.

Step 2: Select the validation-required input field in the Oracle JDeveloper Structure pane to open the Property Inspector. Set its `securityMode` property to **selfSecured** (the default value is **standard**, which means that the field's value is prepared for accelerated validation). When you set the value to **selfSecured**, OA Framework lets you bypass accelerated validation to inspect its parameter value to determine whether the parameter is valid or not.

If you would like to set the `securityMode` attribute programmatically, then you can do so using:

```
bean.setAttributeValue (SECURITY_MODE, SELF_SECURED);
```

Step 3: On any controllers in this page that call `getParameter` to retrieve the value of this validation-required input field, you must implement the `validateFormParameters` method as shown below.

If the validation-required input field's `securityMode` parameter is **standard**, and accelerated validation results in validation failure, processing stops. If the accelerated validation succeeds, OA Framework enters the `processFormData` processing phase so that it can post the parameters to the model.

If the validation-required input field's `securityMode` parameter is **selfSecured**, OA Framework calls the `validateFormParameters` method in every page controller before calling `processFormData`. Within this method, you must write code to inspect the relevant parameter values for the fields that you marked as **selfSecured** and indicate whether or not they are valid by including them in an array list that you return to OA Framework.

OA Framework maintains a running list of valid parameters for the page. Whenever you call `pageContext.getParameter`, OA Framework checks the validated parameters list to see if it includes the given parameter. If so, it returns the parameter value to you. If not, it throws an exception.

Read Step 4 below if your validation-required input field is under a table, HGrid or Gantt region.

For example:

```
public String[] validateFormParameters
        (OAPageContext pageContext,
   ParametersList alreadyValidatedFormParameters)
{
   // Get the parameter(s) that you're interested in. Note
        // that this call to getParameter() does NOT raise
        // exceptions if you access unvalidated parameters.
        String paramToValidate = pageContext.getParameter
                ("<ParameterToValidate">);

        // Add code here to do the parameter validation for
        // the parameters corresponding to the hidden fields
        // that you marked as selfSecured.
        // Read the next step if your hidden field is a child of a
        // table.
   ...

   // Add the validated parameter names to a String array, and
        // return. The OA Framework adds this list of validated parameters
        // to its internal list.
        String[] list = {.....};
   return list;

} // end validateFormParameters()
```

If you have similar validation logic in your model (view object), you can defer validation in the `validateFormParameters` method, and reuse the validation logic in the model. Read Step 4 below for instructions on how to do that.

Step 4 : If your validation-required input field is under a table, HGrid or gantt region, then you should follow the additional steps to validate its value. You should validate the fields on a row by row basis in this case.

- Step 4.1: Defer the validation of your validation-required input field to the model (on the corresponding `rowImpl` class), and return the value as valid in your `validateFormParameters` method. You should add a comment in the `validateFormParameters` method to indicate where the validate logic resides. OA Framework will post this value to the model in the `processFormData` stage. You should then validate its value in the model.

OA Framework auto-generates the name of all fields for every row. You should hence use the `getTableParameterName(String tableId, String colID)` on `oracle.apps.fnd.framework.webui.OAPageContext` to get the actual parameter name.

For example:

```
public String[] validateFormParameters(OAPageContext pageContext,
   ParametersList alreadyValidatedFormParameters)
{
   // Get the parameter(s) that you're interested in. Note
        // that this call to getParameter() does NOT raise
        // exceptions if you access unvalidated parameters.
        String tableChildParamToValidate = pageContext
                .getTableParameterName(tableId, columnId);
        // do not add any code to validate the
        // tableChildParamToValidate value, defer it to the
        // model. Add the validated parameter names to a String
        // array, and return. The OA Framework adds this list
        // of validated parameters to its internal list.

        // Add a comment: The validation of tableChildParamToValidate
        // resides in oracle.apps...VORowImpl.class in the method
        // <the method that holds the validation logic>
        String[] list = {.....};
   return list;

} // end validateFormParameters()
```

- Step 4.2: Validate the values for your fields in the model by including your validation logic in the `setAttributeValue` of its corresponding view attribute in the view object's `rowImpl`. If you cannot use the `setAttributeValue` method (for example, if you need to cross validate multiple fields), then you can move your validation logic to the `validate` method (if your VO is based on a Java EO) or a PL/SQL function (if your VO is based oon a PL/SQL EO).

Usage Notes

- Any code that you add to the `validateFormParameters()` method is subjected to auditing by the OA Framework Compliance Working Group.

Preparing URLs Embedded in JavaScript/Text

Warning: Although the following section describes how to handle URLs embedded in JavaScript, no new code should contain any JavaScript as per the OA Framework Controller Coding Standards and the OA Framework View Coding Standards.

When a URL is generated through the `OAUrl.createURL` API, the URL is automatically prepared for accelerated validation. However, this happens only for URLs that are not embedded in JavaScript. To prepare URLs embedded in JavaScript or text (such as that emitted by a RawTextBean), a new convenience class called `oracle.apps.fnd.framework.webui.OABoundValueEmbedURL` is available.

To take advantage of this API, set it as the BoundValue on the appropriate web bean. If the web bean does not support setting a BoundValue, use the `setAttributeValue` API that is available on the web bean.

Attention: Do not call the `getValue` method on this web bean at the time of setting, as the URL is supposed to be created at render time.

The following code example illustrates the usage of this class:

```
OAImageBean imageBean = (OAImageBean)
  (pageContext.getWebBeanFactory().createWebBean(
  pageContext,OAWebBeanConstants.IMAGE_BEAN));
StringBuffer redirectUrl = new StringBuffer(
  OAWebBeanConstants.APPS_HTML_DIRECTORY);
redirectUrl.append(OAWebBeanConstants.APPLICATION_JSP);
redirectUrl.append("?akRegionCode=HR_COMP_INFO_SS&
  akRegionApplicationId=800&pCompetenceId=
  {!CompetenceId}");
imageBean.setAttributeValue(
  oracle.cabo.ui.UIConstants.ON_CLICK_ATTR,
  new OABoundValueEmbedURL(imageBean,"showCompInfo('",
  redirectUrl.toString(), "');return false;");
```

Skipping Programmatic Validation

OA Framework provides a new API you can use to verify if a parameter can be trusted. If the parameter has not been tampered with, then in your controller logic, you can skip the programmatic validation on the inbound request for better performance.

To use this API, call `pageContextImpl.isValidatedParameter` for the appropriate parameter. The following example illustrates how to verify the integrity of a parameter named OrderID:

```
if(!pageContextImpl.isValidatedParameter("OrderID"))
{
 // Perform your validation here.  You should only perform
 // this validation if the parameter cannot be trusted, that is,
 // this API returns a value of false.
}
```

The `isValidatedParameter` method returns **true** if OA Framework can provide automatic positive validation of the parameter's integrity. The method returns **false** if OA Framework cannot provide automatic positive validation of the parameter's integrity or if the parameter's value is null.

Fixed Key Support

When you take advantage of accelerated validation, each user gets a unique encryption key for their session. For tools such as Load Runner, where content generated for one user needs to be replayed for another user, the unique encryption key poses a problem because the generated URL changes each time a page is accessed. OA Framework provides fixed key support to work around this issue by allowing certain users to have a fixed encryption key for all their user sessions.

To enable fixed key support, set the profiles FND: Fixed Key Enabled and FND: Fixed Key. The profile FND: Fixed Key Enabled allows you enable the use of a fixed encryption key for accelerated validation for a user. If you enable this profile, you can then use the FND: Fixed Key profile to specify what that fixed key is for the user.

Load Runner users should set the FND: Fixed Key Enabled profile to **Y** at their user level and also specify a value for their FND: Fixed Key profile.

Usage Notes

1. Set the fixed key profiles so that fixed key support is enabled for the guest user "GUEST".
2. Successful fixed key support assumes that URLs remain the same across sessions. This includes having a consistent transactionID parameter in the URL. Using a transactionID generated from an earlier run of Load Runner should not affect results.
3. You should always encrypt URL parameters using OA Framework-provided mechanisms, such as token substitution and `OAPageContext.encrypt`. This ensures that the fixed encryption key is used, if enabled. If you encrypt a parameter by calling `OAPageContext.encrypt` and then pass in a different key, the URL constructed will not be consistent and as a result, the URL integrity will fail, causing the Load Runner usage to fail for this page.

Related Information

- Javadoc
  - `oracle.apps.fnd.framework.webui.OAControllerImpl`
  - `oracle.apps.fnd.framework.webui.OABoundValueMAC`
  - `oracle.apps.fnd.framework.CreateIcxSession`
  - `oracle.apps.fnd.framework.webui.OAUrl`
  - `oracle.apps.fnd.framework.webui.OABoundValueEmbedURL`
- OA Framework ToolBox Tutorial / Sample Library
  - test_fwklabsolutions.jsp
  - test_fwktutorial.jsp

Cross-Site Scripting

If you develop code that fails to perform adequate input validation, there is always the risk that an attacker may be able to inject dynamic scripting content such as JavaScript/vbscript, into Oracle E-Business Suite, and run the script(s) in the context of a legitimate Oracle E-Business Suite user. This is referred to as cross-site scripting. Oracle Framework provides a feature called Cross-Site Script scanning (XSS scanning), that provides added security to mitigate cross-site scripting attacks.

When requests are made to get a value or values for a parameter by calling OA Framework methods, the XSS scanning feature performs an automatic scan of the value or values for scripting content. If scripting content is found, an exception is thrown, otherwise the value or values are returned.

You can also programmatically specify that certain parameters not be scanned for cross-site scripting, such as when you already perform proper validation for a parameter value or when validation is not necessary.

Content

- Enabling XSS Scanning
- Disabling XSS Scanning on Specific Parameter Values

Enabling XSS Scanning

To enable XSS scanning, set the profile option, Restrict text input (FND_RESTRICT_INPUT) to **Yes** at the Site level. When XSS scanning is turned on, all parameter values are scanned for scripting content, unless you programmatically disable scanning for individual parameter values.

Disabling XSS Scanning on Specific Parameter Values

There are situations when you may want to disable XSS scanning on a specific parameter value:

- It does not matter whether the parameter value contains scripts, as the developer already performs the necessary escaping of the value when outputting it to the browser.
- The parameter value is never sent back to the browser, as in the case of a password.
- The developer already validates the parameter value in his/her program logic, and guarantees that the value cannot contain scripts. For example, the parameter value may be generated on the server originally and contains a number and its integrity has been validated by OA Framework to guarantee that the value has not been changed in the request.

Unless you are absolutely sure, however, you should let OA Framework perform XSS scanning for a parameter value.

To disable XSS scanning for a specific parameter value on a page, you must implement a validateParametersXSS method, on either of the following:

- Any controllers in this page that call `getParameter` or `getParameterValues` or `getDecryptedParameter`.
- A controller associated with your pageLayout region that validates all your parameter values (if you want to consolidate the scanning in one place).

OA Framework calls the `validateParametersXSS` method on every region controller in your page before calling the `processRequest` or `processFormRequest` method. Within the `validateParametersXSS` method, you must write code to indicate which parameters' values should not be scanned, by including them in an array list that you return to OA Framework. OA Framework then maintains a running list of not-to-be-scanned parameters for the page.

Whenever you call the `pageContext.getParameter`, `pageContext.getDecryptedParameter`, or `pageContext.getParameterValues` method, OA Framework checks the not-to-be-scanned parameters list to see if it includes the given parameter. If the list includes the given parameter, OA Framework returns the parameter value for you, otherwise, OA Framework runs the XSS scan on the parameter value(s). After the XSS scan, if it finds scripts in the value(s), OA Framework throws an exception, otherwise, it returns the parameter value(s) to you.

Example `validateParametersXSS` method:

```
public String[] validateParametersXSS(OAPageContext pageContext,  Vector alreadyValidatedParameters)
{
  // Get the parameter(s) that you're interested in.  Note that this call
  // to getParameter() does NOT raise exceptions if you access unvalidated
  // parameters.

  String paramToValidate = pageContext.getParameter("<ParameterToValidate">);

  // Add code here to do the parameter validation. This step may be
  // skipped if the parameter value is encoded later, or if
  // the parameter value will not be sent back to a browser.

  // Add the parameter names to a String array, and return.
  // OA Framework adds this list of parameters to its internal list.

  String[] list = {.....};
  return list;

} // end validateParametersXSS()
```

Frame Busting

Cross-site Framing is an attack on a client that is related to Cross-site Scripting. In a Cross-Site Framing attack, the attacker may exploit a browser flaw to inject HTML code from a third-party domain into an HTML frame. This enables the attacker to employ clickjacking to maliciously trick a user into clicking on something other than what he or she perceives to be legitimate, to reveal and steal confidential data.

OA Framework content is restricted from being embedded inside a web page of a different "domain". Here, "domain" refers to the domain name on which the site is hosted. This frame busting feature restricts OA Framework-based pages to be embedded only in web pages within the same web domain name of the application. If you plan to frame an OA Framework-based page from a different domain, refer to the warning note provided for the "FND: Disable Frame Busting" profile option.

Server Security

For information on administering server security, refer to the "Fundamental Oracle E-Business Suite Setup Tasks" chapter of the *Oracle E-Business Suite Setup Guide, Release 12.2*.

Data Security

For information on administering data security, refer to the "Access Control with Oracle User Management" and "Oracle User Management Setup and Administration" chapters of the *Oracle E-Business Suite Security Guide, Release 12.2*.

Oracle Application Object Library Security

For information on Oracle E-Business Suite security, refer to the "Oracle Application Object Library Security" chapter of the *Oracle E-Business Suite Security Guide, Release 12.2*.

Secure Configuration

For configuration guidance on securing Oracle E-Business Suite, refer to the appropriate chapters in Part 2, Secure Configuration, of *Oracle E-Business Suite Security Guide*.

AntiSamy Check

For more information about the AntiSamy check performed when you attach or upload a file or create content in a message rich text editor, refer to *Security Configuration Mechanism in the Attachments Feature in Oracle E-Business Suite*, My Oracle Support (formerly Oracle *MetaLink*) Knowledge Document 1357849.1.

File Streaming

OA Framework applications may need to stream a file, such as a PDF file or other document, from the server to the web browser. AOL/J provides a mechanism using a JSP and Java API to stream such files to the client. Refer to the "File Streaming Using AOL/J" topic in the *Oracle E-Business Suite Developer's Guide* for more information.

Virus Scanning

You can enable virus scanning on a file-type attachment before uploading the file.

Oracle E-Business Suite Java Caching Framework Developer's Guide, Release 12

Welcome to the Oracle E-Business Suite Java Caching Framework Developer's Guide. This guide is organized into the following sections:

- Introduction
  - Concepts
  - Architecture
  - Terminology
  - Features
  - Benefits
- Using the Caching Framework
  - Loader Class Concepts
  - Identifying What Data to Cache
  - Setting Up the Environment (for OA Developers)
  - Implementing a CacheLoader Class
  - Registering a Component Cache
  - Business API Example
  - Cautions
  - Caching Framework Seed Data
- Caching Framework Features
  - Distributed Cache
  - Cache Event Handling
  - DB Invalidation for Caching Framework
- Caching Guidelines
  - Choosing the Data to Cache
  - Naming the Component Cache
  - Ensuring Data Consistency
  - Distributed Cache Guidelines
  - Database Invalidation Guidelines
- Cache Administration
  - Cache Deployment
- APIs
  - CacheManager APIs
- Error Messages
- Testing and Troubleshooting
  - Testing the Component Cache
  - Testing Database Invalidation
- Caching Framework Seed Data
  - Terms
  - Standard WHO Columns
  - How to Use Seed Data
  - Registering Component Cache Seed Data
  - Cache Filter Seed Data
  - System Policy Seed Data
- Known Issues
- Frequently Asked Questions

Introduction

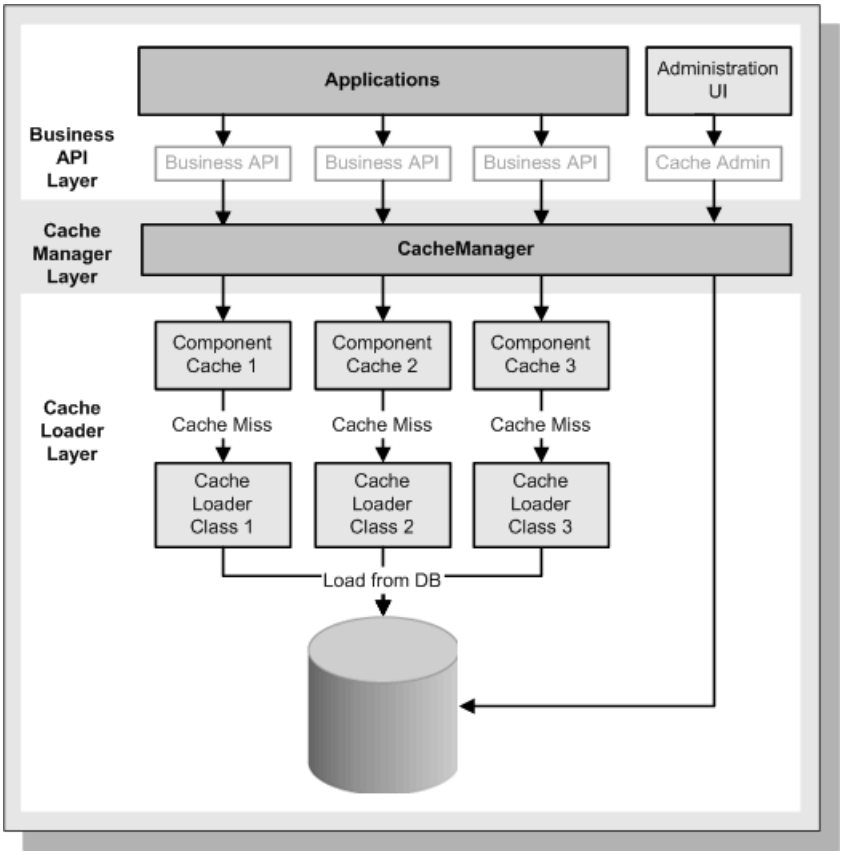
Caching provides a powerful, flexible, easy to use mechanism for storing database results and other Java objects in-memory for repeated usage. This minimizes expensive object initializations and database round-trips, thereby improving application performance.

Concepts

Application data is cached using Component Caches. Each Component Cache is identified by a name. The objects contained in a Component Cache are generally of the same type and share the same caching attributes. Each Component Cache has an associated cache loader class. The loader class has the logic for loading the cached object in case of a cache miss. When an object is requested from a Component Cache, if the object is found, it is returned from the cache. Otherwise, the loader is used to load the object and place it in the cache.

Architecture

The following diagram shows the architecture of the Caching Framework:



Terminology

- **Component Cache:** A logical cache identified by a name. It normally caches the data belonging to a given application component.
- **Cache Loader:** The class provided by the developer which has the logic for loading the object from the data source. Each Component Cache should have a corresponding loader class.
- **Cache Data Partitioning:** Partitioning classifies data within a Component Cache into further categories. Interaction with the cache is limited to data belonging to certain categories.

Features

The Caching Framework provides following features:

- **Synchronization:** Caching Framework takes care of all the synchronization issues in a multi-threaded environment.
- **Distributed Caching:** Distributed caching allows one JVM to inform all the other JVMs about updates to any cached data. This ensures consistency of the cached data across all the JVMS.
- **Data Partition:** This feature allows the cached data to be partitioned based on any partition key. This is useful for JVMS in hosting environments running against a Virtual Private Database (VPD). The Caching Framework ensures that the cache data is partitioned based on security group id.
- **Database Invalidation:** This feature ensures that when the data in a database is updated, the corresponding cached data in the mid-tier is also updated. This is useful when the updates to the database bypass the JVMS, such as updates through forms or sqlplus.
- **Event Support:** This feature allows custom event handlers to be invoked when certain events such as invalidation or updates to the cached data occur. This is useful for maintaining consistency across inter-dependent cached data.
- **Cache Administration:** Caching Framework comes with administration User interface, which is available under the Functional Administrator responsibility. This interface can be used to perform administrative operations including changing the time-out values for cache components, looking at cache usage statistics, and clearing caches.
- **Cache Diagnostics:** Cache diagnostics is a set of diagnostic tests that can identify some of the common problems that may occur when using the Caching Framework.

Benefits

The main benefit of the Caching Framework is increased performance, which is accomplished without any functional compromises. The Framework provides the infrastructure to ensure fast retrieval of the frequently used objects without any database roundtrips. Through distributed caching support and Database Invalidation support it ensures that the cached data is always in sync across mid-tiers and with the database. In addition to fast access, it also greatly reduces the load on the single database because the access is limited to mid-tiers that can scale out.

Another major benefit of the Caching Framework is having a single repository for all data that is long lived. Having a single place for frequently used static data allows easy implementation of memory management policies and administration.

Using local data structures to cache frequently used data has two major problems:

1. The data gets out of sync across java processes and with the database. The only option is to restart the java process to see the updated value.
2. There is no way to implement the central size management policies.

In addition to these benefits, Caching Framework comes with many other useful features as mentioned in the Features section above.

Using the Caching Framework

This section describes the steps involved in using the Caching Framework. To help clarify important concepts, we will use the Catalog cache in the iStore application as an example. Following are the steps for using the Caching Framework:

1. Identify the data that needs to be cached. Not all types of data is suitable for caching.
2. Create, through the UI, the seed data for a component cache. The seed data includes the name and other information such as loader class for the component cache.

   Note: The data identified in step 1 is cached in a component cache, which is a namespace within the Caching Framework. For example, a profile that is cached in a component cache can be named PROFILE_OPTION_VALUE_CACHE.
3. Provide the loader class for the component cache. The loader class contains the logic of loading the object and is used by the Caching Framework when there is a cache miss.
4. The component cache is now ready to use. You can use the Caching Framework APIs with the component cache name as defined in step 2.

   Note: The component cache is initialized as part of the application initialization. There is no need for any explicit initialization.

Loader Class Concepts

The loader class is provided by the developer (step 3) and contains the logic of loading the object from the cache for the specified key.

The application code requests an object from a cache component using the `get` API. The Caching Framework returns the corresponding object (value) if the object is present in the cache. If the corresponding object is not in the cache (cache miss), the Caching Framework uses the loader class to load the object, put in the cache, and return the object. Therefore, the `get` API is guaranteed to return the value corresponding to the key. While interacting with the Caching Framework there is no need to check for cache miss or explicitly put the value in the cache. Therefore, do not use the `put` API unless you intend to update a cached value.

Identifying What Data to Cache

It is important to correctly identify the best data to be cached. 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.
- **Many Updates:** Objects that are frequently updated are appropriate.
- **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.

  Note: Caching objects that are not actively used needlessly occupies JVM memory.

Product catalogs for the iStore application are an example of data that meets most of the above criteria. A site level profile is another example.

Setting Up the Environment (for OA Developers)

The following additional step is required for using the Caching Framework in an OA project:

- Go to **Project Properties > Run/Debug**. Select **Edit** to display the *Default Run Configuration*. Go to **Launch Settings** and in the **Java Options** field, enter **-DJTFDBCFILE= <dbcfileName location>**.

Implementing a CacheLoader Class

This section provides an overview of the loader class, a procedure to implement the Cache Loader class, and some sample code.

The loader class is used for automatic loading of an object if it is not present in the cache. This allows the cache to better manage the loading of objects and to keep the application code cleaner by making "if not in cache, then load" logic unnecessary. In addition, the Caching Framework ensures synchronization in loading the object if multiple threads request the same object from the cache. Objects can still be explicitly added to the cache using the `CacheManager.put()` API.

Procedure

Complete the following procedure to implement the Cache Loader class:

- Develop a class that extends the abstract loader class `oracle.apps.jtf.cache.CacheLoader`. Implement the `load(Object key,CacheContext cacheCtx)` method, where "key" is a key to identify the object in the cache.

```
public abstract java.lang.Object load(Object key, CacheContext
  cacheCtx) throws FrameworkException
```

This API is automatically invoked when there is a cache miss while calling the `CacheManager.get()` APIs. The values of the **CacheContext** and **key** parameters are the same as the supplied in the `CacheManager.get()` API. The objects used as keys have the same limitations as keys of a java.util.Hashtable. That is, they must implement the `hashcode` and `equals` methods.

Sample Code

```
package oracle.apps.ibe.catalog;
import oracle.apps.jtf.cache.CacheLoader;
import oracle.apps.jtf.cache.generic.CacheContext;
public class CatalogCacheLoader extends CacheLoader
{
  /*
  * Applications should extend this method to customize the loading
  * of cache object. This method is called by the caching service when
  * the requested object is not in the cache.
  *
  * @param key Key for which an object should be loaded for
  * @param cacheContext the CacheContext used to load the object
  *
  * @return an object for the given key value to be put into the cache,
  * null if the method fails to find a value for the key value.
  *
  * @exception FrameworkException if an error occurs
  */
  public Object load(Object key, CacheContext cctx)
  {
    CatalogItem catalogItemObj = null;
    /*
    *load the object from the database using the given key to identify the object.
    * Use the partition value from cctx and connection from AppsContext in cctx
    */
    return catalogItemObj;
  }
}
```

Warning: Do not use a dummy loader class that continually returns *null*. This is because it causes certain drawbacks and it also requires extra configuration steps. If it is needed, please consult the Caching Framework development team.

Registering a Component Cache

Registering a component cache involves selecting a name for the component cache as well as its different attributes. This is done through the Caching Framework UI. The component cache definition is stored as seed data in the database.

To register a component cache:

1. Log in to **http://<host:port>/OA_HTML/AppsLocalLogin.jsp** as *sysadmin/sysadmin*.
2. Choose the **Functional Developer** responsibility, select the **Core Services** tab and then select the **Cache Components** subtab.
3. Select the **Create Component** button. This opens the Component Cache creation page.
4. Specify a name for the component cache. This is a brief description that explains the type of objects that the component cache stores. For example, iStore Catalog Cache.
5. Specify the code of the component cache. This must have the application shortname as the prefix and must be unique within an application. For example **IBE_CATALOG_CACHE**.

   Note: This code identifier is used to access the component cache.
6. Specify the application name. For example, **iStore** or **IBE**.
7. Provide a component cache description.
8. Specify the **Loader Class Name**. This is the fully qualified name of the class that implements the logic for loading the object (given a key) in case of a cache miss. For example, `oracle.apps.ibe.catalog.CatalogCacheLoader`.
9. (Optional) Supply the workflow event name registered for database-driven invalidation.

   Note: This allows the invalidation of any cached object in the JVM by invoking the workflow event using PL/SQL in the database. This is suitable for components where the corresponding database tables can be updated by bypassing the JVM. For example, `oracle.apps.ibe.catalog.CatalogChangeEvent`.

The cache component policy must now be specified.

Note: These values can be changed through the Functional Administrator UI by the application administrator at a customer site.

10. Select the **Time Out Type** value. Choose either *Idle Time* (recommended) or *Time to Live* from the dropdown list. Both values refer to the duration after which the object is marked invalid.
    - **Idle Time:** Starts from the last time the object was requested from the cache. Choose this value when the primary consideration is the memory. This option prevents infrequently used objects from being cleaned up from the cache.
    - **Time to Live:** Starts from the time the object is loaded into the cache. Choose this value when the primary consideration is data consistency. This option guarantees that the values are refreshed after the specified time interval regardless of the usage.

      Note: A more suitable way of ensuring consistency is to use the distributed caching functionality and database invalidation.
11. Select the **Time Out After** value. This value refers to the **Time Out Type** chosen from step 10. Oracle recommends choosing *Global Idle Time*.
    - **Global Idle Time:** The component cache will get a timeout value that is equal to the global idle time specified. The current default is 15 minutes. This value can be changed as needed by the application administrator.
12. Select the **Apply** button to create the component cache.

Note: The component cache that you declare is distributed. The updates and invalidations to the objects in the component cache are distributed across other JVMs. This means that the keys that you are putting into the cache *must* be serializable.

The registered component cache is now ready to be used as described in the example below. The component cache will automatically be initialized from the seed data that you created with the first use of the Caching Framework.

Business API Example

This section provides an example business API and discusses how to use CacheContext object in the loader class and CacheManager APIs.

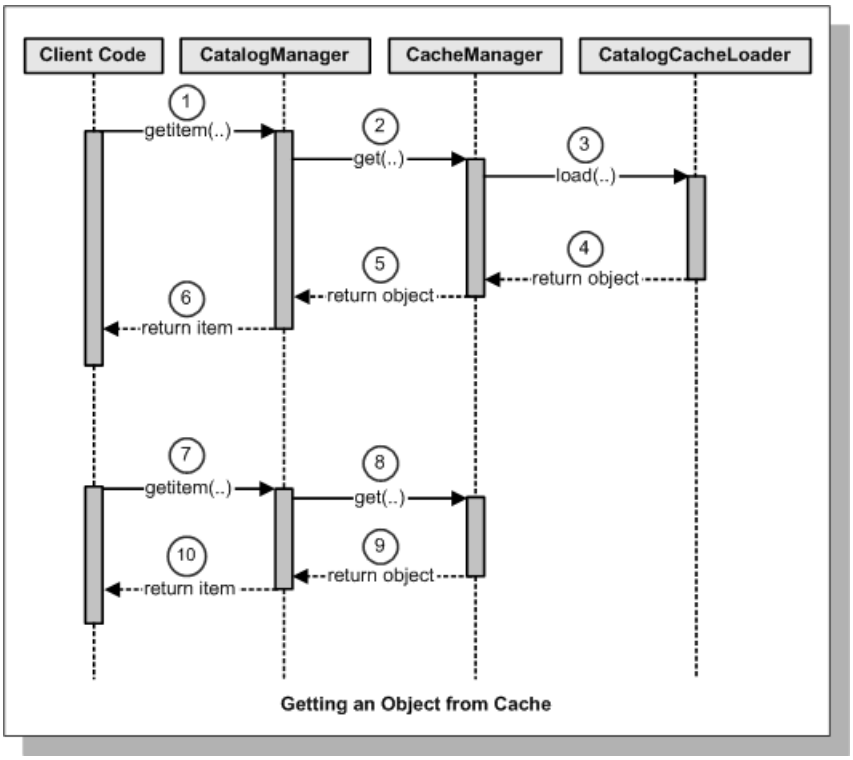
This example shows a class `CatalogManager` that caches the catalog data for iStore. The `CatalogCache` class uses the

Caching Framework to store the catalog items.

```
public class CatalogManager
{
  private static String CACHENAME = "IBE_CATALOG_CACHE";
  private static String APPSHORTNAME = "IBE";
  CatalogItem getItem(Integer itemID)
  {
    oracle.apps.fnd.common.AppsContext appsCtx = <get apps context instance>
    oracle.apps.jtf.cache.generic.CacheContext cctx = appsCtx. getAppsCacheCon;
    return (CatalogItem) CacheManager.get(CACHENAME,APPSHORTNAME, itemID,cctx);
  }
}
```

The use of Caching Framework is encapsulated inside a business API. The `CacheManager.get()` call returns the cached object. It either finds it in the cache store, or uses the CatalogCacheLoader class's `load(..)` API to load the object passing in **itemID** as an argument.

Note: The client should not explicitly check for the presence of the object in the cache (as is the case with hashtable or similar data structures). Please refer to the following graphic:



Getting an Object from Cache

Using CacheContext Object in the Loader Class and CacheManager APIs

Different APIs (`get, put, invalidate, bulkload`) take an instance of a `oracle.apps.jtf.cache.generic.CacheContext` as the parameter. The `CacheContext` instance is useful if you need to pass in additional helper information to the loader class (other than the key) that may be utilized for loading the object. When a `CacheManager` API is called with a `CacheContext` instance, the same instance is passed as the second parameter `cacheCtx` to the `load(Object key,CacheContext cacheCtx)` method in the loader class. You can fill in information in the `CacheContext` instance and retrieve it later in the loader class.

To obtain a `CacheContext` instance you must first obtain the `oracle.apps.fnd.common.AppsContext` instance as follows:

Note: This is helper information *only*. Refer to the latest documentation of respective technology stacks for the most current APIs.

OA Environment

```
((OADBTransactionImpl) rootAm.getOADBTransaction()).getAppsContext()
```

where `rootAm` is the root `ApplicationModule` instance. For the client side code, the `rootAm` can be obtained from the `OAPageContext` instance `pageContext`.

```
(OAPageContextImpl) pageContext.getRootApplicationModule()
```

JTT Environment

```
oracle.apps.jtf.base.session.SessionManager.getFWAppsContext();
```

fnd/AOLJ Code

An AppsContext instance can be created as follows:

```
oracle.apps.fnd.common.WebAppsContext waps =new
  oracle.apps.fnd.common.WebAppsContext(System.getProperty("JTFDBCFILE"));
```

From the `oracle.apps.fnd.common.AppsContext` instance you can obtain the `oracle.apps.jtf.cache.generic.CacheContext` instance as follows:

```
oracle.apps.fnd.common.AppsContext appsCtx = <get apps context instance>
oracle.apps.jtf.cache.generic.CacheContext cacheCtx =appsCtx. getAppsCacheContext();
```

Any information can be stored in the CacheContext instance as a name-value pair:

```
// set loader context if required
cacheCtx.setCacheLoaderContext(<myname>,<myvalue>);
```

The CacheContext instance can then be used in CacheManager APIs to get an object from the cache:

```
Object obj = CacheManager.get(<app>,<componentcache>,<key>,cacheCtx);
```

In case of a cache miss, the following loader class load method would be invoked:

```
public Object load(Object key, oracle.apps.jtf.cache.CacheContext cacheCtx)
```

The parameter `cacheCtx` is the same as `cacheCtx` used in the `CacheManager.get(..)` API. You can obtain `<myvalue>` from the `cacheCtx` as follows:

```
Object myvalue = cacheCtx.getCacheLoaderContext(<myvalue>);
```

If the instance `fndAppsCtx` is needed inside the loader class, it can be obtained using:

```
oracle.apps.fnd.common.AppsContext appsCtx =
  (oracle.apps.fnd.common.AppsContext)
  cacheCtx.get(oracle.apps.jtf.cache.AppsCacheContext.APPSCTXTKEY)
```

Note: The `fndAppsCtx` instance scope and therefore the associate `CacheContext` instance may be a user session. This means that any context set on the `CacheContext` instance will be present across multiple interactions with the cache. The implemented enhancement 4292050 provides an API to clear the context on the `CacheContext` instance. An alternate is to use the same key for different caches.

For `CacheManager` APIs that do not require a `CacheContext` instance as a parameter use the CRM/JTT session to obtain the value of the `CacheUserContext`. **Warning:** These APIs do not work correctly in an OA environment where the concept of the session is changed. Product teams currently using these APIs need to move to using the new APIs.

Cautions

Following are some observations regarding using Caching Framework:

> **Note:** You can test this by attempting to serialize the object. For sample code that checks to see if an object is serializable look at `isSerializable(Object o)` in `oracle.apps.jtf.cache.IASCacheProvider.java`.

- The cache should not be used like a Hashtable. The client logic is not -- *get the object from the cache and if null load the object and put it in the cache.* Instead, the client invokes the `CacheManager.get(..)` API.

  If the object is not in the cache, Caching Framework uses the corresponding loader class to load the object and put it into the cache. Do not explicitly put objects into the cache unless you are updating a value. This also suggests that you must write a proper loader class that has the logic to load an object rather than a dummy loader class that always returns null.

- The `CacheManager.put(..)` API is distributed. This means that in addition to putting the object in the cache, an invalidation message is sent to other JVMs, which will cause the corresponding key in these JVMs to be invalidated.

  Note: It's also possible to have a `put(..)` functionality that is local to the current JVM by passing an explicit flag.

- The keys that you are putting in the cache must be serializable. This generally means, ensuing:

    - The key class is declared as "implements `java.io.serializable`".
    - All the variables inside the class are serializable.
- There is no need to explicitly initialize the component cache. The component cache definition is stored as seed and Caching framework automatically initializes the component cache. Caching Framework APIs are ready to be used in the appropriate code.
- A single loader class instance is used for each component cache. This instance is shared between different threads. Remember this when any instance variables are used in the loader class implementation as access to the instance variable needs to be synchronized.
- Any configuration changes made to component caches through the System Administrator Console will *not* take effect until the JVMs are restarted.
- Multiple threads can access Caching Framework at the same time. Different threads access the same cached object if they use the same key in the same component cache. The application developer using the cache must make sure that the application is thread-safe by using synchronization mechanisms.

Caching Framework Seed Data

The Component Cache definition and the corresponding policy are stored as seed data in the database. This involves repeating the above steps for the creation of component caches against the seed database (seed120) using the Functional developer responsibility. For more details please see the Caching Framework Seed Data section.

Caching Framework Features

This section describes some of the main features of Caching Framework.

Distributed Cache

A distributed cache in Caching Framework allows for the invalidation and updating of cached objects to be propagated to multiple JVMs, including JVMs existing on different machines.

This section discusses:

- Propagation of invalidations and updates.
- Configuration for enabling distributed cache.

Propagation of Invalidations and Updates

The updates and invalidations of the objects in a distributed component cache are propagated to all the other JVMs that are running against the same database. This is suitable for data where the updates need to be seen across all JVMs simultaneously. This functionality, coupled with database-driven invalidation ensures the consistency of data across the JVMs and with the data source. For other types of objects, time-based invalidation ("Time to Live" or "Idle Time") may suffice. This guarantees that the objects in a JVM are reloaded after a specified duration.

Note: There is network overhead associated with declaring a component cache as distributed.

The following outlines how distributed cache functions:

1. Upon startup, each JVM auto registers the host it is running on in the database.
2. When an object gets invalidated or replaced in the cache, an invalidation message is sent to all other JVMs running against the database.
3. If the receiving JVM has the same key present in a component cache, then it is marked invalid.
4. This guarantees that the next time an object, using the same key, is requested from the cache it's loaded from the datasource.

If the component cache is defined to be a local cache, invalidations and updates will only affect the JVM where the `invalidate` or `put` API call is invoked. Please note the following:

- The distributed functionality is relevant only for the `invalidate` and `put` APIs. In each case where the component caches are configured as distributed, an invalidation message is sent, unless the `propagateUpdate` flag is used to send the whole object across. The object is *not* loaded into the other JVMs until it is requested by the client.
- Component caches using distributed functionality *must* use keys that are serializable. `Java.lang.String` is already serializable.

Configuration for Enabling Distributed Cache

Enabling distributed caching is a two step process:

1. Ensure distributed caching is turned on for a specific deployment or a specific JVM.
2. Ensure that the functionality is enabled for a specific component cache. The distributed caching can be turned on for the JVM by using Java D parameters. For more details please refer to the Cache Deployment section:

Follow the steps below to check whether the distributed flag is set for a particular component cache:

1. Login into the HTML Application Administrator Console (sysadmin/ sysadmin).
2. Select the Performance tab and then select **Components**, which is located on left side navigation bar.
3. Choose the correct application from the **View** dropdown list and then select the appropriate link under **Component Identifier**.
4. Verify the **Distributed Mode** checkbox is checked. To enable distributed caching for the component cache, this box needs to be checked.
5. If you change the setting, the Apache/ Jserv must be restarted for the setting to take effect.

See the Caching Guidelines section for details on when distributed caching should be used.

Cache Event Handling

This section provides an overview of handling cache events and discusses:

- Types of cache events.
- Event handling process.
- Implementing event handler.
- Registering event handler with Caching Framework.

Application Modules that use Caching Framework are interested in identifying and handling the cache events occurring in a particular component cache. Caching Framework provides a mechanism in which custom event handlers can be registered with the Framework for handling various events such as invalidations and updates to the objects. This feature is useful when objects in different component caches have inter-dependencies amongst themselves. Currently this feature is supported only when the distributed caching is enabled as both these features rely on the same underlying component. Refer to the Cache Deployment section for more details on this.

Types of Cache Events

In the preceding section, we discussed DB invalidation event, which is an external event that happens external to the caching system. Cache events can also be triggered within the caching system due to following:

- Explicit cache operations within the local caching system.
- In distributed caching, events can also be triggered by a remote caching system.
- Factors such as time and memory.

The following are the types of cache events based on how the event was triggered:

- OBJECT_INV_EXPLICIT
- OBJECT_INV_TIMEOUT
- OBJECT_INV_REMOVED
- OBJECT_INVALIDATED
- OBJECT_UPDATED_SYNC
- OBJECT_UPDATED

OBJECT_INV_EXPLICIT

Corresponds to the cache event triggered due an explicit `invalidate()` call.

OBJECT_INV_TIMEOUT

Corresponds to the cache event triggered due to timeout. The timeout happens after a cached object has exceeded *TimeToLive* seconds since the object was loaded into the cache, regardless of whether the object has been referenced or not. Another cause for timeout is when the cached object is idle for *IdleTime* seconds. In either case an `OBJECT_INV_TIMEOUT` event will result.

Note: *TimeToLive* and *IdleTime* are properties set with the Caching system when it is initialized.

OBJECT_INV_REMOVED

If the memory usage has reached above the maximum allowable limits, the caching system drains a few cached objects to make free space in memory. The `OBJECT_INV_REMOVED` event is triggered if a cached object is removed by the caching system due to free memory space.

OBJECT_INVALIDATED

The `OBJECT_INVALIDATED` event is triggered if either of the `OBJECT_INV_EXPLICIT` or `OBJECT_INV_TIMEOUT` events are triggered.

To handle a specific invalidation event, the application module should register the `FWKCacheEventListener` with identifiers `OBJECT_INV_EXPLICIT` or `OBJECT_INV_TIMEOUT` depending on the type of the event the application is interested in.

Note: Application developers should refrain from registering `FWKCacheEventListener` with the `OBJECT_INVALIDATED` event type along with `OBJECT_INV_EXPLICIT` or `OBJECT_INV_TIMEOUT` because this would result in the `OBJECT_INVALIDATED` to be raised and handled if any of the `OBJECT_INV_EXPLICIT` or `OBJECT_INV_TIMEOUT` gets raised.

OBJECT_UPDATED_SYNC

The Cache triggers the `OBJECT_UPDATED_SYNC` event when the application module updates a cached object. The Event Listener is called as a part of the update call synchronously and the update is considered not complete until the Event Listener completes.

OBJECT_UPDATED

The cache triggers the `OBJECT_UPDATED` event when the application module updates a cached object. The Event Listener is called asynchronously (by a background thread) so the update completes and the control is returned to the application module before the listener completes execution.

Event Handling Process

Event handlers registered with the component cache handle cache events. Caching Framework allows application modules to register one or more event handlers with the component cache. Each event handler is capable of handling multiple event types and maintains a list of event types that the handler can handle. When a cache event occurs in a component cache, each event handler registered with the component cache will be looked up to see if it can handle the event. If so, the handler will be invoked to handle the event.

Implementing Event Handler

Develop a class that extends the abstract cache event listener class `oracle.apps.jtf.cache.FWKCacheEventListener` and implement the `handleEvent(oracle.apps.jtf.ache.FWKCacheEvent tFWKCacheEvent)` method.

The `FWKCacheEvent` class encapsulates various event attributes that are required for handling the event, such as:

- Event Id
- Event Name
- Event key
- Component Key
- Application Short Name
- CacheContext
- Event Attributes

Event Id

The event Id is an integer value corresponding to the event type. The possible values include the following:

- CacheManager.OBJECT_INVALIDATED
- CacheManager.OBJECT_INV_EXPLICIT
- CacheManager.OBJECT_INV_TIMEOUT
- CacheManager.OBJECT_INV_REMOVED
- CacheManager.OBJECT_UPDATED
- CacheManager.OBJECT_UPDATED_SYNC

Event Name

The name of the cached object that triggered the event. This information can be assessed using `getEventName()` API.

Event Key

The key of the cached object that triggered the event. This information can be assessed using `getEventKey()` API.

Component Key

The component key for the component cache where the event occurred. This information can be assessed using `getComponentKey()` API.

Application Short Name

The application short name for the component cache where the event occurred. This information can be accessed using `getAppName()` API.

CacheContext

An instance of the `oracle.apps.jtf.cache.appsimpl.AppsCacheContext` class. It can also be used for getting the partition information associated with the component cache where the event occurred. This information can be accessed using `getCacheContext()` API.

Event Attributes

Includes any additional attributes for the event This can be accessed using `getEventAttribute(Object attributeKey)` API.

Sample Implementation

The following is a sample event handler implementation:

```
package oracle.apps.ibe.catalog;
import oracle.apps.jtf.cache.FWKCacheEvent;
import oracle.apps.jtf.cache.FWKCacheEventListener;
import oracle.apps.jtf.base.resources.FrameworkException;
public class CatalogCacheEventListener extends FWKCacheEventListener
{
 /**
  * Handles the Framework Cache Event.
  * The Application cache event listener class implementation should implement
  * this API with the event processing logic.
  * @param pFWKCacheEvent a FWKCacheEvent value
  * The pFWKCacheEvent represents the event object and encapsulates the event
  * attributes.
  * @exception FrameworkException
  */
  public void handleEvent(FWKCacheEvent pFWKCacheEvent) throws
    FrameworkException
  {
    int tEventId = pFWKCacheEvent.getEventId();
    String tEventName = pFWKCacheEvent.getEventName();
    String tEventKey = pFWKCacheEvent.getEventName();
    String tComponentKey = pFWKCacheEvent.getEventName();
    String tAppName = pFWKCacheEvent.getEventName();
    String tCacheContext = pFWKCacheEvent.getCacheContext();
   If(tEventId == CacheManager.OBJECT_INV_EXPLICIT)
    {
     //Processing for Explicit Invalidation Event
    }
    else if(tEventId == CacheManager.OBJECT_UPDATED)
    {
     //Processing for Object Updated Event
    }
  }
}
```

Registering Event Handler with Caching Framework

The event handler implementation must be registered with Caching Framework to receive call backs when an event is fired. Currently the only way to do it is through the API calls.

```
Int eventIds[2] = {CacheManager.OBJECT_INV_EXPLICIT,
  CacheManager.OBJECT_INV_UPDATED};

CatalogCacheEventListener tCatalogCacheEventLstnr =
  new CatalogCacheEventListener();

CacheManager.registerCacheEventListener(EmployeeInfoCache,HR,
  eventIds, tCatalogCacheEventLstnr);
```

Multiple handlers can be registered for a particular component cache. When an event is fired, the caching system checks each handler to see if the event type is registered with the handler. If the event type is registered, the `handleEvent(FWKCacheEvent)` API is invoked on the handler, which in turn handles the event.

DB Invalidation for Caching Framework

This section provides an overview of DB invalidation for Caching Framework and discusses how to:

- Create business event in Workflow.
- Register business event with component cache.
- Support DB invalidation in cache loader.
- Raise the event.
- Configure the DB Invalidation Feature

As explained above, DB Invalidation provides a general mechanism to originate events from the database. For Caching Framework, these events are used to invalidate objects in a component cache.

The cached data in the middle-tier is a representation of the same data in the database. If the data in the database gets updated, then the cached data needs to be invalidated. In most cases, the updates to the data in the database are originated in the middle-tier, which can subsequently update the cache or mark the data invalid. Distributed caching ensures that these updates are seen in all the middle-tiers. However, in some cases, these updates can happen outside of the middle-tier such as, updates through the Forms server or concurrent programs. In such cases, the DB invalidation mechanism can be used to ensure that the cached data is invalidated when the corresponding database data changes. This ensures that the cached data is always in sync with its datasource. This feature is enabled by applying the ARU 3688496 on top of the caching ARU. In addition, some configuration steps are required for making this feature work. Please refer to the section Configuration for DB Invalidation Feature.

The following is a typical process flow for database invalidation:

1. The user JVM (Apache/ Jserv) caches data such as profiles. The cached data is a representation of some profiles in the database.
2. The user performs an action such as, updating a profile value through Jinitiator, which causes the data in the database to be updated while passing the user JVM.
3. The cached profile value must now be marked as invalid. Subsequent access to the profile in the user JVM causes a cache miss, which in turn results in the updated value being loaded from the database.

The Workflow Business Event system provides necessary infrastructure support for notification of the database data change to the middle-tier. The notification is originated by raising a business event with the correct key. Caching Framework provides the support for processing the notification and invalidating the corresponding cached data in all the user JVM. The developer must ensure that the business event is created through seed data, it is raised with the correct key (a PL/SQL procedure) when the data in the database is updated, and tells Caching Framework which component cache corresponds to the seeded event by providing the event name as part of of the component cache definition. In addition, the developer can also optionally provide key conversion logic for converting the event key (string) to the cached key (object).

Consider the `CatalogCache` example, which is a component cache in the iStore application. The steps discussed in the *Create Business Event in Workflow* section below illustrate how this feature can be used. The steps related with Oracle Workflow configuration are just provided as a guideline. Please consult the *Oracle Workflow Guide* for the most up-to-date information.

Creating Business Event in Workflow

DB Invalidation is built on top of Oracle Workflow. A Business Event in Workflow needs to be created and subsequently raised. The following steps describe how to create a Business Event in Workflow:

Note: For examples, refer to the events `oracle.apps.fnd.profile.value.update` and `oracle.apps.fnd.flex.dff.compiled`. The information below is provided as a guideline only. See the *Workflow Developer Guide* for the latest information.

1. Log in to **Self Service Applications** in any environment.
2. Navigate to **Workflow Administrator Web Applications > Business Events**.
3. Select the **Create Event** button to create a new Business Event. On the Create Event page, provide the following details:

Label	Description
Name: (Name of the business event)	Example: `oracle.apps.ibe.catalog.CatalogChangeEvent`

Display Name: (Display name of the business event)	Example: Catalog Change Event
Status: (Status of the business event: *Enabled* or *Disabled*)	Example: Enabled
Owner Name: (Event owner name)	Example: iStore
Owner Tag: (Event owner tag)	Example: IBE

Â

4. Add this event to the Control Event Group `oracle.apps.fnd.bes.control.group` as follows:
   a. Select the Events subtab.
   b. Enter `oracle.apps.fnd.bes.control.group` in the Search box and then select **Go**.

      Note: The Search result returns a row where the name is oracle.apps.fnd.bes.control.group.

   c. Select the **Update** icon.
   d. Select the **Add Events to Group** button.
   e. Search for the event that you just created and select the **Add to Group button**, then select **Apply**.

      Note: The newly added event appears in the list of events for that event group.

For more information on BusinessEvent creation, please refer to the *Oracle Workflow Guide*.

Registering Business Event With Component Cache

This procedure registers the event created in the steps described in the previous section with Caching Framework. This creates an association between the event created and a specific component cache.

Important: The business event must be created in Workflow.

1. Log into **http://<host:port>/OA_HTML/AppsLocalLogin.jsp** using sysadmin/ sysadmin.
2. Choose the *Functional Developer* responsibility.
3. Select the Core Services tab, then select the Cache Framework subtab.
4. Search for the component cache and select **Edit**.
5. Enter the name of the business event that you created in the previous step in the **Business Event Name** field. For example, `oracle.apps.ibe.catalog.CatalogChangeEvent`.

   Note: If events are already registered with Workflow, developers can now add or update multiple business events separated by a semicolon (;).
6. Select **Apply**. This component cache is now receptive to the above BusinessEvent.

Note: Refer to `PROFILE_OPTION_VALUE_CACHE` as an example. This component cache is associated with `oracle.apps.fnd.profile.value.update`.

Supporting DB Invalidation in Cache Loader

When the PL/SQL layer raises the Business Event to invalidate the cache object, it provides the key of the object to be invalidated in the form of a String. However, Caching Framework allows an object to be used as the key. Similarly, if the Java layer raises the Business Event, then it has to provide the key of the object in the form of String.

To support the above two requirements, the `CacheLoader` implementation should override the two APIs below and provide custom implementation.

```
/**
 * Converts the key in Object form to String
 * @param key the key
 * @return the key converted to a String
 */
public String keyToString(Object key)
/**
 * Converts the key in String form to an Object array.
 * @param key the key
 * @return the key as an Object array
 */
```

```
public Object[] stringToKey(String key, FWKCacheEvent fwkCacheEvent)
```

The first API is used while raising the Business Event in the Java layer. The second is used while processing the BusinessEvent. The event is raised in the PL/SQL layer and a VARCHAR2 is supplied as the key, for example, **strKey**. This key is converted into a array of objects by using the `stringToKey(..)` API. These objects serve as the keys to be invalidated in the component cache. The `oracle.apps.jtf.cache.CacheLoader` class provides the default implementation of these APIs which uses `toString()` for Object-to-String conversion and returns an object array of one string element for String-to-Object conversion.

Note: For an example, refer to `oracle.apps.fnd.flexj.FlexfieldCacheLoader`.

Raising the Event

When all of the above steps are completed, a DB Invalidation Business Event can be raised to invalidate the cache object in all middle-tiers. From the cache perspective, this is needed for the cached data where the corresponding database data can be updated by bypassing the Apache-Jserv middle-tiers. Typically the event is raised in the PL/SQL layer inside the PL/SQL procedure to update the data. The developer *must* ensure that the updates to the database data eventually come through this PL/SQL procedure.

Note: For example code on how to raise an event, refer to AFPFPROB.pls.

The following is an example of how to raise the event:

PL/SQL procedure for raising an Invalidation Business Event:

```
WF_EVENT.RAISE (
                P_EVENT_NAME IN VARCHAR2,
                P_EVENT_KEY IN VARCHAR2,
                P_PARAMETERS IN WF_PARAMETER_LIST_T);
```

Example for invalidating the cache object identified by key Summer_CatalogÂ in IBE_CATALOG_CACHE:

```
PROCEDURE UPDATE_CATALOG( P_CATALOG_KEY IN VARCHAR2) IS
-- update the catalog with the key = Summer_Catalog
L_PARAMETER_LIST WF_PARAMETER_LIST_T := WF_PARAMETER_LIST_T();
WF_EVENT.RAISE(
               P_EVENT_NAME = oracle.apps.ibe.catalog.CatalogChangeEvent,
               P_EVENT_KEY = P_CATALOG_KEY,
               P_PARAMETERS = L_PARAMETER_LIST);
```

This BusinessEvent is processed by the special JVM started by the concurrent manager that is running the workflow java deferred agent listener service. The processing of the BusinessEvent causes the following `CacheManager` API to be invoked:

```
public static void invalidate(String componentId, String appName, Object key,
  CacheContext cacheContext) throws FrameworkException
```

Here are the values for different parameters:

Parameter	Description	Value
componentId	The component cache with which the event was registered. See Register Business Event with Component Cache.	Example: IBE_CATALOG_CACHE
appName	The application under which the event was registered.	Example: IBE
key	Obtained from the P_EVENT_KEY value in the PL/SQL layer. The `stringToKey()` method of the cache loader for the component cache is invoked to convert it into a String.	Example: Summer_Catalog
cacheContext	The `cacheContext` contains the data partition values.	For an Application environment, these are the security group ID and the database ID. By default these values are obtained from the database session, which raises the event in the PL/SQL layer. **Note:** It's also possible to specify the security group ID and database ID values explicitly while raising the event. See below.

As a result of the processing of event, the corresponding cached object is marked invalid.

Example Code for Specifying the Security Group ID and Database ID Values Explicitly

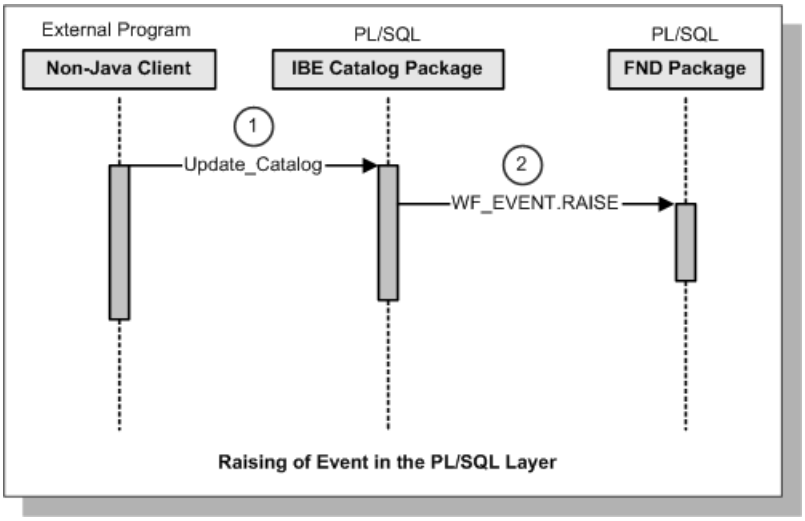
```
L_PARAMETER1 WF_PARAMETER_T:= WF_PARAMETER_T('BES_SECURITY_GROUP_ID');
L_PARAMETER2 WF_PARAMETER_T:= WF_PARAMETER_T('BES_DATABASE_ID');
L_PARAMETER_LIST.EXTEND;
L_PARAMETER_LIST(1):= L_PARAMETER1;
L_PARAMETER_LIST.EXTEND;
L_PARAMETER_LIST(2):= L_PARAMETER2;
```

Additional Items Regarding Event Processing

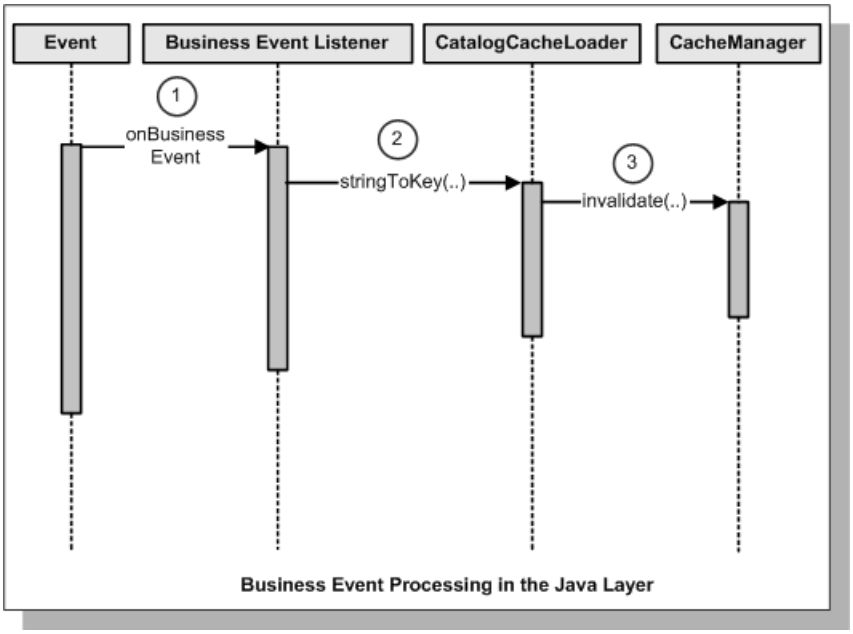
This section provides important facts regarding event processing:

- Every time an event is raised, it is processed in a separate thread.
- The events are processed by the Java Deferred Agent Listener running in a standalone JVM (GSM) started by a concurrent process. This JVM must be running all the time. This JVM sends a distributed invalidation message causing the corresponding key in all of the JVMs to be marked invalid.

The following graphics illustrate how a business event is raised in the PL/SQL layer and processed in the Java layer:



Raising of Event in the PL/SQL Layer



Business Event Processing in the Java Layer

Configuring DB Invalidation Feature

This feature is built using the business event support provided by Oracle Workflow, the apps caching infrastructure and the

underlying distributed caching functionality provided by IAS cache.

When an update to the database data happens, a workflow event is raised. The Workflow JVM running the Java Deferred Agent Listener processes this event. This JVM then issues a distributed invalidation message to all the other JVM's running against the same database and the cached data in the JVM is invalidated. The following configuration is necessary for this feature:

1. Caching Framework *must* be running in the distributed mode. This is the default configuration of Caching Framework. The **-DLONG_RUNNING_JVM=true** is set and it ensures that Caching Framework runs in distributed mode. For backward compatibility the same can be ensured by setting **-DCACHEMODE=DISTRIBUTED**.

2. The Workflow Java Deferred Agent Listener *must* be running. This runs as a thread in a special workflow service container JVM and processes the business events being raised in the database. This should be running by default. To verify, make sure that Oracle Applications Manager (OAM) is running, login into SSA and perform the following steps:
   a. Select the *System Administrator* responsibility.
   b. Select **Workflow** (OAM).
   c. Select on the icon next to **Agent Listeners**.
   d. Query for Workflow Java Deferred Agent Listener by selecting *Name* from the **Filter** dropdown list.

      Note: If the **Status** column shows *Running* then skip the following steps.

   e. Select the **Workflow Agent Listener Service** link (under Container).
   f. Select **Workflow Agent Listener Service** and if the status is not running, select **Start** from the dropdown list. Then select **Go**.

      Note: Make sure the status changes to the green icon representing a running state.

   g. Return to the Workflow Java Deferred Agent Listener page. Select **Start** from the dropdown list (under Actions) and then select **Go**.

      The **Status** column should show *Running*.

For additional trouble-shooting please refer to the Testing and Troubleshooting section.

Caching Guidelines

Proper use of caching improves application performance by reducing database round-trips. However, cached objects remain in the memory, as they are stored in the cache (which is static in scope). Not all data objects are suitable for caching. In addition, using distributed caching is not feasible all the time, as there are network overheads associated with its use. Below are general guidelines for using Caching Framework.

Choosing the Data to Cache

For information about what data to cache, see "Using the Caching Framework", Identifying What data to Cache.

Naming the Component Cache

The component cache name should start with the application short name as the prefix. For example, all caches defined for istore (IBE) should start with IBE as the prefix as in IBE_CATALOG_CACHE.

Ensuring Data Consistency

This section provides an overview of ensuring data consistency and discusses:

- Consistency from external updates.
- Time-based invalidation.

Objects in the middle-tier are a representation of some data in a data source (usually the database). Ideally, all cached objects should be consistent with their data source and across all JVMs.

The following is the sequence of operations a developer *must* perform to ensure that cached data is consistent:

- When an update occurs to the cached data in a JVM, the update should be written through to the data source. This typically means updating the cached data to the database.

- The developer *must* invoke either of the following two APIs to inform Caching Framework about the update:

  - Invoke `CacheManager.put(...)` API to explicitly replace the old cached instance with the new instance.
  - Invoke the `CacheManager.invalidate(...)` API so that he next time the object is requested from the cache, it is loaded from the data source. This is used when the updated instance of the object is not available.

This ensures that for the component caches, which are declared distributed, the invalidation message is sent to the other JVM. As the updated object is already saved to the database, the next time the object is requested in any of the other JVMs, a cache miss occurs and the updated instance of the object is loaded.

Consistency from External Updates

Database data can in some cases be updated outside the JVM. This can happen through sqlloader, concurrent programs, or through a different JVM such as JInitiator. When this happens, the data being cached should reflect this change. To handle this, a database-driven invalidation mechanism is available. This allows developers to invoke a PL/SQL procedure with appropriate parameters when updates to database data occurs. The changes are then picked up by the JVMs. For details, see the Database-Driven Invalidation section. For component caches that use both distributed caching and database-driven invalidation features to ensure that data is always consistent, you can specify a "0" value for the **Time to Live** or **Idle Time**. This means that the cached data is always valid and it is not cleaned from the cache until dictated by memory constraints.

Time-Based Invalidation

Time-Based Invalidation is useful when it is acceptable for the cached data to be out of sync with the data source for the specified duration. For such data, there is no need to use either distributed caching or database-driven invalidation. Instead, a positive value for either the **Time to Live** or **Idle Time** can be specified. This ensures that fresh data is loaded after the specified duration. The changes to the cached data in the other JVMs are seen only after the specified duration of time expires. The advantage of this setup is there is no associated network overhead.

Distributed Cache Guidelines

The distributed cache incurs network overhead when invalidation messages are sent to all participating JVMs. Because the number of JVMs can be large, only data that seldom changes should be cached.

Database Invalidation Guidelines

Database invalidation is useful *only* when the updates to the database data bypass the JVMs and the corresponding cached data becomes out of sync with the database. The database invalidation mechanism is built using Oracle Workflow and has considerable overhead associated with it. This mechanism is suitable only when the updates to database data are infrequent and the corresponding cached data must be invalidated right away. Following are two alternatives, which are available for other circumstances:

- **Using the cache timeout:** This guarantees that the cached data is not older than the specified duration.
- **Restarting the middle tier:** This option can be used when large amounts of data are being loaded into the database and downtime is acceptable.

Cache Administration

This section discusses Cache Deployment.

Cache Deployment

Note: The following cache deployment steps are required *only* if you are using either the distributed caching feature or the cache event handling feature.

If your environment is auto-config enabled apply the tech stack rollup patch H 3416234. Otherwise, the equivalent java -D parameters *must* be added. For an Apache/Jserv these are specified in the jserv.properties file as:

```
wrapper.bin.parameters=-D<name>=<value>
```

The list of parameters is as below:

- **-DAPPLRGF=<*a writeable log directory*>**

  The writeable log directory can be the Jserv log directory. A file with the name *javacache.log* is created under this directory. If it is not specified, the java current working directory is the default. If the default directory is not writeable, an error message is written to the jserv log file without any adverse effect on the functionality.

- **-DLONG_RUNNING_JVM=true**

APIs

This section discusses the APIs that a developer requires to access the cache.

See the Javadocs for Caching Framework for further information.

CacheManager APIs

The `oracle.apps.jtf.cache.CacheManager` class exposes the APIs a developer needs to access the cache.

```
public static Object get(String componentId,
                         String appName,
                         Object key,
                         CacheContext cacheContext)
                         throws FrameworkException
```

This API retrieves an object from the cache. If the object cannot be found in the cache, the `CacheLoader.load(Object key, CacheContext context)` method is called to load the object into the cache. The `CacheContext` parameter is used to extract data partition names and their corresponding values.

```
public static void put(String componentId,
                       String appName,
                       Object key,
                       Object value,
                       boolean propagateUpdate,
                       CacheContext cacheContext)
                       throws FrameworkException
```

This API explicitly puts an object into the given component cache. If a key already exists in the cache it is replaced with the new value. If not, a new entry is added to the cache. The `propagateUpdate` flag is relevant only if the component cache is declared distributed. When true, it sends the *value* object across to the other JVMs. The value object must be serializable. When the flag is false, an invalidation message is all that is sent across to the other JVMs. The *key* object must be serializable in both cases. If the component cache is data-partition-aware, then the `CacheContext` parameter is used to extract the data partition values to stripe the cache data.

```
public static void invalidate(String componentId,
                              String appName,
                              Object key,
CacheContext cacheContext)
throws FrameworkException
```

This API invalidates an object with the given key in the component cache. The `CacheContext` parameter is used to extract the data partition values. The data partition values are used to identify cache data if the component cache is data-partition-aware. If the component cache is declared distributed, then the invalidation message is sent across to the other JVMs.

```
public static boolean isPresent(String componentId,
                                String appName,
                                Object key,
CacheContext cacheContext)
throws FrameworkException
```

This API determines whether an object with the given key exists in the specified component cache. The `CacheContext` parameter is used to extract the data partition values. The data partition values are used to identify cache data if the component cache is data-partition-aware.

```
public static Object[] keys(String componentId,
                            String appName,
CacheContext cacheContext)
throws FrameworkException
```

This API returns all keys of a specific component cache. The `CacheContext` parameter is used to extract the data partition values. The data partition values are used to identify cache data if the component cache is data-partition-aware.

```
public static Object[] values(String componentId,
                              String appName,
CacheContext cacheContext)
throws FrameworkException
```

This API returns all values in a component cache. The `CacheContext` parameter is used to extract the data partition values. The data partition values are used to identify cache data if the component cache is data-partition-aware.

```
public static void bulkload (String componentId,
                             String appName,
CacheContext cacheContext)
throws FrameworkException
```

This API explicitly bulkloads a component cache. This is different from the bulkload of a component cache during cache initialization.

During initialization, the component cache is bulkloaded for all data partition values if the component cache is data-partition-aware. In this explicit `CacheManager.bulkload()` API, the component cache is only bulkloaded for the data partition values contained in the `cacheContext`. In both cases, the `CacheLoader.bulkload()` API is invoked, as described in the Implementing a CacheLoader Class section.

This API is especially useful if a component cache is invalidated for a specific data partition ID by `CacheManager.invalidate(componentId, appName, cacheContext)`. This explicit `bulkload` API can then be used to warm up the component cache. The `CacheContext` parameter is used to extract the data partition ID to be bulkloaded. The `CacheContext` parameter in the `CacheManager.bulkload()` API is passed to the `CacheLoader.bulkload(cacheContext)` API. The developer who implements the `CacheLoader.bulkload(cacheContext)` API can use the `CacheContext` instance to preload the right data for a specific data partition IDs.

Error Messages

The list below summarizes the error messages you might encounter with Caching Framework:

- **JTF-1966:** Caching Framework not initialized. Check Logger output for errors, especially look for JTF-1968 or JTF-1970.
- **JTF-1967:** Invalid operation. Either the Component Cache is not registered in the Caching Framework, or the application is not enabled for the machine.
- **JTF-1968:**Â Caching Framework fails to initialize. Please check error log for details.
- **JTF-1969:** An error occurred while reading the Caching Framework configuration from the database.
- **JTF-1970**: Error occurred while trying to instantiate a CacheProvider object. Make sure that the cache provider is one of the two listed earlier in this guide and that it is included in the classpath.
- **JTF-1971:** An error occurred in the Cache Framework operation. Check the associated exception for details.
- **JTF-1972:** Component name for the Caching Framework cannot be null.
- **JTF-1973:** Failure to register a new Component Cache, which has the same name as an existing one. Please contact the JTT team if you see this error.
- **JTF-1974:** The CacheProvider abstract method in Caching Framework is not implemented in the class. Please contact the JTT team if you see this error.

Testing and Troubleshooting

This section discusses how to:

- Test the component cache.
- Test database validation.

Testing the Component Cache

To troubleshoot Caching Framework use the Diagnostic Framework. The diagnostic test for Caching Framework can be accessed from the Oracle Diagnostic UI (DEV120 env):

`http://<hostname:port>/OA_HTML/jtfqalgn.htm`

To log into the system use sysadmin/ sysadmin. Alternatively, the same can be accessed by logging into AppsLocalLogin as sysadmin, selecting either *CRM ETF Administration* or *CRM HTML Administration* responsibility, and then selecting the **Diagnostics** link under **Setup**.

There are two types of diagnostic tests:

1. **Basic tests to verify the basic configuration related with Caching Framework:** These tests do not require any user inputs and are more suitable for the system administrator to run. To access, select the Basic tab, choose *CRM Foundation* from the **Application** dropdown list, and then select **Caching Framework** on the left side. Select *Run Without Prerequisite* to run the test. Select the icon under the **Report** column to see the results. This test prints out the basic information about Caching Framework's configuration and component caches.
2. **Advanced tests that can be used to troubleshoot a specific component cache:** To access, select the Advanced tab, select *CRM Foundation* from the **Application** dropdown list, and then select *Cache Component Diagnostic Test* on the left side. You can run tests on individual component caches. The *put* and *invalidate* tests use a dummy key to *put* and *invalidate* an object in a component cache. For the *get* test, the value of **Cache Object Key** *must* be supplied. Currently the only key type supported is a String type. If the key is an implementation of a class, then there should be a toString() implementaion in that

class. Similarly for value also. The loader classes should override the keyFormat and valueFormat methods(present in CacheLoader.java class), if they want to show any specific format for the key and value in the advanced diagnostic test. For the *get* test, the `CacheManager.get()` method is invoked with the specified key as a String.

Below are some additional troubleshooting tips for component caches using non-string keys that can be used during development.

1. To verify that the object is actually being cached, make sure that the **Time to Live** and **Idle Time** are set to "*0*". Write a short test program with two consecutive `CacheManager.get(..)` calls. Put a break point (or `System.out`) in your loader class `load()` method. Only break once, or have one output from `System.out`, as the second invocation of `CacheManager.get(..)` will return the cached object. Also, the object reference from the two `get` calls should be identical.

2. If the object is not being cached, check the loader class `load()` method outside of Caching Framework. The simplest way is to create an instance of the loader class using it's default constructor. Invoke the `load()` method and see whether it returns the expected object back.

   Note: The same can be used to verify the `stringToKey()` API in the loader class used for the database invalidation functionality.

3. The objects in the cache can be enumerated using the `CacheManager.keys(...)` and `CacheManager.values(...)` API's.

Testing Database Invalidation

This section discusses how to:

- Perform database invalidation tests.
- Bounce the workflow JVM.
- Verify the workflow event is getting raised.
- Verify workflow event processing.
- Verify Java object cache log file location.

Performing Database Invalidation Tests

To test your implemenation, If you have implemented database invalidation:

1. Perform an action that causes updates to the database data while bypassing the user JVM (typically Apache/ Jserv), such as updating a profile through Jinitiator.

   To test, the user *must* update the data in the database through a UI or directly. Also, the user *must* ensure that an event is raised when the update occurs.

   Note: The event can be raised using the PL/SQL API or through the Workflow Administrator Web Application UI.

2. Access the corresponding cached object. If database invalidation works correctly, you will see the updated value.

3. To test, the value of the data in the user JVM must be accessed. It should be the new updated value. Alternatively, you can examine the keys in the cache through the Advanced Diagnostics test. The key that corresponds to the data that is getting updated in step 1 above, must not be present in the cache after the update occurs.

   Note: Because the processing of events is asynchronous, you must wait at least one or two minutes after updating the data to see the effect in the middle-tier data.

For database invalidation to work correctly, all the different underlying pieces must work as expected. Some of these pieces belong to the infrastructure and some are supplied by the user. All the seed data must be in place, the background services *must* be running, and the runtime must behave as expected. Diagnostic tests are provided for checking the configuration. The **Cache DB Invalidation** test, under the Basic diagnostics tab, can be run to verify that the general infrastructure configuration is in place.

The CacheComponent DB Event Invalidation test, under the Advanced diagnostics tab, can be run for a specific component cache for which the database invalidation functionality is being tested. The poplist lists *only* the component caches that have corresponding workflow defined. If your component cache does not appear, check the component cache definition to make sure that the business event is defined.

If all the diagnostic tests pass but the functionality still does not work, check the runtime behavior by:

- Bouncing the workflow JVM.
- Verifying the workflow event is getting raised.
- Verifying the workflow event is processed correctly.
- Verifying the Java object cache log file location.

Bouncing the Workflow JVM

After applying the appropriate patch, the workflow JVM that is running the Java Deferred Agent Listener, needs to be bounced.

Note: This is not the Apache Jserv process.

To bounce the workflow JVM:

1. Login to SSA and select the *System Administrator* responsibility.
2. Select **Workflow** (under Oracle Applications Manager (OAM).
3. Select the **Agent Listeners** icon and locate the row where the **Name** column is *Workflow Java Deferred Agent Listener*.
4. Select the *Workflow Agent Listener Service* under the **Container** column.
5. Verify that the radio button for the *Workflow Agent Listener Service* row under the **Select** column is selected.
6. Select **Restart** from the second dropdown list and then select **Go**.
7. Select **OK**. The **State** column changes to *Restarting*.
8. After a short delay, select **Reload**. The **State** column changes to *Activated*.

Verifying the Workflow Event is Getting Raised

To verify that the workflow event is getting raised:

1. Issue the following query:

   ```
   select to_char(enq_time, 'yyyy-mm-dd hh24:mi:ss') ,to_char(deq_time,'yyyy-mm-dd
   hh24:mi:ss') ,msg_state,user_data from applsys.aq$wf_java_deferred where enq_time >
   to_date ('2004-11-1 11:19:00','yyyy-mm-dd hh24:mi:ss') order by enq_time desc;
   ```

   Notes:
   - The `to_date()` value should not be more than a few seconds greater than the time **t**.
   - This query should return at least one row where the string representation of `user_data` value contains text of the form `('BES_EVENT_NAME', 100, 'oracle.apps.fnd.menu.entry.insert')` or `('BES_EVENT_NAME', 100, 'oracle.apps.fnd.menu.entry.update)`.
   - For profile update it would be `('BES_EVENT_NAME', 100, 'oracle.apps.fnd.profile.value.update')`.
   - The third value, `oracle.apps.fnd.profile.value.update`, is the `'event name'`. It should also have text of the form `AQ$_JMS_USERPROPERTY('BES_EVENT_KEY', 100, '10001:0:0:FND_SSO_LOCAL_LOGIN_MASK')` where the third value should correspond to the data that got updated/ created.
   - The value `'10001:0:0:FND_SSO_LOCAL_LOGIN_MASK'` is the `'event key'`.

2. The value of the **msg_state** column begins as *Ready* and then it changes to *Processed*.

   Note: Attempt to raise an event through the Workflow Administrator Web Application UI by selecting **Events**, searching for the desired event, and then using the icon under the **Test** column in the Search result.

Verifying Workflow Event Processing

Event processing can be confirmed by examining the log files. This requires changing the log level of the workflow JVM that is running the Java Deferred Agent Listener, which processes the event.

To change the log level:

1. Go to Bouncing the Workflow JVM and complete steps 1-3.
2. Locate the row where the **Name** column is *Workflow Java Deferred Agent Listener*.
3. Select the checkbox under the **Select** column, which is located next to the row mentioned in the previous step.
4. Select **Edit**.
5. Select Next to go to a second page. Select *Procedure* from the **Log Level** dropdown list.

   Note: No changes are required if the **Log Level** value is *Procedure* or *Statement*.

6. Select Finish.
7. Restart the Workflow Agent Listener service. See Bouncing the Workflow JVM

To examine the runtime logs:

1. Cause a database invalidation event by performing an action to update the data. See Performing Database Invalidation Tests.
2. Go to Bouncing the Workflow JVM and complete steps 1-3.
3. Locate the row where the **Name** column is *Workflow Java Deferred Agent Listener*.
4. Select **View Log**.

   The log file contains the following series of messages:

   - Business Event received in the cache handler:<event name> with the workflow context:<context value>.
   - Business Event=<event name>key=<event key>corresponds to the app=<Application Short Name>and component key=<Cache Component Key>and loader class=<loader class name>.
   - Just about to call stringToKey with key=<event key>.
   - Obtained the keys <keys>.
   - Invoking CacheManager.invalidate for component=,Cache Component Key> app=<Application Short Name> and key=<list of keys>.

     Note: This message should appear at least once.

Verifying Java Object Cache Log File Location

The location of the Java object cache log file is available as the output of running the Caching Framework diagnostic test in the **Basic** mode. The location is the value of the form:

IAS LogFile Name = /slot02/appmgr/tsk1029acomn/rgf/tsk1029a/javacache.log

The location needs to writeable and can be changed through auto-config or by changing the java -DAPPLRGEF=<new location>.

Report any exceptions in the log file.

Caching Framework Seed Data

Seed data is data that is required by applications or components in order to work "out of the box." Seed data refers to the data that comes embedded with the applications as opposed to custom or user data that is entered by application users. Seed data is added to Oracle applications to allow users to use native functionality without having to perform additional configuration. Seed data can be thought of as "boot-strapping" data which is inherent to the application.

This section discusses:

- Terms.
- Standard WHO columns.
- How to use seed data.
- Registering component cache seed data.
- Cache filter seed data.
- System policy seed data.

Terms

- **FNDLOAD:** This utility from the AOL team is used to upload or download seed data to and from a database.
- **LCT File:** This is the control file that contains the rules used by FNDLOAD to upload or download data to and from schema objects. LCT files are created by the owner of the schema objects. Typically the download mechanism stripes the data downloaded by application ID. This is helpful to differentiate data owned by different product teams.
- **LDT File:** This is the file that is generated when FNDLOAD is run against a database.

Standard WHO Columns

- **CREATED_BY:** Stores the USER_ID of the person creating the row/data. This column is present in all Oracle Application tables.
- **CREATION_DATE:** Stores the date of creation for the row being entered by the user. This column is present in all Oracle Application tables.
- **LAST_UPDATED_BY:** Stores the USER_ID of the person who last updated the row. This column is present in all Oracle Application tables.

- **LAST_UPDATE_DATE:** Stores the last updated date of the particular row. This column is present in all Oracle Application tables.
- **Object Version:** This represents the version of the object that is being modified. It refers to the standard version concept used to keep track of changes being made to the object.

How to Use Seed Data

Log into the System Administrator Console. Seed data that is to be entered into the SEED database for every release (for example, SEED120) must be entered using the DATAMERGE/DATAMERGE user. Using any other user will override the seed data identifier.

Application teams using Performance Framework must register and extract the following seed data:

- Caching Framework seed data.
- Component Cache.
- Cache Filter.
- System Policy.

The LCT file to download and upload component caches is provided by the OA Framework team. LCT files are found under **$JTF_TOP/patch/115/import**. Development teams using the Oracle E-Business Suite Caching Framework must register all the component caches that belong to their application in the seed database.

Note: Every application team owns the component cache seed data belonging to their application.

Registering Component Cache Seed Data

This section discusses:

- Database tables.
- LCT file.
- Naming convention.

Application developers must register all the component caches belonging to their application through the System Administrator Console UI.

Database Tables

The following tables are used to store the information about component cache:

- jtf_prefab_ca_comps_b
- jtf_prefab_ca_comps_tl
- jtf_prefab_ha_comps

LCT File

The LCT file used for component cache is **jtfprefabcache.lct**. Each application team should download component cache seed data for their application by specifying the application_short_name parameter. The command to download the data as follows:

```
FNDLOAD apps/apps@seed120 0 Y
  DOWNLOAD jtfprefabcache.lct <ldt file name> JTF_PREFAB_CA_COMPS
  APPLICATION_SHORT_NAME=<your application short name>
```

For example for istore (IBE) the command would be:

```
FNDLOAD apps/apps@seed120 0 Y
  DOWNLOAD jtfprefabcache.lct ibecachedef.ldt JTF_PREFAB_CA_COMPS APPLICATION_SHORT_NAME=IBE
```

Naming Convention

The LDT file should be named **<*appshortname*>ccache.ldt** For example, jtfccache.ldt.

Cache Filter Seed Data

This section discusses:

- Database tables.
- LCT file.
- Naming convention.

Application developers should register all the cache filters belonging to their application through the System Administrator Console UI. For details, refer to the online help of the System Administrator Console or the corresponding manual, the *Oracle E-Business Suite CRM System Administrator's Guide*.

Database Tables

The following tables are used to store information about cache filter:

- jtf_prefab_ca_filters_b
- jtf_prefab_ca_filters_tl
- jtf_prefab_ha_filters

LCT File

The LCT file used for cache filter is **jtfprefabcache.lct**. Each application team should download seed data for their application by specifying the application_ short_name parameter. The command to download the data is as follows.

```
FNDLOAD apps/apps@seed120 0 Y
  DOWNLOAD jtfprefabcache.lct <ldt file name> JTF_ PREFAB_CA_FILTERS
  APPLICATION_SHORT_NAME=<your application short name>
```

For example for istore (IBE) the command would be:

```
FNDLOAD apps/apps@seed120 0 Y
  DOWNLOAD jtfprefabcache.lct ibecachefilters

  JTF_ PREFAB_CA_FILTERS APPLICATION_SHORT_NAME=IBE
```

Naming Convention

The LDT file should be named **<*appshortname*>cfilter.ldt**. For example, ibecfilter.ldt for istore application (IBE).

System Policy Seed Data

This section discusses:

- Database tables.
- LCT file.
- Naming convention.

A system policy defines the resources available or unavailable for the cache. Currently, it specifies the limit on the JVM memory at which no more objects are cached. The shipped value is 750 MB.

Note: This data is seeded and extracted by the OA Framework team.

Database Tables

The following table is used to store information about System Policies.

- jtf_prefab_sys_policies

LCT File

The LCT file used for system policy is **jtfprefabpolicy.lct**. As the data is not shared by other application developers, they are not striped by application short name. The command to download the data is shown below.

```
FNDLOAD apps/apps 0 Y DOWNLOAD
  jtfprefabpolicy.lct <ldt file name> JTF_PREFAB_ SYS_POLICIES
```

Naming Convention

The LDT file should be named **<*appshortname*>spolicy.ldt**. For example, jtfspolicy.ldt.

Known Issues

A loader class returning a *null* value should not be used even though Caching Framework can handle it. It requires dynamic registration of component caches, which can potentially affect some other administration features.

Frequently Asked Questions

Q. Why should I use Caching Framework instead of a hashtable?

A. There are two main problems with using a hashtable:

- How to keep data cached in the middle-tier in sync with the database (or any other data source)?

  This data can become inconsistent with the database as well as across middle-tiers.

- How to ensure that the hashtable size does not grow too big?

  Caching Framework solves both the above issues through features such as distributed caching support, database driven invalidation, and cache memory management. In addition it provides other useful features such as preloading support, and data partitioning support.

Q. Is it possible to use Caching Framework like a hashtable without providing a loader class?

A. Yes. However, you still need to provide a dummy loader class. This use of Caching Framework is not recommended as you miss out on some administration features. If there is a strong requirement, contact the OA Framework team.

Q. Can Caching Framework work without the seed data?

A. Yes. It is possible to create and use a component cache with just the API calls. Once again, contact the OA Framework team if you need to do this.

Q. How does the distributed caching work?

A. By default, the distributed caching works by sending the invalidation message to other JVMs. The list of machines is maintained in the database. As each JVM starts, it makes an entry into this list (if it is not already present). When the other JVMs receive this invalidation message, they mark the object invalid if it is present. When the object is requested in the other JVMs, it is loaded from the database at that time. Also, it is possible to send the whole object across to the other JVMs using the appropriate `put(..)` APIs. This is not recommended as it introduces network overhead. Please refer to the API documentation for details.

Q. Which APIs are distributed?

A. Only the `invalidate(...)` and `put(...)` APIs are distributed. For example, the `get(...)` API does not automatically load the object into the cache.

Q. Is preloading distributed?

A. Preloading is done in each of the JVMs. Each JVM should preload the objects returned by the `bulkload` API.

Q. When is Time to Live or Idle Time used?

A. It is useful for the data, where it is acceptable, to have some maximum delay before the updates or changes are seen across all the JVMs. The data is guaranteed not to be older then the specified time. You do not need to declare the component distributed or use database-driven invalidation, thus avoiding network overhead.

Q. What is the difference between Time to Live and Idle Time?

A. Both are durations of time after which the object in the cache is marked invalid. The difference is in the duration starting moment. Time to Live starts from the time the object was brought into the cache. Idle Time starts from the last access of the object.

Chapter 7: Testing and Debugging

Overview

- JDeveloper Debugging
  - Setting Breakpoints in Library Classes
  - Reading an Exception Stack Trace
  - Setting an Exception Breakpoint
  - Reading Call Stack and Data at Breakpoint
- Adding Debug Classes to the Classpath of the Release 12.2 File System Structure
- Remote Debugging w/ Apache Installation
- Examining Page Content
- Enabling and Examining the Validation Failure Message Log Output

JDeveloper Debugging

Tip: For a detailed introduction to debugging and all of the techniques described here, see the Debugging Lab in the OA Framework ToolBox Tutorial.

Setting Breakpoints in Library Classes

If you want to set a breakpoint in an OA Framework class, or any underlying class in the technology stack, follow these steps:

1. Select the JDeveloper menu option **Search > Go to Java Class**.
2. Specify the fully qualified class to open the source. For example, if you want to add a breakpoint to `OAPageLayoutBean`, you must enter `oracle.apps.fnd.framework.webui.beans.OAPageLayoutBean`.

   Note: Your project libraries must include the source for this to work.

3. Use **Search > Go to Line Number** if you know what line number you want to debug, then toggle a breakpoint on that line.

Reading an Exception Stack Trace

When you get an exception stack trace, look for the content immediately following `## Detail 0 ##` as shown below (although the OA Framework exception stack traces are large, this is where you'll find the relevant an exception). In this example, the exception that you want to focus on is the `oracle.jbo.SQLStmtException`.

```
## Detail 0 ##
   oracle.jbo.SQLStmtException: JBO-27122: SQL error during statement preparation.
   ...
   void oracle.jbo.server.QueryCollection.executeQuery(java.lang.Object[], int)
     QueryCollection.java:608
    void oracle.jbo.server.ViewObjectImpl.executeQueryForCollection(java.lang.Object,    java.lang.Object[], int)
     ViewObjectImpl.java:2650
    void oracle.apps.fnd.framework.server.OAViewObjectImpl.executeQueryForCollection(java.lang.Object,    java.lang.Object[], int)
     OAViewObjectImpl.java:1790
    void oracle.jbo.server.ViewRowSetImpl.execute(boolean, boolean)
     ViewRowSetImpl.java:523
```

Setting an Exception Breakpoint

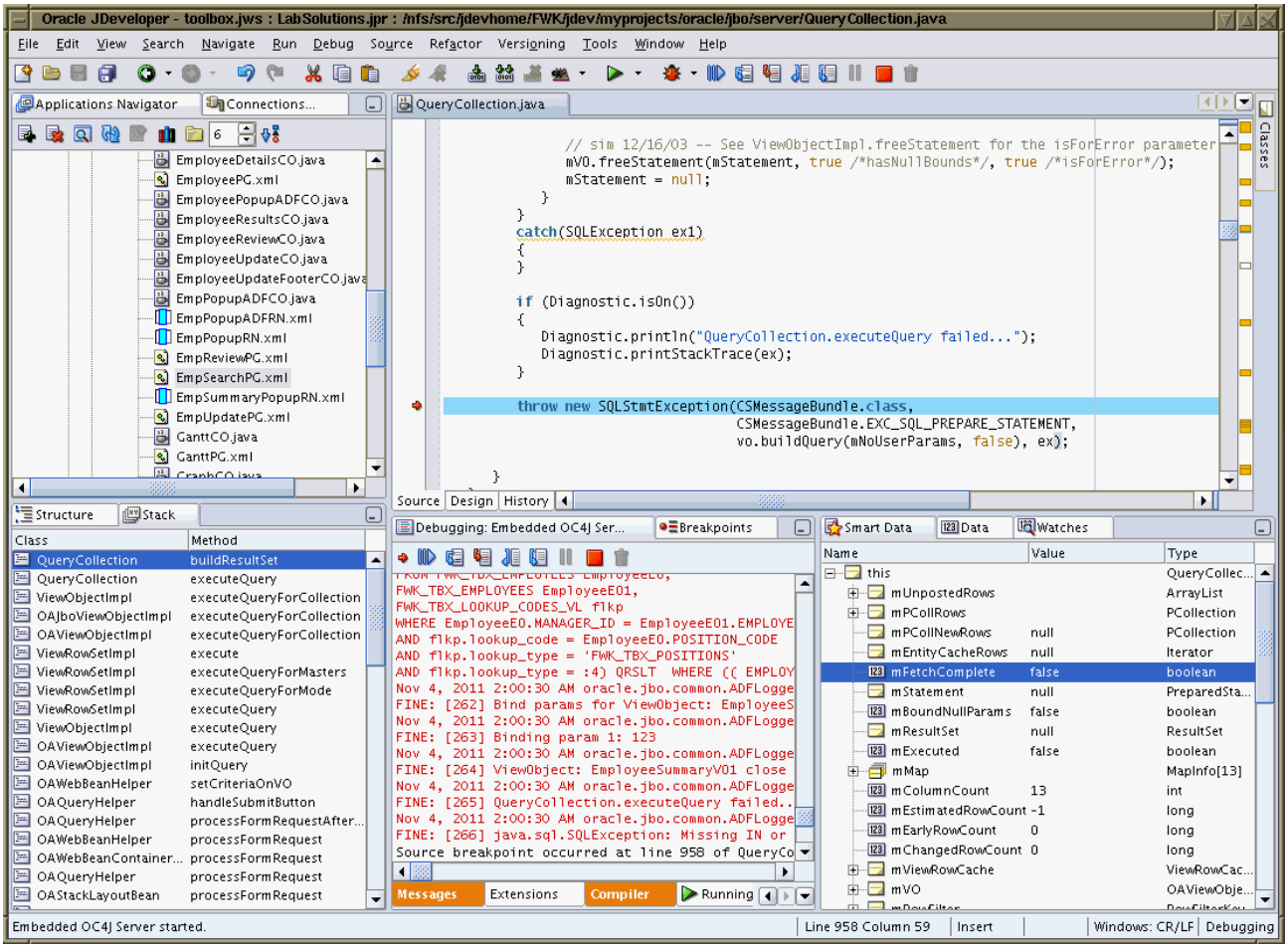
Once you've identified an exception to debug, you need to set an exception breakpoint as described in these steps:

1. Run your module in debug mode.
2. Before you get to the code that is throwing the exception you want to debug, go to the JDeveloper menu option **View > Debug Windows > Breakpoints** (the Breakpoints window automatically comes up when you start debugging).
3. Right click on the Breakpoints window and select **New Breakpoint** option.
4. When the Breakpoints wizard appears, set the **Breakpoint Type** to `Exception`.
5. Specify the **Exception Class** using a fully qualified class name. In this example, you would enter `oracle.jbo.SQLStmtException`.
6. Continue exercising your code until you get the point where the exception is thrown.

Reading Call Stack and Data at Breakpoint

When you reach the point in your code where a breakpoint has been recorded, JDeveloper will stop and highlight the associated code line (in this example, we're showing were an exception is thrown for the `oracle.jbo.SQLStmtException` breakpoint):

Figure 1: Example showing JDeveloper state when an exception breakpoint is reached,



Warning: This works only if your JDeveloper project library has the source available for the class that throws the exception. For instance, JDBC source is NOT included in an OA Framework project so you would have to put a breakpoint at a different layer in the reported in exception stack. Also the class and source files have to be in synch for the breakpoints to work correctly.

Call Stack

In the **Stack** window above, the call stack is shown with the class and method. You can export the call stack to a text file by right clicking on the stack window and choosing the **Export** option.

Data

In **Data** window (in the "this" folder), the current variable values and types are shown. You can right-click on the Data window, and customize the properties to show the object address. To speed up debugging performance, it is often useful to hide the Static fields folder and close the Smart Data and Watches windows.

Adding Debug Classes to the Classpath of the Release 12.2 File System Structure

Background

In Oracle E-Business Suite Release 12, `<JAVA_TOP>` points to a directory that contains a product team's complete set of compiled java code and is present in `<COMMON_TOP>/java/classes`. Oracle E-Business Suite Release 12 applications such as oacore and oafm access a product team's java code from `<JAVA_TOP>` by simply setting the `library path` property in the OC4J Application descriptor **orion-application.xml**. Any directory or JAR file specified in the `library path` property is automatically added to the classpath of the application. Oracle E-Business Suite applications such as oacore and oafm have this property set as follows in their **orion-application.xml** file:

```
<library path="<JAVA_TOP>" />
```

Beginning in Release 12.2, Oracle E-Business Suite applications run on an Oracle WebLogic Server 11g container. In Oracle WebLogic Server 10.3, however, there is no such concept of setting a property to declare a library path, as in OC4J. Instead, the best way to add `JAVA_TOP` to the classpath of an application is to reference a shared library.

In Oracle E-Business Suite Release 12.2, product teams use **adadmin** to generate a JAR file of their product's Java code present under `<JAVA_TOP>/oracle/apps/<product name>` and then copy it to `$JAVA_TOP/oracle/apps/fnd/jar`. A shared library (Web Application) is then deployed, containing a signed JAR file from each product team in the manifest. All Oracle E-Business Suite applications can then reference this shared library. For additional information about **adadmin**, refer to the chapter "Applications DBA System Maintenance Tasks and Tools" in the *Oracle E-Business Suite Maintenance Guide*.

Debug Classes

During the development phase, debug classes are necessary to help product teams develop and test their java code. Since the java code is now in an archived format (as signed JAR files generated through **adadmin**), a product team must load their debug classes into the classpath before loading their original product JAR file.

The simplest way to achieve this is to copy the debug class files to the `<OA_HTML>/WEB-INF/classes` directory. Oracle WebLogic Server loads the classes in the `WEB-INF/classes` directory before loading the class files present in the product JAR files (which are loaded via the shared library).

However, since product JAR files are signed (by the jarsigner tool), Oracle WebLogic Server will throw `java.lang.SecurityException` when it tries to load some classes from the signed JAR files and some classes from the `WEB-INF/classes` directory, as shown in this example:

```
weblogic.application.ModuleException:  class "oracle.apps.fnd.security.AppsServletFilter"'s
```

```
  signer information does not match signer information of other classes in the same package
 at weblogic.servlet.internal.WebAppModule.startContexts(WebAppModule.java:1512)
 at weblogic.servlet.internal.WebAppModule.start(WebAppModule.java:482)
 at weblogic.application.internal.flow.ModuleStateDriver$3.next(ModuleStateDriver.java:425)
 at weblogic.application.utils.StateMachineDriver.nextState(StateMachineDriver.java:52)
 at weblogic.application.internal.flow.ModuleStateDriver.start(ModuleStateDriver.java:119)
Truncated. see log file for complete stacktrace
```

To avoid having Oracle WebLogic Server throw an exception, perform one of the following options, listed in order of preference.

Option A: Disable Jarsigning Verification

Oracle WebLogic Server classloader utilities check for the size and signature of JAR files. So even if you create a debug JAR file and sign it with the same signature as your main product JAR file, Oracle WebLogic Server throws an exception.

To avoid this:

- Set the java property `-Dweblogic.classloader.noJarSigners=true` in **startWeblogic.sh** to disable the jarsign verification. In production mode, the jarsign verification is enabled.
- Copy the debug class files along with the package structure to `<OA_HTML>/WEB-INF/classes` or create a JAR file of debug classes and copy it to `<OA_HTML>/WEB-INF/lib`.
- Bounce the Oracle E-Business Suite middle tier using the command:

  ```
  $ADMIN_SCRIPT_HOME/adoacorectl.sh stop/start
  ```

Option B: Remove Jarsigning Information from the JAR file

- Make a backup of your product's original JAR file present in `$JAVA_TOP/oracle/apps/fnd/jar`.
- Run the following **ad** utility to remove the signing information from your product JAR file:

  ```
  adjava -mx512m -nojit oracle.apps.ad.jri.admetarm -inputFile <backup of original jar> -outputFile <newjar>
  ```

- Copy the debug class files along with the package structure to `$OA_HTML/WEB-INF/classes` or create a JAR file of debug classes and copy it to `$OA_HTML/WEB-INF/lib`.
- Bounce the Oracle E-Business Suite middle tier using the command:

  ```
  $ADMIN_SCRIPT_HOME/adoacorectl.sh stop/start
  ```

- Restore your product's original JAR file after java code testing and debugging is complete.

Option C: Regenerate the Product JAR File

- Make a backup of your product's original JAR file present in `$JAVA_TOP/oracle/apps/fnd/jar`.
- Add the debug class files or make changes to the required class files under the $JAVA_TOP location.
- Run **adadmin** to regenerate the product JAR file.
- The product JAR file is updated with your modified class files.
- Bounce the Oracle E-Business Suite middle tier using the command:

  ```
  $ADMIN_SCRIPT_HOME/adoacorectl.sh stop/start
  ```

- Restore your product's original JAR file after java code testing and debugging is complete.

  Note: Sometimes **adadmin** will not update the JAR file if the version of a class file is unchanged. If this happens, you may need to force regenerate the JAR file.

  Note: The version and size of a class file is stored in `<JAVA_TOP>/JRIMETA.DAT`. If you modify a class file in `JAVA_TOP` and run **adadmin** to update the JAR file, `JRIMETA.DAT` may get modified, which makes this option undesirable.

Remote Debugging w/ Apache Installation

Prerequisites

1. Setup Apache. You can obtain the appropriate download from http://otn.oracle.com/
2. Configure the Apache server for the OA Framework. This should include configuring `OA_HTML`, `OA_MEDIA`, `FND_TOP` for the server as well as setting the classpath. This should be a relatively simple task if you have already have a `APPL_TOP` to which you can map a drive. Remember to also update the Application Framework Agent for the user who is going to do the testing; this should point to the Apache server. Test the configuration by launching your Framework application.
3. Setup a Jdeveloper environment which uses the same version of the OA Framework as the Apache. For example, if you are using OA Framework 510 then you should setup a Jdeveloper that points to the 510 directory on the appropriate drive. This is necessary because the source code in Jdeveloper must match the classes on your Apache server. If the source code and classes are not in sync your breakpoints will not work properly.
4. Download the appropriate Java Platform Debugger Architecture (JPDA) from http://java.sun.com/products/jpda/download.html. Install this in a directory that your Apache server can access.

Set up Remote Debugging

Configure Apache for Remote Debugging

1. Comment out the `wrapper.bin` line in the `jserv.properties` file. Add the following new wrapper.bin directive :
   ```
   wrapper.bin=J:NT/<directory_your_jdeveloper_points_to>/jdk/bin/java.exe
   ```
2. Add the following two lines to the jserv.properties file :

   ```
   wrapper.bin.parameters= -Xrunjdwp:transport=dt_socket,server=y,suspend=n,address=4000 -Xdebug -Xnoagent -Djava.compiler=NONE

   wrapper.path=<path_to_the_JPDA_installation>jpda-1.0\bin
   ```

3. Restart the Apache server.
4. View your `error.log` and/or `jserv.log` files to make sure that Apache/Jserv did not have any trouble with starting the JVM.

Configure Jdeveloper for Remote Debugging

Make sure the Apache server is up before executing the following steps.

1. In Jdeveloper edit the properties for the project you will use to remote debug and launch the application:
   a. Select *Configurations -> Debugger -> Remote*.
   b. Select the checkbox labeled *Remote Debugging*.
   c. Select the radio button labeled *Attach to JPDA*.
   d. Set breakpoints in the desired places in the code. Click *OK* to apply the changes. Close the Project properties window.
   e. Right click on your startup JSP (in most cases this is called test.jsp).
   f. Select *debug*. The "*Remote Debugging Dialog*" window will be rendered.
   g. For *Host* enter you Apache server hostname e.g. `<server>.example.com`
   h. For *Port*, enter the port specified under the address option in step 2 of *Configure Apache for Remote Debugging* described above. In this example, for instance, `address=4000`
   i. Select *OK* to launch the application in debug mode.

   If you get into debug mode then you have successfully activated remote debugging for your apache server.

2. **Execute this step only after Apache has been started and Jdeveloper has been set up for remote debugging.**
   Launch Internet Explorer and access your application on the Apache Server, the way you usually do. If everything was set up correctly, and you set the breakpoints in the right places, you should reach your first breakpoint in the course of the execution. From then on you debug the way you normally do through the code.

Examining Page Content

You can produce a UIX XML representation of your page by enabling the uix component tree dump. You can use this feature to:

- Produce UIX files for test cases or to be used with UIX viewing tools.
- If a component does not render as expected, you can use the UIX dump to very that the HTML is valid and complete.

Enable in JDeveloper

To enable this locally in JDeveloper, set the value of the `OADumpUIXTree` cookie to **1** (you can enable this in your **Project Settings Run Options** page by moving the **OADumpUIXTree** from the **Available Options** to the **Selected Options**). The OA Framework writes the UIX component dump output to a file with the name **page_<*session id*>.uix** at `%JDEV_USER_HOME%\myhtml\oa_html\fwk\t*`.

Note: Be certain to clean this directly frequently in your workspace while this is enabled.

Enable In a Deployed System

To enable this in a deployed system, set the profile option **FND: Debug Log Enabled** to **Yes**, then set the profile option **FND: Debug Log Module** to **DUMP_UIX_TREE**. The OA Framework writes the UIX component dump output to a file with the name **page_<*session id*>.uix** at `$HTML_TOP\fwk\t*`.

Tip: A concurrent manager job can be used to periodically clean up this directory.

Examining the Validation Failure Message Log Output

You can use the FND_VALIDATION_LEVEL, FND_FUNCTION_VALIDATION_LEVEL and FRAMEWORK_VALIDATION_LEVEL profile options to enable the accelerated validation mode. You can also override these profiles with corresponding Java Virtual Machine (JVM) settings using any of the following techniques:

- Test JSP
- JDeveloper Project Settings
- Apache / Quik Apache Settings

Note: Currently, the JVM settings are supported only by OA Framework and not by AOLJ/RF. You can rely on this when running in JDeveloper, but not when using Apache/Quik Apache (use the profile options instead).

Test JSP

In your launching test jsp page, set the JVM properties as follows:

```
System.getProperties().put("FND_VALIDATION_LEVEL", "ERROR");
System.getProperties().put("FND_FUNCTION_VALIDATION_LEVEL", "ERROR");
System.getProperties().put("FRAMEWORK_VALIDATION_LEVEL", "ERROR");
```

JDeveloper Project Settings

Add the following **-Dparameter** entries to your project's JVM options (open your Project Settings window and navigate to Configurations > Development or Production > Runner):

```
-DFND_VALIDATION_LEVEL=ERROR -DFND_FUNCTION_VALIDATION_LEVEL=ERROR
  -DFRAMEWORK_VALIDATION_LEVEL=ERROR
```

When debugging in JDeveloper, note that any validation failure messages can also be seen in the JDeveloper Message Console window.

Apache / Quik Apache Settings

For an Apache installation, modify the **jserv.properties** file and add the following (or, make the corresponding change to your Quik Apache configuration):

wrapper.bin.parameters=-DFND_VALIDATION_LEVEL=ERROR
wrapper.bin.parameters=-DFND_FUNCTION_VALIDATION_LEVEL=ERROR
wrapper.bin.parameters=-DFRAMEWORK_VALIDATION_LEVEL=ERROR

When debugging issues in Apache, please note that logging needs to be enabled to view the validation failure messages. The logging level can be set to **UNEXPECTED** (the highest level) to reduce the logging output. See the Logging document for information on enabling logging.

Note that logging can also be configured at the JVM level as well. For example, the following shows how to enable logging to a flat file:

-DFND_VALIDATION_LEVEL=ERROR
-DFND_FUNCTION_VALIDATION_LEVEL=ERROR
-DFRAMEWORK_VALIDATION_LEVEL=ERROR
-DAFLOG_ENABLED=TRUE
-DAFLOG_LEVEL=UNEXPECTED
-DAFLOG_FILENAME=/proddev/MAClogs/proddb11510/proddb11510_16AprLOG.txt

Reviewing the Validation Failure Message Log Output

Once you enable accelerated validation, you can review the validation failure message log output for errors. To find validation-related issues in the log output, search for **MACCHECK**.

- URL issues are identified with the label **MACCHECKURL:**
- Form parameter issues are identified with the label **MACCHECK:**
- Function validation issues are identified with the label **MACLITECHECK:**

- "Referer URL" is the page where the link was rendered (if available in the HTTP header). Note that "Current URL" will be different from "Incoming URL" in the case of a JSP forward.

Discovering Page, Technology Stack and Session Information

Overview

This document describes the two tools that you can use to obtain information about a page's definition, the current session and the technology stack.

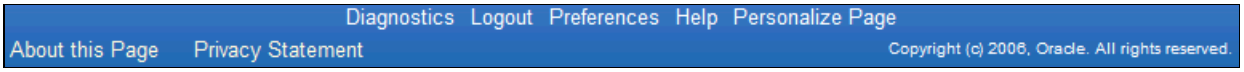
- "About" Page
- OAInfo.jsp

"About" Page

If you want to know information about the current page's definition (particularly useful if you plan to personalize or extend it) and the current page's context information, select the page's **About this page** link as shown in Figure 1 below.

Tip: The **About this page** link renders automatically for all OA Framework pages if the Diagnostics or admin-level Personalization feature (as of Release 12.1.3) is enabled. See the documentation for the FND: Diagnostics / FND_DIAGNOSTICS and the Personalize Self-Service Defn / FND_CUSTOM_OA_DEFINTION profile options for additional information.

Figure 1: "About this page" link in a page footer



The "About" page renders the following subtabs that display information about the page, the page's context, and the environment:

- Page
- Personalization
- Page Context
- Technology Components
- Java System Properties
- Profiles
- Patches

Page Subtab

The **Page** subtab, as shown in Figure 2 provides detailed information about the Page Definition, Flexfield References and Translatable Items of the page.

- **Page Definition HGrid** - displays the web bean structure of the page along with the controller, application module, view objects and view attributes associated with the web bean.
  - In the View Object column, if you select a linked value, you can display the substitution (if any), version, query statement, entity objects and attributes of the view object, as shown in Figure 3.
  - **Business Component References Details** - expand this heading to display full package information about the application modules, retained application modules, view objects and controllers referenced by the page, as shown in Figure 4.
- **Flexfield References table** - displays the segment detail information of the currently referenced flexfields (if any) in the page and warnings about non-supported Oracle E-Business Suite Flexfields features and developer mode errors (if any). Select **Show All Details** to view all the flexfield references, as shown in Figure 5.
- **Translatable Items table** - expand this heading to display the details of the translatable items on the page, as shown in Figure 6.

Note: Figures 2 through 6 illustrate the information displayed in the Page subtab and are captured by running the ItemSearchPG page of the LabSolutions project in the Toolbox Tutorial. After performing a search based on Item Number, select the resulting Item Number link to display the Item Details page. Then select the **About this Page** link to display the Page subtab content.

Figure 2: "About" page content

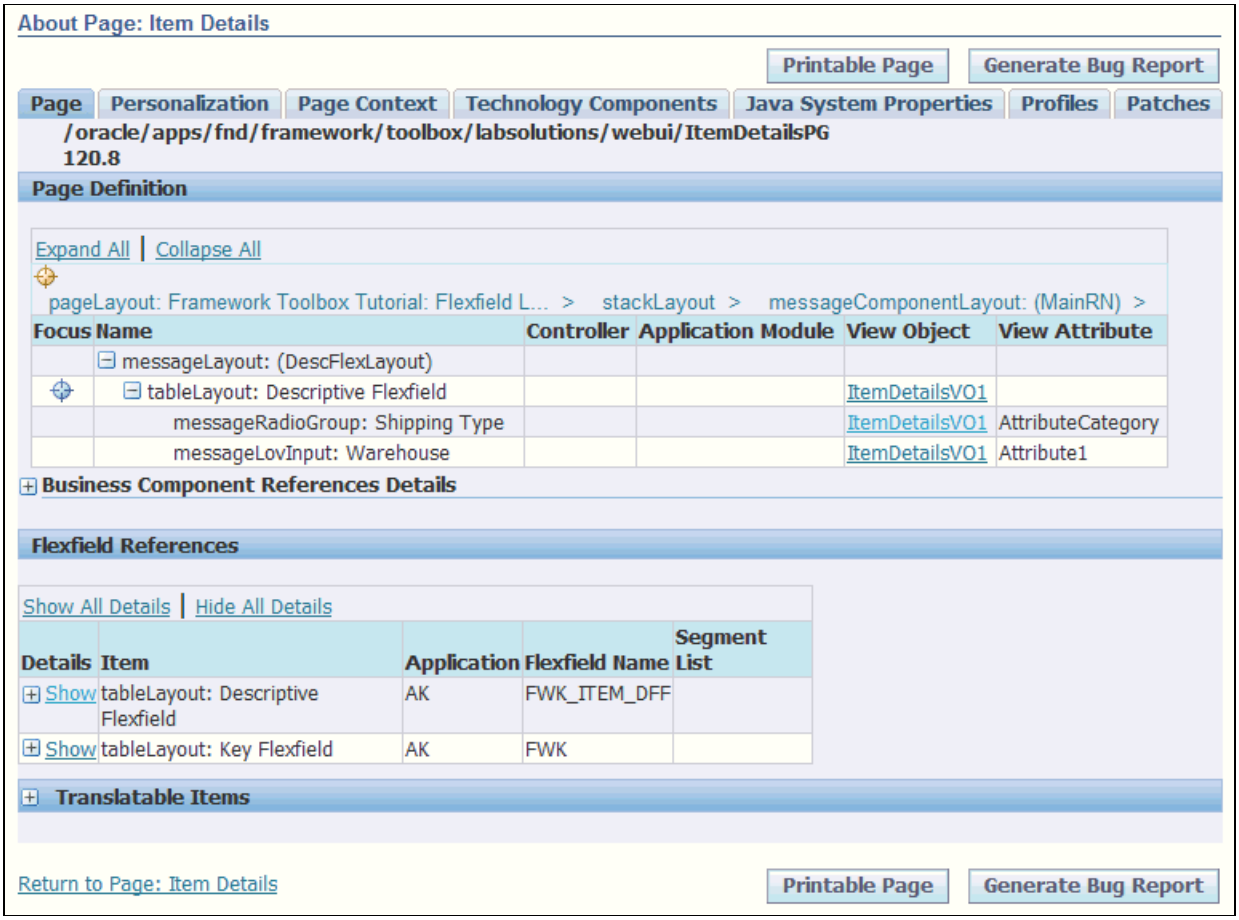


Figure 3: "About View Objects" content in the "About" page **Page** subtab

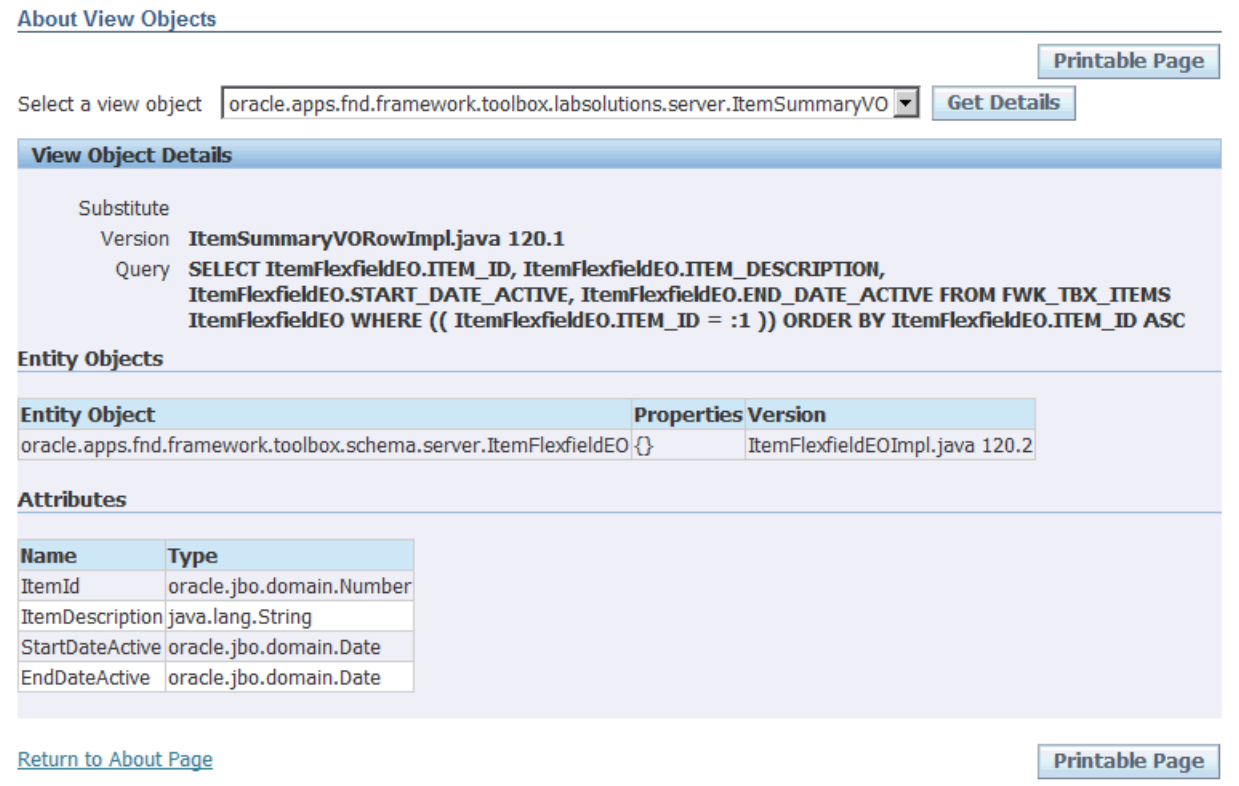
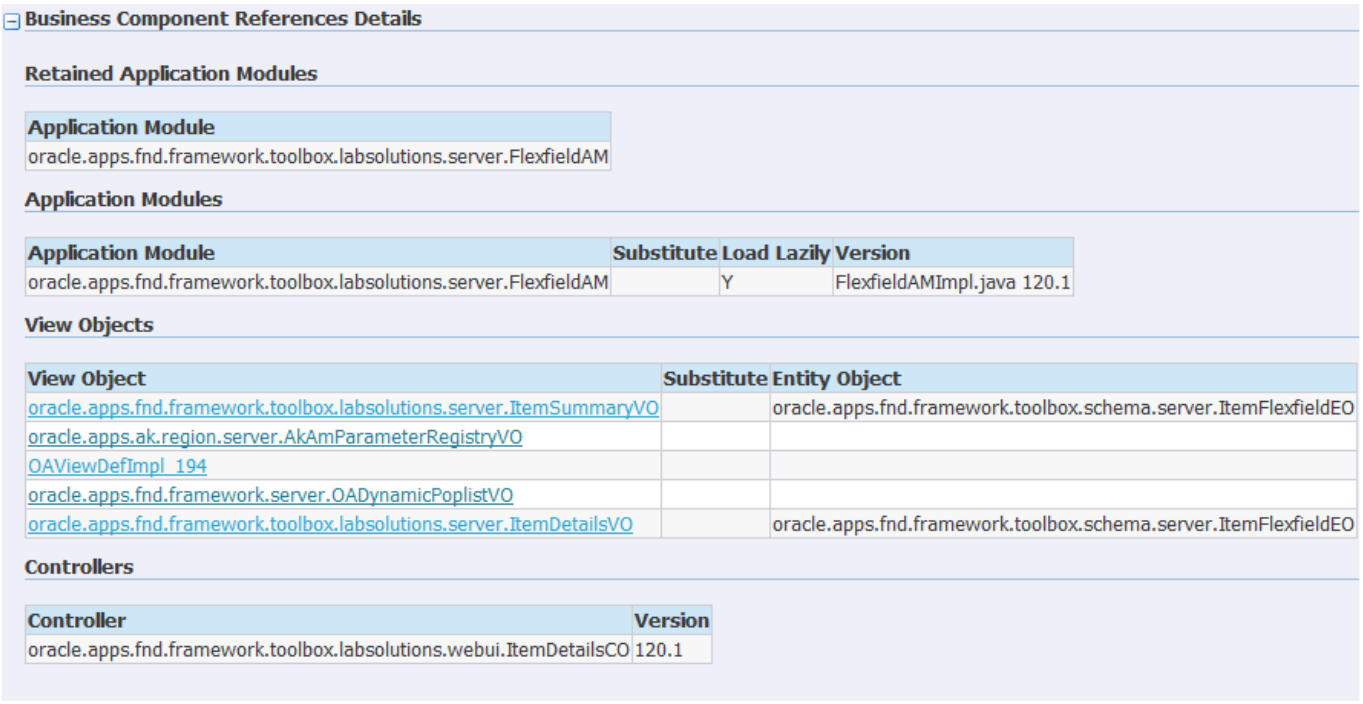


Figure 4: Business Component Reference Details in the "About" page **Page** subtab



Business Component References Details

Retained Application Modules

Application Module
oracle.apps.fnd.framework.toolbox.labsolutions.server.FlexfieldAM

Application Modules

Application Module	Substitute	Load Lazily	Version
oracle.apps.fnd.framework.toolbox.labsolutions.server.FlexfieldAM		Y	FlexfieldAMImpl.java 120.1

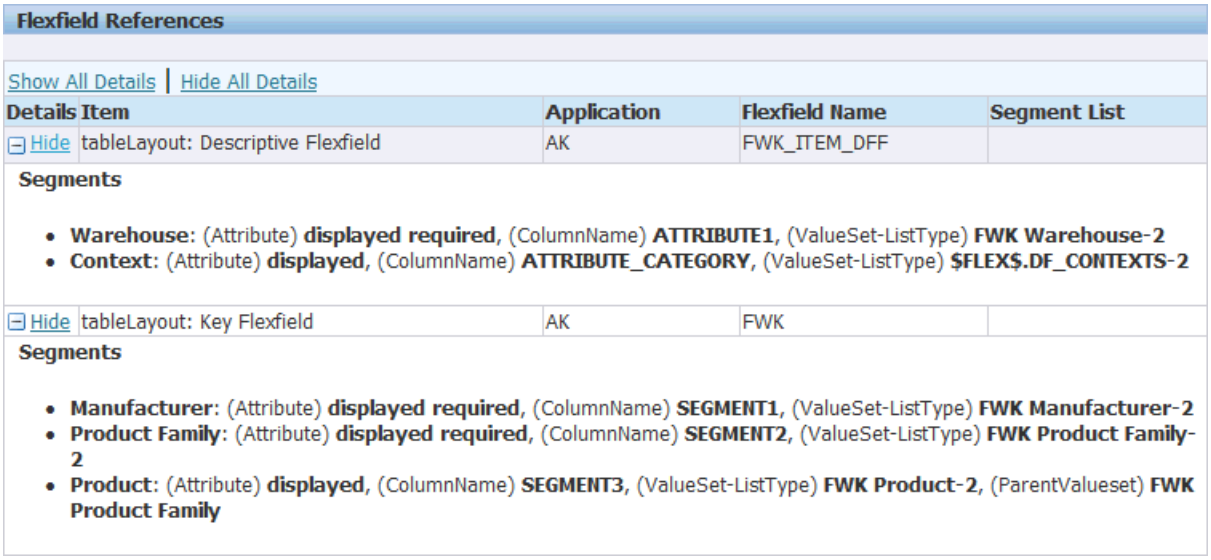
View Objects

View Object	Substitute	Entity Object
oracle.apps.fnd.framework.toolbox.labsolutions.server.ItemSummaryVO		oracle.apps.fnd.framework.toolbox.schema.server.ItemFlexfieldEO
oracle.apps.ak.region.server.AkAmParameterRegistryVO		
OAViewDefImpl_194		
oracle.apps.fnd.framework.server.OADynamicPoplistVO		
oracle.apps.fnd.framework.toolbox.labsolutions.server.ItemDetailsVO		oracle.apps.fnd.framework.toolbox.schema.server.ItemFlexfieldEO

Controllers

Controller	Version
oracle.apps.fnd.framework.toolbox.labsolutions.webui.ItemDetailsCO	120.1

Figure 5: Flexfield References in the "About" page **Page** subtab



Flexfield References

Show All Details | Hide All Details

Details	Item	Application	Flexfield Name	Segment List
Hide	tableLayout: Descriptive Flexfield	AK	FWK_ITEM_DFF	

Segments

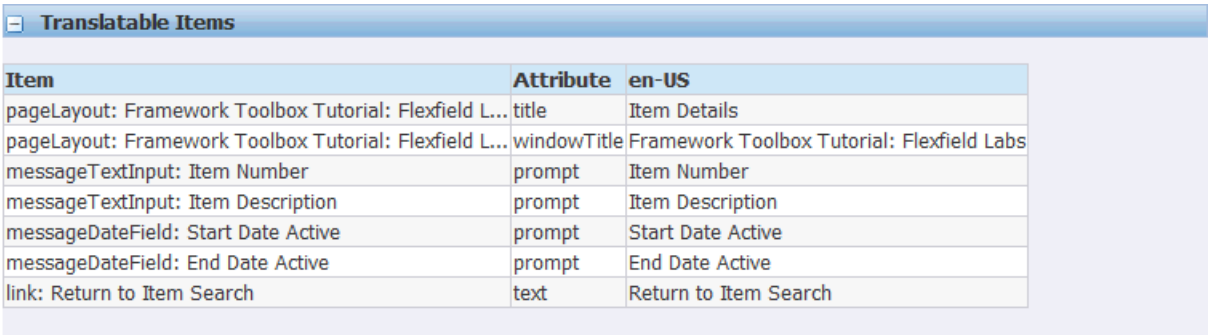
- **Warehouse**: (Attribute) **displayed required**, (ColumnName) **ATTRIBUTE1**, (ValueSet-ListType) **FWK Warehouse-2**
- **Context**: (Attribute) **displayed**, (ColumnName) **ATTRIBUTE_CATEGORY**, (ValueSet-ListType) **$FLEX$.DF_CONTEXTS-2**

Hide	tableLayout: Key Flexfield	AK	FWK	

Segments

- **Manufacturer**: (Attribute) **displayed required**, (ColumnName) **SEGMENT1**, (ValueSet-ListType) **FWK Manufacturer-2**
- **Product Family**: (Attribute) **displayed required**, (ColumnName) **SEGMENT2**, (ValueSet-ListType) **FWK Product Family-2**
- **Product**: (Attribute) **displayed**, (ColumnName) **SEGMENT3**, (ValueSet-ListType) **FWK Product-2**, (ParentValueset) **FWK Product Family**

Figure 6: Translatable Items in the "About" page **Page** subtab



Translatable Items

Item	Attribute	en-US
pageLayout: Framework Toolbox Tutorial: Flexfield L...	title	Item Details
pageLayout: Framework Toolbox Tutorial: Flexfield L...	windowTitle	Framework Toolbox Tutorial: Flexfield Labs
messageTextInput: Item Number	prompt	Item Number
messageTextInput: Item Description	prompt	Item Description
messageDateField: Start Date Active	prompt	Start Date Active
messageDateField: End Date Active	prompt	End Date Active
link: Return to Item Search	text	Return to Item Search

Personalization Subtab

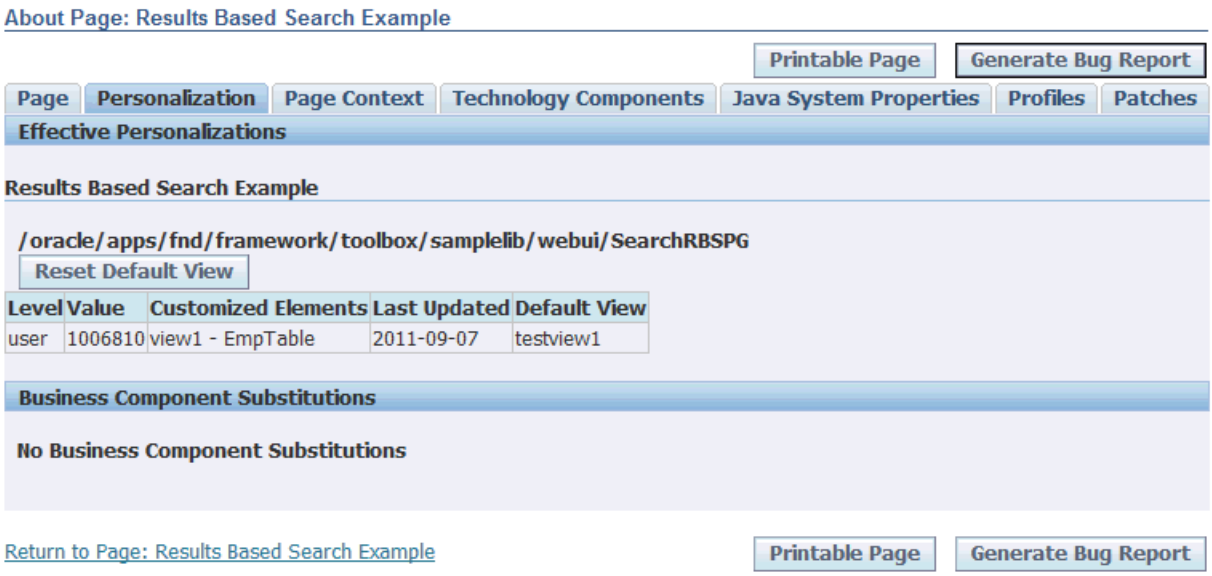
The **Personalization** subtab identifies any personalizations that are in effect for the current personalization context, as shown in Figure 7. You can use the Manage Personalization Levels button on this page to recover from errors that result from a personalization.

For example, suppose you create a personalization for the Hello World page and when you select **Return to Application** from the Personalization UI, an **Error Page** results because your personalization is invalid. To recover from this error, select the **About previous Page** link in the **Error Page**. In the "About" page that displays, select the **Personalization** subtab, then select the **Manage Personalization Levels** button to navigate to the Manage Personalization Levels page, where you can delete the offending personalization. Once you delete the personalization, you can return to the "About" page and then navigate back to your original page without error.

Similarly, you can create a user personalized view, and set that view as the default view for the page. If that default view gets corrupted (for example, if Oracle ships a change to the base page that may render the personalized view invalid), an **Error Page** will result when you run that page. To correct the problem, you need to reset the page so that the default view originally set by the system administrator renders. Select the **About previous Page** link in the **Error Page**. In the "About" page that displays, select the **Personalization** subtab, then select **Reset Default View**. This resets the page to display the default view originally set by the system administrator. If no default view was set by the system administrator, the base page renders.This allows you to run the page again so you can then navigate to the **Personalize Views** page to either delete or correct the offending view.

The **Personalization** subtab also lists any BC4J component substitutions for the page.

Figure 7: Effective Personalizations shown in the Personalization Subtab

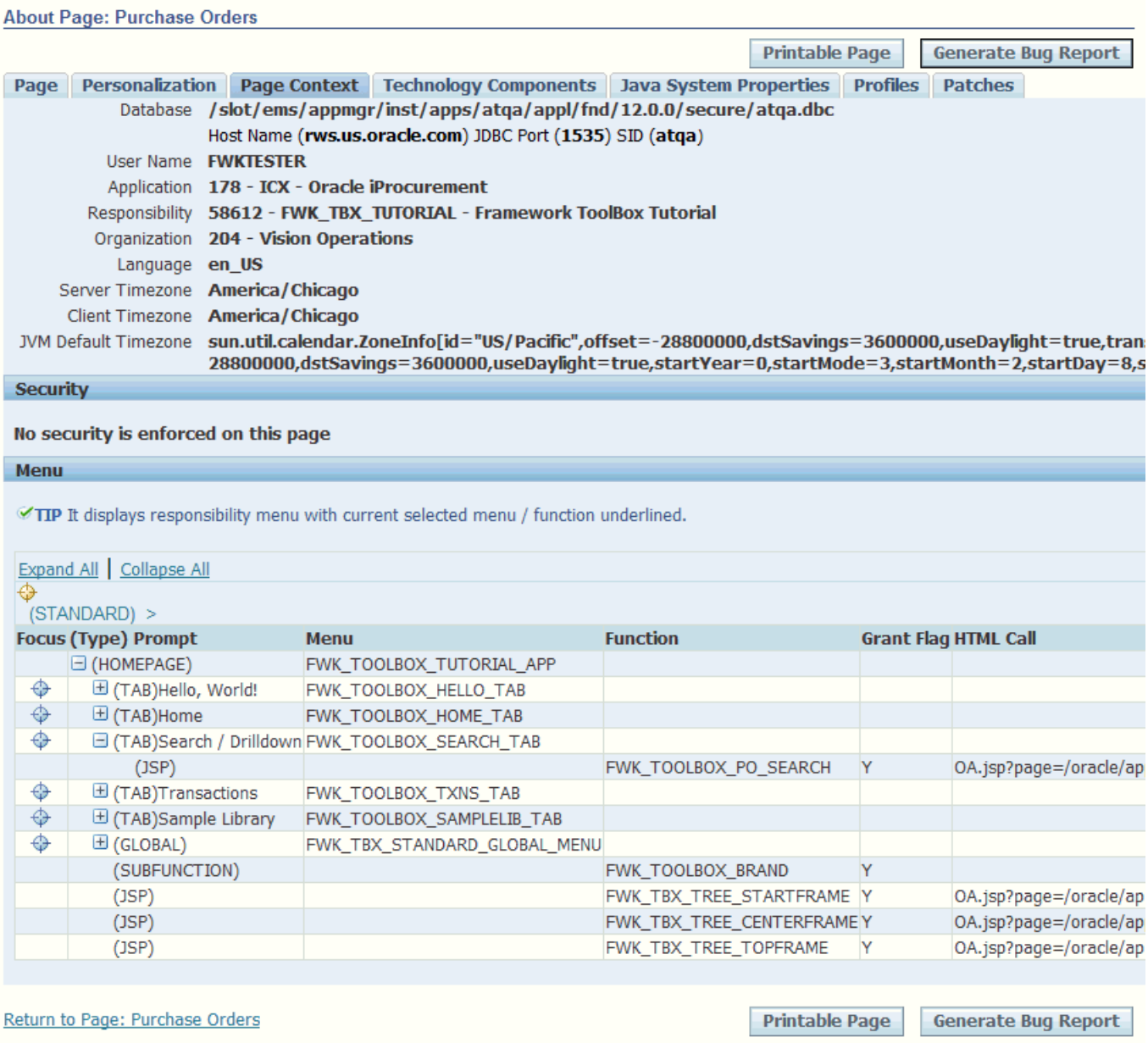


Page Context Subtab

The **Page Context** subtab displays the context information that the current page is running against, such as the database, application, responsibility, and so on. It also lists any security rules being enforced and displays in an HGrid, the current responsibility menu. The currently selected menu/function is underlined in the HGrid.

The "About" page **Page Context** subtab shown in Figure 9 appears after selecting the **Search & Drilldown** menu function from **test_fwktutorial.jsp** of **Tutorial.jpr**.

Figure 9: "About" page **Page Context** subtab



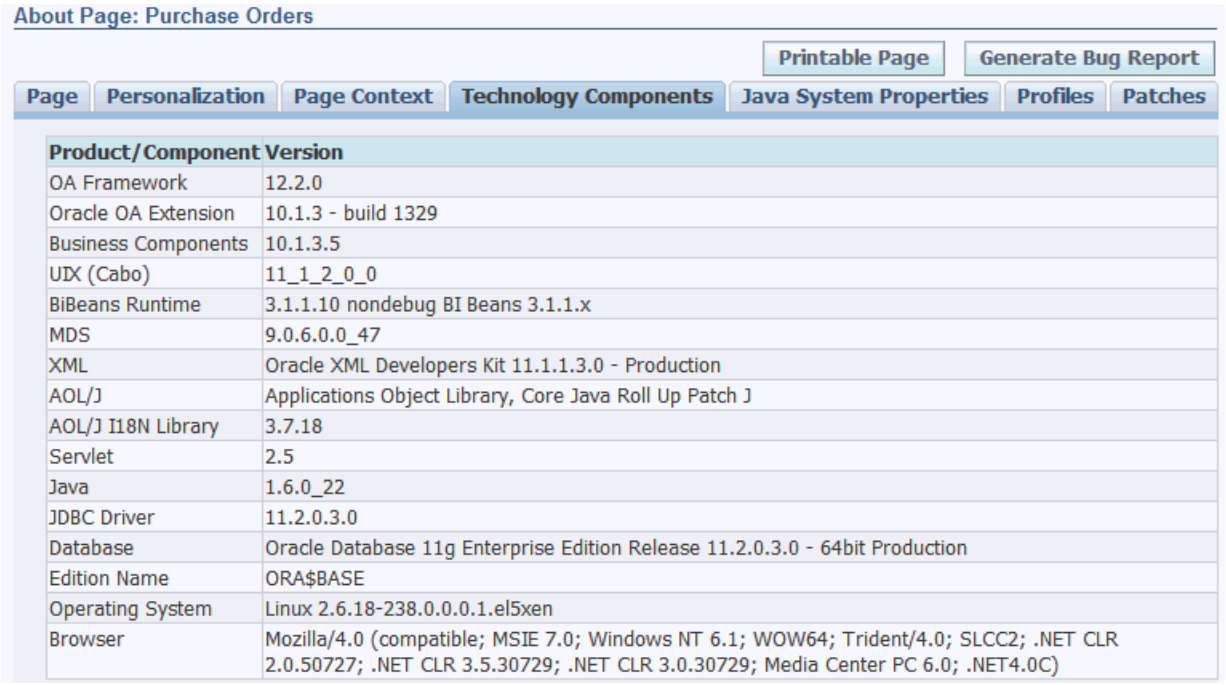
Technology Components Subtab

The **Technology Components** subtab shows the versions of the major products and components of the current technology stack, as shown in Figure 10.

Note: As of Release 12.2, the AOL/J I18N Library information automatically displays in the **Technology Components** subtab if the I18N Library supports the version information.

Note: As of Release 12.2, the database Edition Name displays in the **Technology Components** subtab.

Figure 10: "About" page **Technology Components** subtab

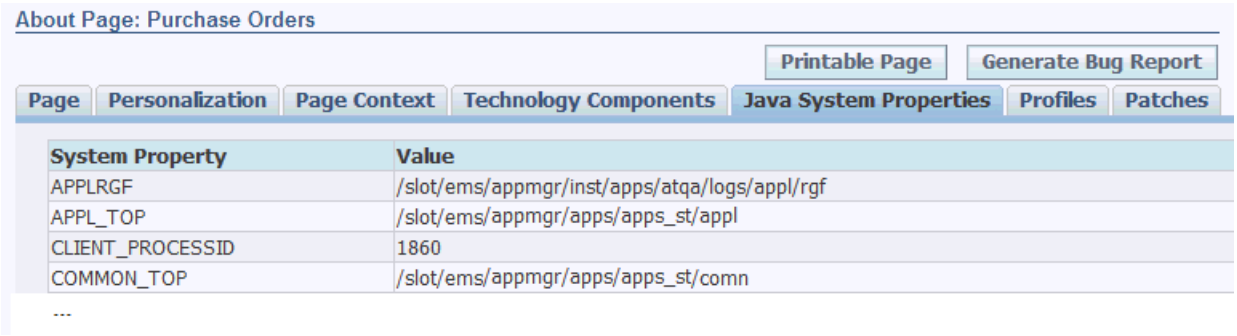


Product/Component	Version
OA Framework	12.2.0
Oracle OA Extension	10.1.3 - build 1329
Business Components	10.1.3.5
UIX (Cabo)	11_1_2_0_0
BiBeans Runtime	3.1.1.10 nondebug BI Beans 3.1.1.x
MDS	9.0.6.0.0_47
XML	Oracle XML Developers Kit 11.1.1.3.0 - Production
AOL/J	Applications Object Library, Core Java Roll Up Patch J
AOL/J I18N Library	3.7.18
Servlet	2.5
Java	1.6.0_22
JDBC Driver	11.2.0.3.0
Database	Oracle Database 11g Enterprise Edition Release 11.2.0.3.0 - 64bit Production
Edition Name	ORA$BASE
Operating System	Linux 2.6.18-238.0.0.0.1.el5xen
Browser	Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; .NET4.0C)

Java System Properties Subtab

The **Java System Properties** subtab lists all java system properties in alphabetical order, as shown in Figure 11.

Figure 11: "About" page **Java System Properties** subtab



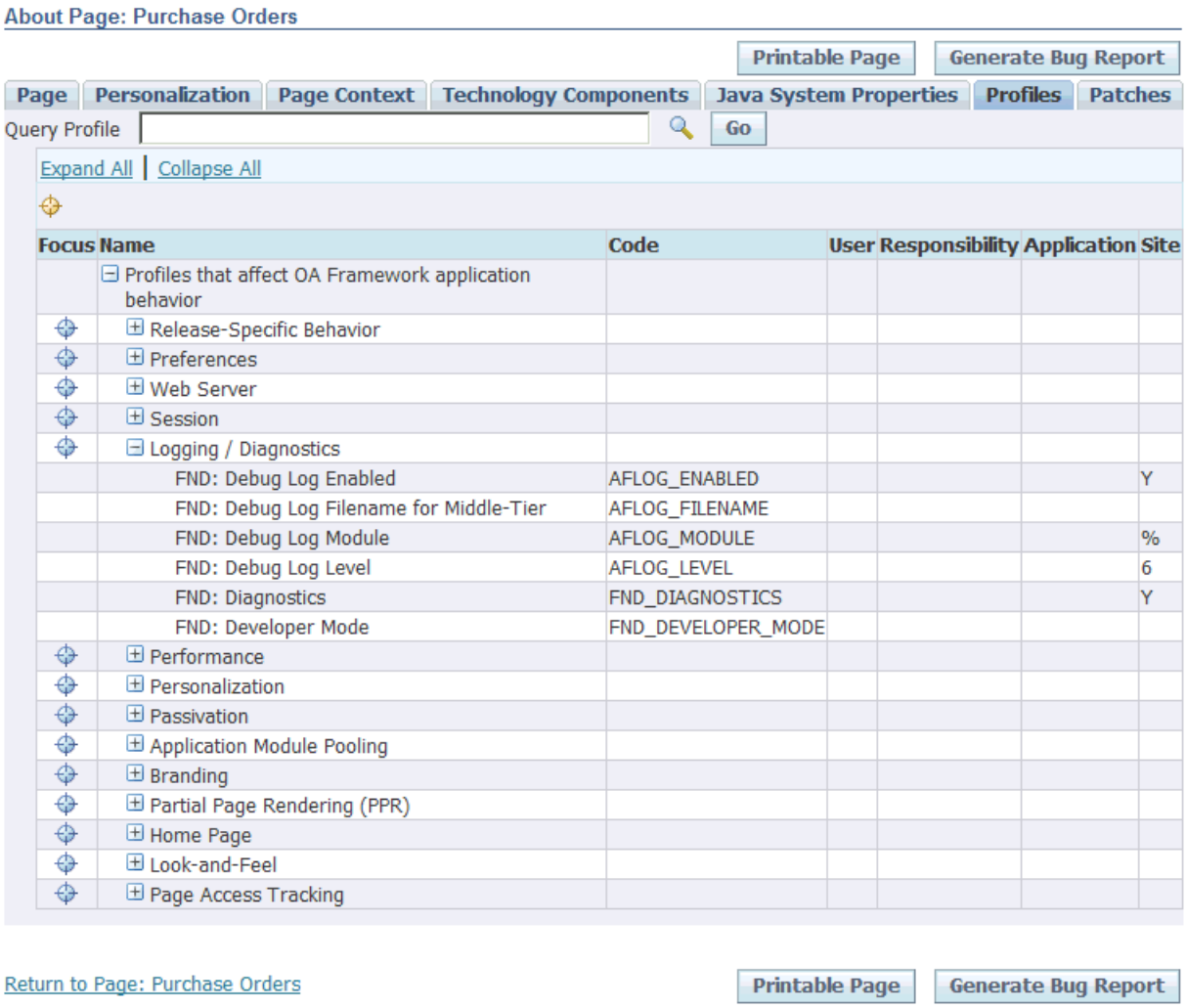
System Property	Value
APPLRGF	/slot/ems/appmgr/inst/apps/atqa/logs/appl/rgf
APPL_TOP	/slot/ems/appmgr/apps/apps_st/appl
CLIENT_PROCESSID	1860
COMMON_TOP	/slot/ems/appmgr/apps/apps_st/comn

...

Profiles Subtab

The **Profiles** subtab, as shown in Figure 12, lists those profiles that affect OA Framework application behavior and the values of those profiles at each level. You may also use the LOV provided to query any other profile.

Figure 12: "About" page **Profiles** subtab

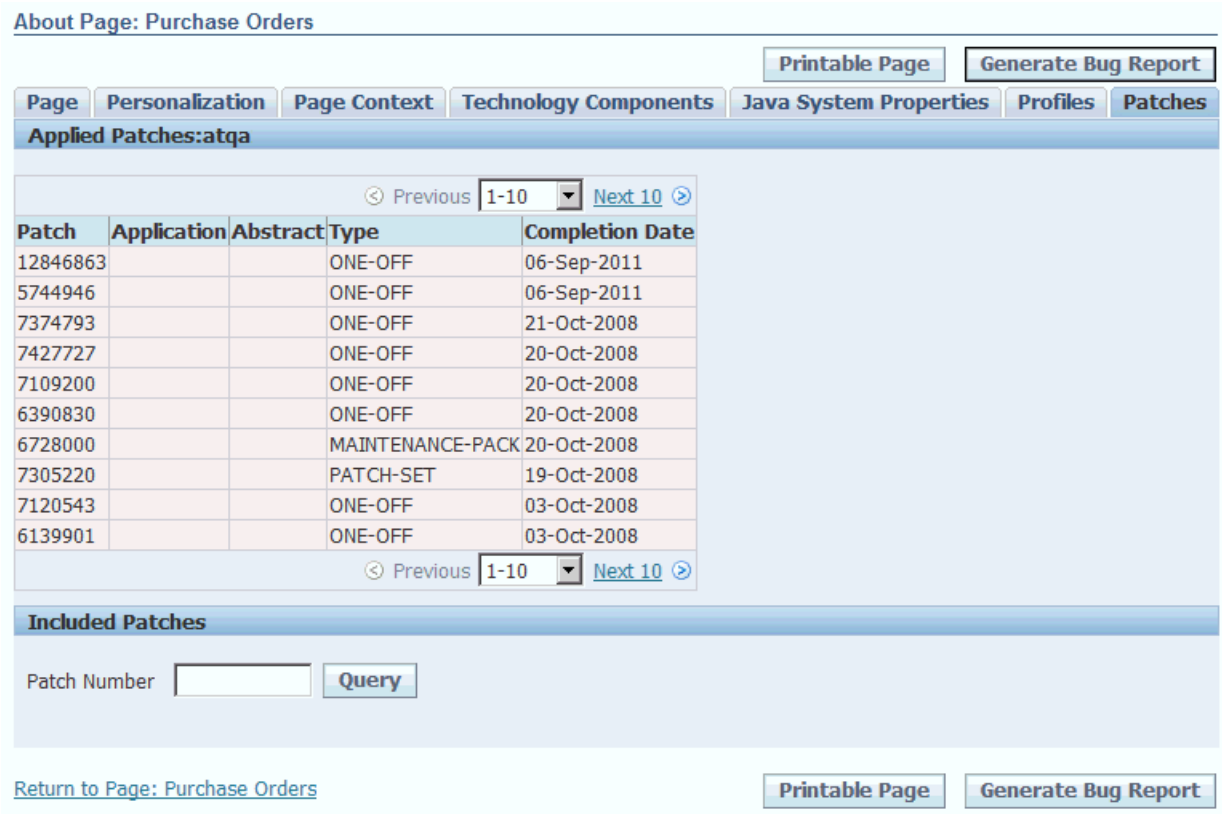


Patches Subtab

The **Patches** subtab, as shown in Figure 13, lists the patches that were applied to the current environment, with the most recently applied patch listed first. If the same patch was applied multiple times, only the most recent application appears. If you know a specific patch number, you can query for that patch number in the **Included Patches** region to determine if the patch was included in an applied patch.

Note: As of Release 12.2, the **Patches** subtab lists only those patches that have completed their cut-over phase in online patching.

Figure 13: "About" page **Patches** subtab



OAInfo.jsp

OAInfo.jsp is no longer supported. For information about a page's definition or context or to view additional information about the current environment, you can select the About this Page link which renders at the bottom of each OA Framework-based page when you enable the FND: Diagnostics / FND_DIAGNOSTICS profile option.

Logging

Overview

The OA Framework logging API is an easy-to-use tool for writing code execution log information to a special table in the database or to a middle-tier file so it can be viewed in a user friendly Diagnostics page. You can also configure logging to append log information directly to the end of an existing screen.

This document is organized as follows:

- How Logging Works
- Enabling Logging
- FND Logging at middle tier node

For complete information about logging, refer to the *Oracle E-Business Suite Maintenance Guide* available on the Applications Release 12 Online Documentation CD.

For complete information on how to use the Oracle Application Manager **Applications Dashboard** to enable logging, view log information and view System Alerts, refer to the chapter called "Monitoring Oracle E-Business Suite with Oracle Applications Manager" in the *Oracle E-Business Suite Maintenance Guide,* available on the Applications Release 12 Online Documentation CD.

How Logging Works

If you want to record code execution information, you must first add OA Framework logging API method calls in the appropriate places in your code. At runtime, if logging is enabled, each message that you opt to write is recorded in a database table (or a flat file if specified).

To view log messages, select the **Diagnostics** global button from any page (this global button is configured as part of the standard global menu added to each page; the display of this menu item is controlled by the profile option FND: Diagnostics (FND_DIAGNOSTICS)).

Note: "retainAM=Y" is added as a parameter to the URL which links the **Diagnostics** global button to the Diagnostics page. As a result, the application module of the page from which a user navigates to the Diagnostics page is retained in the Diagnostics page. Also when a user navigates out of the Diagnostics page using the **Return to Application** link, the application module is retained or released based on the value of the retainAM parameter in the originating page from which the user navigated to the Diagnostics page.

After you select the Diagnostics button to navigate to the **Diagnostics** page, you can select:

- **Show Log** - directs you to the *Oracle Applications Manager* where you can view a "snapshot" of your Oracle E-Business Suite system. For complete information on how to use the pages of the *Oracle Applications Manager*, choose the **Help** button in any *Oracle Applications Manager* page or refer to the chapter called "Monitoring Oracle E-Business Suite with Oracle Applications Manager" in the *Oracle E-Business Suite Maintenance Guide,* available on the Applications Release 12 Online Documentation CD.
- **Show Log on Screen** - allows you to specify a log level and display Java log messages for a particular HTTP Request-Response at the end of the current page. For additional information about this feature, refer to the section titled "Using Logging to Screen" in the chapter called "Logging Features in Oracle E-Business Suite" of the *Oracle E-Business Suite Maintenance Guide,* available on the Applications Release 12 Online Documentation CD.
- **Set Trace Level** - displays the Set Trace page where you can specify the level of information to collect in a database trace. See enabling a database trace for additional information about this feature.
- **Show Pool Monitor** - displays the Application Pool Monitor where you view different aspects of runtime and configuration information about the JVM. See Monitoring the Application Module Pool for additional information about this feature.
- **Oracle Diagnostics** - is a separate link on the Diagnostics page (not a choice on the poplist). If you select this link, you navigate to the Oracle Diagnostics home page. To use Oracle Diagnostics, see the *Oracle E-Business Suite Maintenance Guide,* available on the Applications Release 12 Online Documentation CD.

Performance Considerations

The cost of the logging API calls in your code with logging *turned off* is negligible. OA Framework caches the logging profile options for the current user, so once that setting is cached, you only pay the price of calling the API to determine if logging is enabled before you actually write any diagnostics.

Running with logging turned on *does* have an impact on performance, but users typically do not enable logging unless they are trying to diagnose a system problem.

Logging APIs

The logging methods are published by `oracle.apps.fnd.framework.webui.OAPageContext` and `oracle.apps.fnd.framework.server.OADBTransaction` for client and server-side code access respectively (for example,

if you want to make logging calls in a UI controller, use `OAPageContext`. If you want to make logging calls from an application module, use `OADBTransaction`). The logging methods that you use in these classes are described in the "Using Java" section of the "Logging Features in Oracle E-Business Suite" chapter of the *Oracle E-Business Suite Maintenance Guide,* available on the Applications Release 12 Online Documentation CD.

Log Message Contents

Each log message consists of several component which are fully described in the "Logging Features in Oracle E-Business Suite" chapter of the *Oracle E-Business Suite Maintenance Guide,* available on the Applications Release 12 Online Documentation CD.

You are responsible for writing the text that is inserted into the log table (by passing the text message to the logging APIs). These messages should be as concise and informative as possible.

Module

In addition to the level, the module (fully qualified class name) that is reporting the log message is also recorded. This is used to identify the origin of the message.

Log Level Usage Guidelines

For details on Log Level usage guidelines, refer to the "Logging Features in Oracle E-Business Suite" chapter of the *Oracle E-Business Suite Maintenance Guide,* available on the Applications Release 12 Online Documentation CD.

Warning: Logging has an associated performance cost. Be very careful about low-level logging (for example, procedure and statement level). Also, ALWAYS check if logging is enabled *before* concatenating the strings that form your log message.

Enabling Logging

Although logging behavior is ultimately controlled by a set of profile options, you should use the Oracle Applications Manager Log Setup page to configure your logging behavior. The changes you make to this page automatically update the appropriate log-related profile options. For information on how to use the Log Setup page, refer to the section titled "Setting Up Logging" in the chapter titled "Monitoring Oracle E-Business Suite with Oracle Applications Manager" in the *Oracle E-Business Suite Maintenance Guide,* available on the Applications Release 12 Online Documentation CD.

Profile Option Usage Guidelines

For a deployed module (as opposed to one that is currently under development), only UNEXPECTED errors (those that require administrator attention) should be logged. Also, users, in general, should not be accessing the **Diagnostics** page. The site level profile option value for FND: Diagnostics (FND_DIAGNOSTICS) should be set to **No.**

If a problem is noted at the customer site, then the system administrator should log in as a user with the profile option FND: Diagnostics (FND_DIAGNOSTICS) set to **Yes** to investigate the problem.

FND Logging at middle tier node

Steps to get the FND log messages

1. To enable Logging at the middle tier layer, set the following parameters by the adding following lines to the file:

   $FMW_Home/Oracle_EBS-app1/applications/oacore/APP-INF/wls.properties

   ```
   AFLOG_ENABLED=TRUE
   AFLOG_LEVEL=UNEXPECTED
   AFLOG_MODULE=fnd%
   AFLOG_FILENAME=/tmp/fnd.log
   ```

2. To enable logging at the database layer:

   Login to the application using the 'System Administrator' responsibility and set the following **System Profiles** to as follows at the **Site** level:

Profile Name	Value
FND: Debug Log Enabled	Yes
FND: Debug Log Module	fnd%
FND: Debug Log Level	Statement
FND: Debug Log Filename for Middle-Tier	<*path*>/fnd.log

Note: Make sure you specify the absolute path to your logging directories instead of <path>. Logging settings at the middle tier take precedence over settings at the database layer.

3. Bounce the following servers:

   HTTP Server (using `adapcctl.sh`)
   OC4J Server (using `adoacorectl.sh`)

4. Run the following SQL statement to obtain the fnd_log_messages max sequence:

   ```
   SQL> select max(log_sequence) from fnd_log_messages
   ```

5. Reproduce the problem.

6. Upload the following FND Log information:

   fnd.log

7. Run the following sql statement and upload the result in Microsoft Excel format:

   ```
   SQL> select module, message_text, log_sequence
   from fnd_log_messages
   where module like 'fnd%' and log_sequence
   ```
   > &max(log_sequence) from result-of-step above

Steps to get the log details (i.e. SOP log statements) from customer instance using debug class files

Refer to the Adding Debug Classes to the Classpath of the Release 12.2 File System Structure instructions in the Debugging OA Framework Applications topic.

Inspecting the MDS Repository Content

Overview

This document describes how to query and inspect your base and personalization metadata from the MDS repository using the `JDR_UTILS` package. The migration from AK to MDS occurs during the upgrade to 11.5.9 or by applying an equivalent family-pack or mini-pack patch. Your application will be migrated to MDS if the profile option **FND: Migrated to JRAD** is set to **Yes** at your application level.

Note that the OA Framework supports the use of the `JDR_UTILS` package for debugging purposes only, and not in your production code.

The `JDR_UTILS` package contains the following commonly used APIs:

- `PROCEDURE listContents(p_path VARCHAR2, p_recursive BOOLEAN DEFAULT FALSE)` -- Prints the contents of a package.

  For the non-recursive case, this will list the documents, package files and package directories. For the recursive case, this will list the document, package files and empty package directories (i.e. packages which contain no documents or child packages). In order to differentiate documents from package directories, package directories will end with a '/'.

  Parameters:
  `p_path` -- The path in which to list the documents. To specify the root directory, use '/'.
  `p_recursive` -- If TRUE, recursively lists the contents of sub-directories. Defaults to FALSE.

- `PROCEDURE listCustomizations(p_document VARCHAR2)` -- List the customizations for the specified document.

  Parameters:
  `p_document` - the fully qualified document name, which can represent either a document or package file.

- `PROCEDURE listLanguages(p_document VARCHAR2)` -- Lists the supported languages for the specified document.

  Parameters:
  `p_document` - the fully qualified document name, which can represent either a document or package file.

- `PROCEDURE printDocument(p_document VARCHAR2, p_maxLineSize NUMBER DEFAULT MAX_LINE_SIZE)` -- Prints the contents of an OA Extension document to the console.

  Parameters:
  `p_document` - the fully qualified document name, which can represent either a document or package file.
  `p_maxLineSize` - the maximum size of line. This defaults to 255 which is the maximum allowable size of a line (the 255 limit is a limitation of the `DBMS_OUPUT` package).

  Limitations:
  Documents larger than 1000000 bytes will fail as DBMS_OUPUT's maximum buffer is 1000000 bytes.

- `PROCEDURE printTranslations(p_document VARCHAR2, p_language VARCHAR2, p_maxLineSize NUMBER DEFAULT MAX_LINE_SIZE)` -- Prints the translations for the document in XLIFF format.

  Parameters:
  `p_document` - the fully qualified document name, which can represent either a document or package file.
  `p_language` - the language to use for the translations.
  `p_maxLineSize` - the maximum size of line. This defaults to 255 which is the maximum allowable size of a line (the 255 limit is a limitation of the `DBMS_OUPUT` package).

- `PROCEDURE deleteDocument(p_document VARCHAR2)` -- Delete the document from the repository. Please use this API with caution since deleting a document can lead to unexpected errors.

  Parameters: `p_document` -- a fully qualified document name, which can represent either a document or package file, example: `/oracle/apps/ak/attributeSets`.

See `../fnddev/fnd/11.5/patch/115/sql/JDRUTEXS.pls` for the complete specification of the `JDR_UTILS` package.

Example Usage

To put the various APIs supported by the `JDR_UTILS` package in context, this document uses the following example to illustrate the steps that you need to take:

Your workflow region is defined in AK under the Application "Application Object Library (Application Short Name: FND)" and has the "WFNTFWLFULLPAGE " AK region code. This region has personalizations defined in AK as well.

You have migrated to MDS, and would like to do the following:

- Find your region's equivalent document in the MDS repository by examining the regionMap.xml file.
- Print your region's MDS document.
- List any personalizations (also known as customizations) for your region.
- Examine its site level personalization.
- Delete all its personalizations made by the user "JFROST".
- Get all its translations.

Contents

- Step 1 : Reading the regionMap.xml file
  - Step 1.1 : Find your application short name.
  - Step 1.2 : Inspect your regionMap entries.
- Step 2 : Print your region's MDS document.
- Step 3 : List any personalizations for your region.

- Step 4 : Examine the site level personalization.
- Step 5 : Delete a personalization document .
  - Step 5.1: Find the user id for your user.
  - Step 5.2 : Delete a user personalization document given the user id.
- Step 6 : Get all translations. for a particular document.
- Appendix A: Interpreting Personalization Content for Rich Table Interactions

Prerequisite Reading

- OA Framework Personalization Guide

Step 1 : Reading the regionMap.xml file

If your region was migrated from AK to MDS, then you should read the `regionMap.xml` file to find its corresponding MDS document. Note that this step is not appropriate if you defined your region using the OA Extension. In that case, you will already have a page name for your MDS document corresponding to your region.

The `regionMap.xml`, created during the migration process, is an XML document (also stored in the JDR tables in the database), that maintains the mapping between your AK region and its equivalent MDS document. Every application has one such file to maintain the mappings for all its regions.

Step 1.1 : Find your application short name

You need the application short name of your application in order to retrieve its MDS documents. You can find the application short name of your application given your application id using the following commands:

```
% sqlplus <apps_username>
Enter password: <apps_password>
SQL> select
decode(lower(app.application_short_name),'sqlap','ap',
'ofa','fa','sqlgl','gl',lower(app.application_short_name)) application
from fnd_application app
where app.application_id = <yourAppId>;
```

Note that using the command above is essential since some applications use a different application short name other than what is stored in FND_APPLICATIONS for their MDS documents.

Step 1.2 : Inspect your regionMap entries

The regionMap contents for a particular application can be displayed using the following sql commands. Replace <application> your application short name **in lower case**:

```
% sqlplus <apps_username>@<yourDatabase>
Enter password: <apps_password>
SQL>set serveroutput on
SQL>spool <application>.lst
SQL>execute jdr_utils.printDocument('/oracle/apps/<application>/regionMap');
```

So, for our example, to print the `regionMap.xml` for the application FND, you should use the following commands:

```
% sqlplus <apps_username>@<yourDatabase>
Enter password: <apps_password>
SQL>set serveroutput on
SQL>spool fnd.lst
SQL>execute jdr_utils.printDocument('/oracle/apps/fnd/regionMap');
```

Now search the <application>.lst file from the SQL output for your AK region. For our example, you should search for WFNTFWLFULLPAGE in fnd.lst. You will find the following mapping:

```
<oa:pageLayout
extends="/oracle/apps/fnd/wf/worklist/webui/FullWorklistPG.WFNTFWLFULLPAGE"
docName="WFNTFWLFULLPAGE" user:akRegionStyle="PAGE_LAYOUT"/>
```

The above means that the region with AK regionCode "WFNTFWLFULLPAGE" corresponds to the MDS document `/oracle/apps/fnd/wf/worklist/webui/FullWorklistPG`. You can ignore the repeated WFNTFWLFULLPAGE proceeding the `/oracle/apps/fnd/wf/worklist/webui/FullWorklistPG` reference. This MDS locator is the base page document reference that was delivered by Oracle E-Business Suite as part of the upgrade from AK to MDS.

Step 2 : Print your region's MDS document

Once you know the MDS document corresponding to your region, you can inspect it further using the `printDocument` API. So, for our example, you can print the contents of the WFNTFWLFULLPAGE region using the following command:

```
% sqlplus <apps_username>
Enter password: <apps_password>
SQL>set serveroutput on
SQL>spool WFNTFWLFULLPAGE.lst
SQL>execute jdr_utils.printDocument('/oracle/apps/fnd/wf/worklist/webui/FullWorklistPG');
```

This document would look like:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<!-- dbdrv: exec java oracle/jrad/tools/xml/importer XMLImporter.class java &phase=dat+24
checkfile:~PROD:~PATH:~FILE &fullpath_~PROD_~PATH_~FILE -username &un_apps -password &pw_apps -dbconnection
&jdbc_db_addr -userId "1" -rootPackage /oracle/apps/~PROD -rootdir &fullpath_~PROD_mds_directory -->
```

```
<page xmlns="http://xmlns.example.com/jrad" xmlns:ui="http://xmlns.example.com/uix/ui"
xmlns:oa="http://xmlns.example.com/oa" xmlns:user="http://xmlns.example.com/jrad/user" file-version="$Header:
FullWorklistPG.xml 115.15 2003/09/08 06:46:59 shanjgik noship $" version="9.0.3.6.3_419" xml:lang="en-US">
<content>
<oa:pageLayout id="WFNTFWLFULLPAGE" akRegionCode="WFNTFWLFULLPAGE" regionName="Worklist"
amDefName="oracle.apps.fnd.wf.worklist.internal.server.FullWorklistAM" helpTargetAppShortName="fnd"
helpTarget="FND_WFNTFWLFULLPAGE">
.....</oa:pageLayout>
</content>
</page>
```

Step 3 : List any personalizations for your region

You can use the MDS base document reference for your region to display all its personalization documents including the ones that were created as part of the AK to MDS personalization migration process using the `listCustomizations` API.

So, for our example, you should use:

```
% sqlplus <apps_username>
Enter password: <apps_password>
SQL>set serveroutput on
SQL>spool fnd_WFNTFWLFULLPAGE_customizations.lst
SQL>execute jdr_utils.listCustomizations ('/oracle/apps/fnd/wf/worklist/webui/FullWorklistPG');
```

The `jdr_utils.listCustomizations` command shown above returns the following list of personalizations at the "localization", "function", "site" and "user" levels. Note that the actual results may vary, and the output here is used to demonstrate the use of these APIs only.

- /oracle/apps/fnd/customizations/localization/IN/wf/worklist/webui/FullWorklistPG
- /oracle/apps/fnd/customizations/function/ZZZ_LOCAL_FUNCTION/wf/worklist/webui/FullWorklistPG
- /oracle/apps/fnd/customizations/site/0/wf/worklist/webui/FullWorklistPG
- /oracle/apps/fnd/customizations/user/1324/wf/worklist/webui/FullWorklistPG

Step 4 : Examine the site level personalization

You can identify the personalization level from the output above by examining the personalization document reference. The words "`localization`", "`function`", "`site`", "`user`" and so on, follow the word *customization* in the document path. You should use this full document path to print and examine this personalization document from the MDS repository. You can print the personalization documents using the `printDocument` API.

So, to examine the site level personalization for your region WFNTFWLFULLPAGE, you should use the following commands:

```
% sqlplus <apps_username>
Enter password: <apps_password>
SQL>set serveroutput on
SQL>spool WFNTFWLFULLPAGE_Site.lst
SQL>execute jdr_utils.printDocument('/oracle/apps/fnd/customizations/site/0/wf/worklist/webui/FullWorklistPG');
```

This document would look like:

```
<?xml version='1.0' encoding='UTF-8'?>
<customization xmlns="http://xmlns.example.com/jrad"
xmlns:ui="http://xmlns.example.com/uix/ui" xmlns:oa="http://xmlns.example.com/oa"
xmlns:user="http://xmlns.example.com/user" version="9.0.3.6.7_683"
xml:lang="en-US"
customizes="/oracle/apps/fnd/wf/worklist/webui/FullWorklistPG"
developerMode="false">
<modifications>
<move element="Subject" after="HelpText"/>
<move element="SSSubject" after="Subject"/>
<move element="SSFrom" after="SSSubject"/>
<move element="SSSent" after="SSFrom"/>
<modify element="Subject" initSortSeq="first" rendered="false"/><insert after="Subject">
<oa:staticStyledText id="test" rendered="true"
queryable="true" sortState="no" totalValue="false" userCustomizable="true"
adminCustomizable="true" initSortSeq="none"
user:akAttributeCode="TEST_ATTRIBUTE"
user:akAttributeApplicationId="2500"/>
</insert></modifications>
</customization>
```

Here is a simple table that explains the different tags used in the customization documents:

Tag	Meaning	Example
`insert`	Insert a new item at the specific level	`<insert after="Subject">` `<oa:staticStyledText id="test" rendered="true"/>` => inserts a new item staticStyledText item with id "test" after the item with id "Subject" with a rendered property set to true.
`modify`	Modify or Personalize a property of an existing item like changing a prompt, rendered flag and so on.	`<modify element="Subject" initSortSeq="first" rendered="false"/>` => personalizes item with id "Subject" to turn its rendered flag of and include an initial sort setting of first order sort.
`move`	Re-order items within a region or document.	`<move element="Subject" after="HelpText"/>` => moves item with id "Subject" after the item with id "HelpText".

queryCriteria	Admin defined criteria object for constructing the where clause for a table region.	</queryCriteria> <criterion element="Srownerid" operator="is" operand="-1111"/> <criterion element="Srowner" operator="is" operand="Unassigned" joinCondition="and"/> </queryCriteria> The individual criterion as shown above are joined together using the joinCondition to create the where clause.
criterion	Individual criterion of a queryCriteria object - forms portions of the where clause	<criterion element="Srownerid" operator="is" operand="-1111"/> Means that a where clause needs to be defined for the element "Srownerid".
operator	Operator that is used for constructing the where clause in SQL.	<criterion element="Srownerid" operator="is" operand="-1111"/> Means that a where clause needs to be defined for the element "Srownerid" with the operator "is" (which in SQL translates to "=").
operand	Actual value for the element in a criterion.	<criterion element="Srownerid" operator="is" operand="-1111"/> Means that a where clause needs to be defined for the element "Srownerid" with the operator "is" and value "-1111". This will translate to the following in SQL: <columnNameForSrownerid> = "-1111"
joinCondition	The individual criterion are joined together using the joinCondition to create the where clause.	<criterion element="Srownerid" operator="is" operand="-1111"/> <criterion element="Srowner" operator="is" operand="Unassigned" joinCondition="and"/> There is one join condition per queryCriteria. This will translate to the following in SQL: <columnNameForSrownerid> = "-1111" **AND** <columnNameForSrowner> = "-Unassigned"
developerMode	If set to true on a personalization document, it cannot be edited or deleted at the customer site.	<customization customizes="/oracle/apps/.." **developerMode="true"** ..>

Step 5 : Delete a personalization document

You can delete a personalization document using the deleteDocument API by specifying its full document path as show below:

```
% sqlplus <apps_username>
Enter password: <apps_password>
SQL>set serveroutput on
SQL>execute jdr_utils.deleteDocument ('<fullPathOfDocument>')
SQL>commit;
```

In our example we would like to delete all personalizations made for the user JFROST on your region WFNTFWLFULLPAGE. You should hence specify the full path of the user personalization document for the user JFROST.

Step 5.1: Find the user id for your user

Since the personalization document path uses the userId, you should first find the user_id corresponding to the user JFROST. You can do so using the following SQL:

```
% sqlplus <apps_username>
Enter password: <apps_password>
SQL> select user_id from fnd_user where user_name = "JFROST";
```

The above command gives the following output:

```
 USER_ID
----------
      1324
```

Note that the actual results may vary in your use and the output here is used to demonstrate the use of these commands only.

Step 5.2 : Delete a user personalization document given the user id.

Now, look at the output from Step 3 to find the user personalization document corresponding to JFROST (userid: 1324). You can then delete it using the following command:

```
% sqlplus <apps_username>
Enter password: <apps_password>
SQL>set serveroutput on
SQL>execute jdr_utils.deleteDocument
('/oracle/apps/fnd/customizations/user/1324/wf/worklist/webui/FullWorklistPG');
```

```
SQL>commit;
```

Note that there are two other forms of user personalization documents:

- Oracle-seeded user-level -- User personalizations seeded by Oracle: This personalization document use the special keyword "seeded_developer" instead of the userid. There is one seeded_developer document for a specific region that stores all its seeded developer views.

  So, in our example, the seeded_developer document will be of the form:
  `/oracle/apps/fnd/customizations/user/seeded_developer/wf/worklist/webui/FullWorklistPG`

- Admin-Seeded User-Level -- User personalizations created by the administrator for all users: This personalization document use the special keyword "seeded_customer" instead of the userid. There is one seeded_customer document for a specific region that stores all its seeded customer views.

  So, in our example, the seeded_customer document will be of the form:
  `/oracle/apps/fnd/customizations/user/seeded_customer/wf/worklist/webui/FullWorklistPG`

Step 6 : Get all translations for a particular document

If you would like to know what languages are supported for your document, you can use the `listLanguages` API.

So, in our example, you can list all translations on your WFNTFWLFULLPAGE using the commands below:

```
% sqlplus <apps_username>
Enter password: <apps_password>
SQL>set serveroutput on
SQL> execute jdr_utils.listLanguages ('/oracle/apps/fnd/wf/worklist/webui/FullWorklistPG');
```

Appendix A: Interpreting Personalization Content for Rich Table Interactions

Rich table Interactions such as resize and reorder, introduced in Release 12.2.3, persist across sessions by means of user-level views. Previously, OA Framework supported views only for tables under query regions. Now OA Framework extends view support to classic and advanced tables in all pages.

For example, suppose you use rich table interactions to resize and reorder the columns on the Employees Search page, `/oracle/apps/fnd/framework/toolbox/labsolutions/webui/EmpSearchPG`. To list the personalized view, execute:

```
jdr_utils.listCustomizations('/oracle/apps/fnd/framework/toolbox/labsolutions/webui/EmpSearchPG');
```

The output may appear as follows:

```
/oracle/apps/fnd/framework/toolbox/labsolutions/webui/customizations/user/0/EmpSearchPG
```

where `user/0` indicates a user level personalization for user ID 0 (in this case, user SYSADMIN).

To view the contents of this personalization document, execute:

```
jdr_utils.printDocument('/oracle/apps/fnd/framework/toolbox/labsolutions/webui/customizations/user/0/EmpSearchPG');
```

The content of the document may appear as follows:

```
<?xml version='1.0' encoding='UTF-8'?>
<customization xmlns="http://xmlns.example.com/jrad" version="9.0.6.0.0_51" xml:lang="en-US"
  customizes="/oracle/apps/fnd/framework/toolbox/labsolutions/webui/EmpSearchPG" developerMode="false">
<views>
<view id="view1" default="true" element="ResultsTable" MDSActiveView="true" name="view1">
<modifications>
<modify element="ResultsTable" columnswidthkey="DeleteSwitcher:78,PopupLayout:126,EmpNum:337,EmpNameLayout:92,
  EmpStatus:85,UpdateImage:259,Position:77,AttachmentImage:69,MgrName:65,"/>
<move element="EmpNum" after="AttachmentImage"/>
<move element="EmpStatus" after="EmpNum"/>
<move element="EmpNameLayout" after="EmpStatus"/>
<move element="Position" after="EmpNameLayout"/>
<move element="MgrName" after="Position"/>
<move element="DeleteSwitcher" after="MgrName"/>
<move element="UpdateImage" after="DeleteSwitcher"/>
<move element="PopupLayout" after="UpdateImage"/>
</modifications>
</view>
</views>
</customization>
```

Column Resize

For classic tables, as shown in the Employees Search page example above, if a user resizes the columns in the table, the personalization document shows an attribute named **columnswidthkey,** set on the table web bean, that specifies the table's new column widths. For advanced tables, the personalization document displays the column width information at each column level, as shown here:

```
<?xml version='1.0' encoding='UTF-8'?>
<customization xmlns="http://xmlns.example.com/jrad" version="9.0.6.0.0_51" xml:lang="en-US"
  customizes="/oracle/apps/fnd/framework/toolbox/samplelib/webui/AdvancedTablePG" developerMode="false">
<views>
<view id="view1" default="true" element="Table3RN" MDSActiveView="true" name="view1">
<modifications>
<move element="Tbl3StartDate" after="Tbl3Salary"/>
<move element="Tbl3LastName" after="Tbl3StartDate"/>
<move element="Tbl3FirstName" after="Tbl3LastName"/>
<modify element="SalaryCol" width="98"/>
<modify element="FirstNameCol" width="261"/>
```

```
<modify element="LastNameCol" width="224"/>
<modify element="EndDateCol" width="101"/>
<move element="MeaningCol" after="LookupCodeCol"/>
<move element="DescriptionCol" after="MeaningCol"/>
<move element="LookupTypeCol" after="DescriptionCol"/>
<move element="SalaryCol" after="EmpNameColGrp"/>
<move element="EndDateCol" after="SalaryCol"/>
<modify element="Tbl3FirstName" width="192"/>
<modify element="Tbl3StartDate" width="98"/>
<modify element="Tbl3LastName" width="151"/>
<modify element="Tbl3Salary" width="105"/>
</modifications>
</view>
</views>
</customization>
```

Column Reorder

If a user reorders the column in a table, the personalization document saves the information as follows, for both classic and advanced tables:

```
<move element="EmpNum" after="AttachmentImage"/>
<move element="EmpStatus" after="EmpNum"/>
<move element="EmpNameLayout" after="EmpStatus"/>
<move element="Position" after="EmpNameLayout"/>
<move element="MgrName" after="Position"/>
<move element="DeleteSwitcher" after="MgrName"/>
<move element="UpdateImage" after="DeleteSwitcher"/>
<move element="PopupLayout" after="UpdateImage"/>
```

Column Hide/Show

If a user Hides or Shows a column in a table, the personalization document saves the information as follows, for both classic and advanced tables:

```
<modify element="EmpNum" rendered="false"/>
<modify element="EmpStatus" rendered="false"/>
<modify element="EmpNameLayout" rendered="true"/>
<modify element="Position" rendered="true"/>
<modify element="MgrName" rendered="true"/>
<modify element="DeleteSwitcher" rendered="true"/>
<modify element="UpdateImage" rendered="true"/>
<modify element="PopupLayout" rendered="false"/>
```

Deleting a Personalization

Currently, you can not, from the user interface, delete a user view created as result of rich table interactions. However, as an administrator, you may execute `jdr_utils.deleteDocument` to delete the information related to column resizing and reordering, such that when you display the page again, the previous personalizations no longer appear.

Note: Since the rich table interaction view document is generated at the user level, you should always identify the correct user ID before deleting.

Warning: For a table under a query region, the `jdr_utils.deleteDocument` command does not distinguish between the personalization data generated from a user's rich table interactions versus that from an actual view the user creates from the Views panel of the Search page. Using the `jdr_utils.deleteDocument` command deletes all saved information.

Testing OA Framework Applications

Overview

- Prerequisite
- Running in 'Test' Modes
  - Developer Test Mode
  - Back Button Test Mode
  - Connection Test Mode
  - Recommended Test Scenarios
- BC4J Diagnostics
- Using the Business Components Browser (BC4J Tester)
- Verifying HTML Page Size
- Verifying SQL Performance (Enabling a Database Trace)
- Monitoring the Application Module / JDBC Connection Pools

Prerequisite

Generally, when testing your work, you should make sure that the FND: Diagnostics profile option value is set to **Yes** for your development environment. This ensures that you have access to the error stack directly in your page when unexpected exceptions are thrown. If this value is set to **No**, a generic user-friendly error message displays with no access to the error stack.

While running your pages in JDeveloper, enabling the Developer Test Mode as described below (even if the **FND: Diagnostics** profile option value is **No**) also ensures that you will see detailed error messages.

Quality Assurance teams should also set the profile option value to **True** to ensure that any bugs they log include meaningful error information.

Running in 'Test' Modes

The OA Framework includes several "test" modes that let you quickly and easily identify various coding standards violations. This section describes each test mode ("Developer," "Back Button," and "Connection") individually, and then discusses their use together in recommended testing scenarios.

Developer Test Mode

This mode tests code compliance with numerous coding standards.

Enabling Developer Test Mode

To enable this test mode, set the set **FND: Developer Mode** (FND_DEVELOPER_MODE) profile option value to **Yes**.

Back Button Test Mode

We do not recommend that you use the Back button test mode. Please see Supporting the Browser Back Button: Testing Back Buton Navigation for additional information.

Connection Test Mode

Enables connection release tests.

Note: You should use the Connection Test Mode for pages whose application module's `Retention Level` is set to **CONNECTION_AGNOSTIC**.

Enabling Connection Test Mode

To enable this test mode, set a cookie value in your test.jsp or set a JVM-level system property as shown (the possible values are described below):

- In your test.jsp, set the cookie value `OAConnectionTestMode`.
- Alternatively, set the JVM-level system property `oa.connection.testmode` (for example: `-Doa.connection.testmode=1`).

Note: The cookie value takes precedence over the system property value. The system property value can be used in the Quik Apache test environment.

The connection test mode can be set to the following values:

Value	Description	Comments
0	Connection test mode is off; no connection release test occurs.	This is the default value. Note: The connection "test" may be disabled, but connection release activity will occur when connection pooling is enabled.
1	Connection test mode is on	This case simulates a request boundary connection release (occurs when the AOL/J connection harvester lazily reclaims connections if system load is high, or the **FND: Application Module Connection Pool Enabled** profile value is set to **Yes**). You should use this test mode value to cerify pages whose root application module's `Retention Level` is set to **CONNECTION_AGNOSTIC**. In this test mode, if the current page's root application module `Retention Level` is **CONNECTION_AGNOSTIC** or **MANAGE_STATE**, the OA Framework releases its connection at the end of page processing before forwarding to another page. Database state will be lost, and any posted changes will be rolled back.
2	Connection test mode is on; this mode also simulates setting the root application module's `Retention Level` to **CONNECTION_AGNOSTIC**.	If you believe that your code is connection-agnostic (meaning it does not depend on the use of a single connection across requests), but have not yet changed the application module's `Retention Level` to **CONNECTION_AGNOSTIC**, you can use this mode for preliminary testing. Do NOT use this test mode value to certify your connection-agnostic application modules (use the "1" value). In this test mode, the OA Framework releases the current page's root application module connection at the end of page processing before forwarding to another page, *regardless of the application module's retention level*. Database state will be lost, and any posted changes will be rolled back.

Usage Notes

- Use the connection test mode to test the connection release instead of changing the **FND: Application Module Connection Pool Enabled** system profile value as this affects all the application users.

What Happens?

When the connection is released, the database state will be lost and the posted changes will be rolled back. Make sure that the the behavior when the connection test mode is turned on or off are the same. If the application page cannot recover some state values and therefore cannot continue the transaction when the connection test mode is turned on, it should at least gracefully fail with some customized error message instead of throwing a severe exception with an error stack.

Verifying Connection Test Mode

To verify that the connection test mode is set correctly, enable BC4J diagnostics (`-Djbo.debugoutput=console`) in JDeveloper and look for the diagnostic statement Connection test mode = 0 (or 1, or 2) in your JServ or error log. You can also call `OAPageContext.isConnectionTestMode`, which returns true if the mode value is 1 or 2.

Recommended Test Scenarios

This section describes how to use the different test modes together during development unit test and system test phases.

Warning: You should address all your Developer Mode test exceptions *before* proceeding to Step 2. Upstream tests are not valid if errors persist from prerequisite tests.

Unit Testing

The unit tests should help you identify code flaws before releasing it to others.

Note: You should not use the Back button test mode. Please see Supporting the Browser Back Button: Testing Back Button Navigation for additional information.

Step	Settings	Verifies
1	OABackButtonTestMode=0	Your pages will work in the most common page flow scenarios.

System Testing

The system tests should simulate typical customer installations. For example, the "Developer Mode" test should never be on at a customer site.

Note: You should not use the Back button test mode. Please see Supporting the Browser Back Button: Testing Back Button Navigation for additional information.

Step	Settings	Verifies
1	OABackButtonTestMode=0	Your pages will work in the most common page flow scenarios.

Regression Testing

A system tester can also use regression test suites to automate these tests.

BC4J Diagnostics

BC4J diagnostics is fully integrated with FND logging in that you can direct BC4J logging to the same destination and control it using the same flags as FND logging. Prior to Release 12, if you enabled BC4J diagnostics by setting `-Djbo.debutoutput=console` in JDeveloper, the output would go to the console or stdout. The previous output did not correlate to the FND diagnostic messages that were printed elsewhere, and you could only control it through JVM runtime parameters. The JVM runtime parameters would only take effect if you bounced the web application server's listener and JVM and would affect all users on that JVM.

The integration of BC4J diagnostics with FND logging now results in the following features:

1. BC4J diagnostic messages are transformed into FND diagnostics messages so that they share the same format and include message level, module name and header information about the user/execution thread that generated the message.
2. Output is directed to the same destination as FND diagnostics, providing easy correlation with Oracle E-Business Suite messages. The output can be directed to the screen, a file or a database table.
3. The same flags and filters that control FND diagnostics are now recognized by BC4J diagnostics. This includes the AFLOG profiles (AFLOG_LEVEL, AFLOG_MODULE, AFLOG_FILENAME, and AFLOG_ENABLED) and its correponding URL parameters and profile options. You can now also control BC4J diagnostics through the **Show Log on Screen** page accessible from the **Diagnostics** global button.

   Note: The module name that corresponds to BC4J log statements is `jbo`. The level that BC4J diagnostics map to is STATEMENT.
4. BC4J diagnostics can now be controlled at runtime, system-wide or for a particular user. You can also set BC4J diagnostics on a running JVM without having to restart it.

You can enable all these features by setting the `jbo.debugoutput` to point to the `OADiagnostics` class, which in turn is responsible for the diagnostics message transformation. In JDeveloper, edit the **Project Settings** for your project. Select **Configurations** > **Runner** in the navigation tree and add the following to the **Java Options** field:

```
-Djbo.debugoutput=oracle.apps.fnd.framework.server.OADiagnostics
```

In a production environment, you need to add the above line to an oc4j.properties file or equivalent.

For complete information about logging, refer to the *Oracle E-Business Suite Supportability Guide* available on the Applications Release 12 Online Documentation CD.

Using the Business Component Browser (BC4J Tester)

When you create view objects and associate them with application modules, you can use the Business Component Browser (also known as the BC4J Tester) to run your view objects before you build an OA Framework UI for them, or write any code to support your BC4J objects. For example, you can query view objects (including the ability to navigate through master rows to query children linked

with a view link).

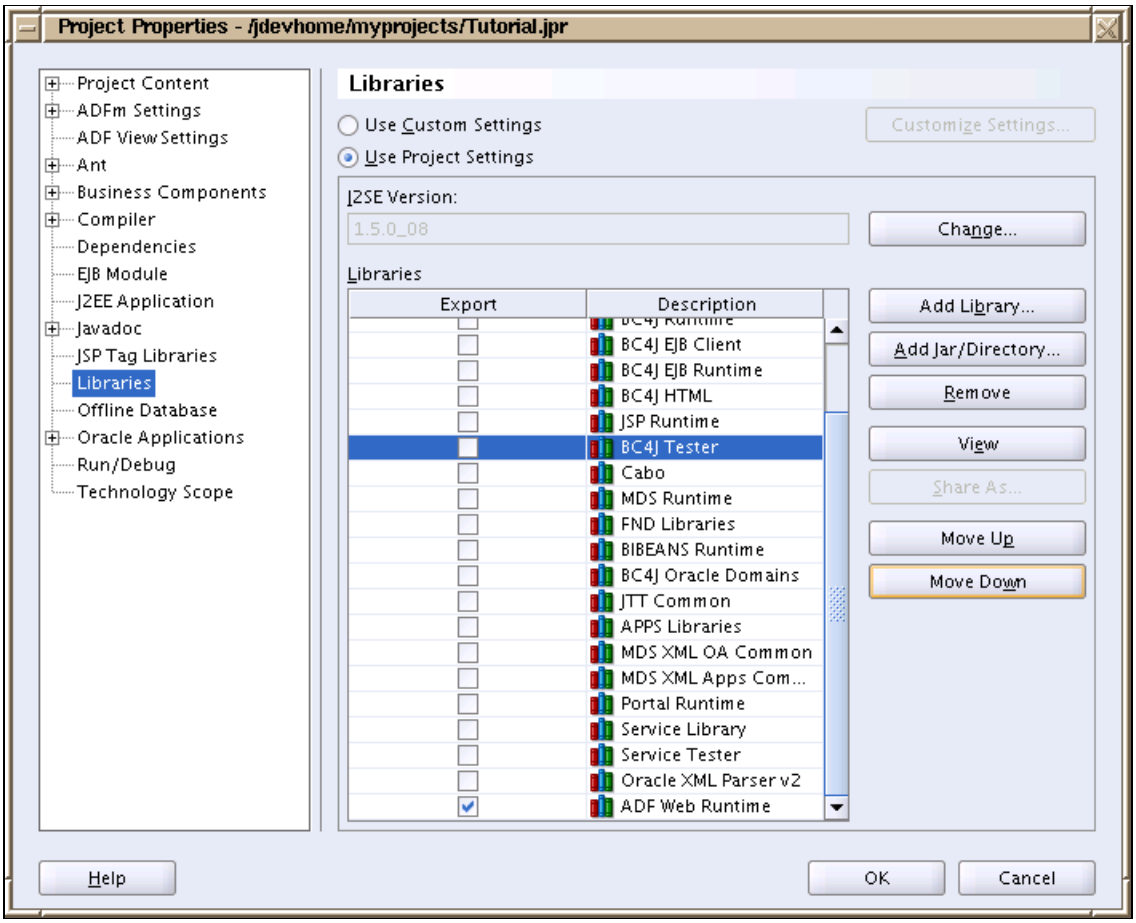
To get started with an OA Framework project, follow these instructions:

Step 1: Ensure the **BC4J Tester** library has been added to your project *before* the **Cabo** library.

- Select your project in the **Applications Navigator**, right-click and select **Project Properties ...**
- In the **Project Properties** window, select the **Libraries** node.
- In the **Libraries** page, select **Add Library** to display the **Add Library** dialog. Select the **BC4J Tester** library from the list of available libraries and select **OK**.
- In the **Libraries** page, use the **Move Up** button to move the **BC4J Tester** library until it is sequenced before the **Cabo** library as shown in Figure 1.

Tip: If you have multiple projects in your workspace, you will need to make this change to the library list for all of them to avoid a runtime error.

Figure 1: Project Settings showing the BC4J Tester sequenced before the Cabo library



Step 2: To start the Tester, select your application module in the **Applications Navigator,** right-click and select **Test...** . BC4J displays a **Connect** dialog; select the **Connect** button to proceed.

Step 3: When the **Oracle Business Component Browser** window displays, your application module's view objects are listed. Select a view object, right-click and select **Show** to query the data. A navigation bar displays with the results so you can scroll through the rows and test creating and deleting rows.

For additional information about using the features of the Business Component Browser, see the Oracle JDeveloper 10*g* Help (look for the topic *Testing Business Components*).

Verifying HTML Page Size

To verify your HTML page size:

Step 1: Run your page in Internet Explorer.

Step 2: Right-click on the page and select **Properties** to view the size.

If this doesn't work, use the **File > Save As** or **View > Source** Internet Explore menu option to create a file. You can then check its size on the file system.

Note that the size includes HTML only (no images).

Verifying SQL Performance (Enabling a Database Trace)

To enable a database trace:

Step 1: Set the profile option FND: Diagnostics to **Yes** at the user level.

Step 2: Login to the E-Business Suite Home Page (or run your page from JDeveloper) as this user.

Step 3: Enable the trace.

1. Select the Diagnostics global button at the top of the page.
2. Select **Set Trace Level** from the poplist and select **Go**.
3. Select the desired trace level from the following list of available options and select **Save**.
    - **Trace (regular)** - Enable standard SQL_TRACE functionality.
    - **Trace with binds** - Enable SQL_TRACE functionality with trace bind values. If you enable this level, Oracle reports bind variable information such as the bind variable data types and the actual bind variable data.
    - **Trace with waits** - Enable SQL_TRACE functionality with trace waits. This level helps identify the wait events that the session is waiting on. For example, the session may be waiting for an I/O event or waiting for a particular latch. This information is especially useful when the SQL statement time is suspicious in terms of actual work versus elapsed time. The wait events can be used to locate bottlenecks and areas of contention. The session may be waiting on I/O events which may be causing abnormal elapsed times. You can use this level to spot latch wait, as well as spot full table scans and index scans.
    - **Trace with binds and waits** - Enable SQL_TRACE functionality with both trace bind values and waits.

      In general, the **Trace** and **Trace with Waits** levels are the most useful in terms of SQL execution statistics. To analyze the bind variable information or wait event information, you need to examine the raw SQL trace file (not tkprof report). The raw SQL trace file contains the data which tkprof uses to build an easy-to-read format. For example, by scanning through the raw SQL trace file with a utility such as grep or awk to look for the WAIT event lines in the trace, you can quickly identify in the trace the large wait times.

- Write down the trace id number(s).

Step 4: Use your menu to navigate to the task you want to perform with a trace.

Step 5: If you want to perform multiple traces in the same session with different trace levels, simply repeat Step 3 and select a new trace level. When you're ready to disable the trace feature, repeat Step 3 but choose Disable Trace.

Step 6: Go to `user_dump_dest` for your database and collect the raw trace file(s) suffixed by the trace id number(s) you recorded.

Monitoring the Application Monitor / JDBC Connection Pools

See Application Module and Connection Pooling for instructions.

Oracle Application Framework Troubleshooting

Overview

The most current version of this document is published in My Oracle Support (formerly Oracle*MetaLink*) Knowledge Document 1512113.1. It describes common problems and solutions related to the installation and configuration of the OA Framework, and running OA Framework-based self-service applications (see the Oracle Application Framework Development FAQ for common problems and questions related to ongoing development using the OA Framework).

Chapter 8: OA Framework File Standards

Overview

This document describes file naming conventions, standard file contents, and directory (package) structures for all OA Framework application files. The document is organized as follows:

- Package Naming
- File Naming
- Standard File Contents/Organization
  - Java Files
  - OA Component XML Files
  - BC4J XML Files
- Package / Source Control Directory Structure

Standard Compliance

The naming and package structure standards described in this document are required *only if* they can be applied consistently to all the files in your product. If you have older files that comply with earlier versions of the OA Framework naming and package/directory standards, it is not necessary to rework them to satisfy the current guidelines (in this case, we recommend that you follow the original conventions when creating new files to ensure consistency within your product). That said, however, it is preferable for all products to follow the same conventions to facilitate discovery, reuse and customer extensions. If you can schedule a refactoring exercise, you should.

Related Information

- Files in a Typical OA Framework Application

Package Naming

Note: For customers, see Extending OA Framework Applications for recommended package structure and naming conventions for custom logic.

Package names are used to group Java classes/interfaces and individual XML UI files. The Oracle corporate standard on package names requires that they begin with a lower case letter and use initial capitals rather than underscores for each subsequent word in the name (for example: oracle.apps.fnd.wf.statusMonitor).

At the highest level, all Oracle E-Business Suite code goes into the oracle.apps package. You should *NOT* create your code under the following restricted packages:

- oracle.apps.fnd.framework
- oracle.jrad
- oracle.mds
- oracle.jbo
- oracle.cabo
- oracle.jdbc

See the Package / Source Control Directory Structure section below for additional package definition instructions.

Warning: Do not confuse the Java package (directory structure) with the OA Components Package, a special XML file that can contain multiple, related OA components. Naming standards for the OA Components Package are provided below.

File Naming

All files in an OA Framework application should comply with the following naming conventions.

Java Class/Interface Names

File names for general Java classes and interfaces should comply with Oracle corporate naming guidelines: class names should be succinct, descriptive nouns.

- A factory class for a class should be named by appending the word Factory to the original class's name (for example, OAWebBeanFactory)
- Exception classes should have the word Exception as the last part of their names (for example, OAAttrValException)

Name Length Limit

Java Files

Java files names may not exceed 50 characters. If inner classes are used, the concatenated inner class name with the outer class name may not exceed 50 characters.

OA Extension Component XML Files

OA Extension (page, region, attribute set and package) XML file names may not exceed 30 characters. Furthermore, for performance reasons, these file names must always be as short as possible *regardless* of the 30 character limit. Given this, you must try to use *common* abbreviations like "Emp" for "Employee," "Num" for "Number ," "Desc" for "Description" and so on whenever possible. Do not sacrifice clarity for brevity. If the abbreviation cannot be understood by the average consultant or customer, do not abbreviate.

BC4J XML Files

BC4J XML files names may not exceed 30 characters.

Capitalization

For all file type names (except for the generated BC4J server.XML, the LAF `<lookAndFeelFamilyName>-<device>.xml` (all lower case),`<lookAndFeelFamilyName>-<device>.xss` (all lower case), and custom renderer `<webBeanType>.uit` files), the first letter in every word should be capitalized *except* where standard, upper case suffixes are required as described below. Note that approved abbreviations and acronyms should also use the initial capitalization scheme. For example:

Correct	Incorrect
PoSearchPG	**PO**SearchPG
Suppliers**Lov**RN	Suppliers**LOV**RN

Naming Summary by File Type

This provides an at-a-glance look at the naming standards by OA Framework file type.

File Extension	File Type	Standard	Examples
.xml	Page Definition	<Object><Function>PG or <Object>PG 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). **Warning:** Never give pages step number names (for example: PoCreateStep1PG.xml, PoCreateStep2PG.xml). Always describe the page function (PoDescPG.xml, PoLinesPG.xml)	• EmployeePG.xml (employee update) • EmpCreatePG.xml (differentiated only if update and create are separate tasks) • EmpViewPG.xml (view employee info) • EmpHirePG.xml (hire new employees) • EmpPromotePG.xml (promote employees) • EmpSearchPG.xml (search for employees) • HomePG.xml (home page) • MgrHomePG.xml (home for tailored for managers) • PoAttachPG.xml (attachments specific to POs) • AttachPG.xml (attachments not specific to one object)

.xml	Shared Region Definition	<Object><Function-Structure>RN or <Object>RN The region 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 or the structure (search, promote, hire, approve, view, table, HGrid, Tree and so on). For some pages, the object is sufficient. **Note:** Some older regions are named with suffix "RG". All new regions should use "RN as shown". **Warning:** There are additional naming standards for regions which are created within the context of a parent page, and not in their own XML file as is the case for a shared region. See the OA Component XML Files section below for additional information.	• EmpSummaryRN.xml • PoApproveRN.xml • CustomerContactsRN.xml • ItemTypeHGridRN.xml
.xml	Shared List of Values (LOV) Definition	<DescriptiveName>LovRN The Lov should be named for the objects the LOV queries.	• RetiredEmpsLovRN.xml • AllEmpsLovRN.xml • MgrEmpsLovRN.xml
.xml	Attribute Set Package	<TableName> An attribute set file name should match the name of the associated table (every table should have its own attribute set). **Note:** If you need to create an attribute set package for commonly used transient attributes with no associated base table, name the package **Transient**. See the OA Component XML Files section below for additional information about attribute set naming.	• FwkTbxEmployees.xml (table name is FWK_TBX_EMPLOYEES) • FwkTbxPoHeaders.xml (table name is FWK_TBX_PO_HEADERS) • Transient.xml
.xml	UI Components Package	<ModuleName> The package file should be named for the module files it contains. See the OA Component XML Files section below for additional information about the package file, including when and how to use it.	• NewHire.xml • ItemCatalog.xml • PoCreate.xml • ApprovedSupplierList.xml
.xml, .java	Entity Object	<EntityName>EO The EO should be named for the objects stored in its underlying entity. For example, the entity FWK_TBX_PO_HEADERS stores purchase order headers, so the associated EO name is PurchaseOrderHeaderEO. EO names are always singular as they model a single object. **Exception:** Java entity objects created for **_TL** tables should be named **<EntityName>TLEO**. The entity object created for the corresponding **_B** table should follow the standard naming convention. See Entity Objects for Translatable Tables for additional information.	• EmployeeEO.xml EmployeeEOImpl.java • SupplierEO.xml SupplierEOImpl.java • SupplierSiteEO.xml • SupplierSiteEOImpl.java • PurchaseOrderHeaderEO.xml PurchaseOrderHeaderEOImpl.java • PurchaseOrderLineEO.xml PurchaseOrderLineEOImpl.java • LookupCodeTLEOL.xml LookupCodeTLEOOImpl.java • LookupCodeEO.xml LookupCodeEOImpl.mxl

.xml	[Entity] Association Object	<Parent>To<Child>AO The AO name should convey the relationship between a parent and its child entities (or entity if it's a 1:1).	• PoHeaderToLinesAO.xml • SupplierToSitesAO.xml • EmployeeToContactsAO.xml
.xml, .java	View Object / View Row	<DescriptiveName>VO The VO name should convey the nature of the query. VO names are always plural as they model a collection of rows.	• AllEmployeesVO.xml AllEmployeesVOImpl.java AllEmployeesVORowImpl.java • NewHiresVO.xml NewHiresVOImpl.java NewHiresVORowImpl.java
.xml	View Link	<Master>To<Detail>VL The VL name should convey the relationship between the master and detail VOs.	• ManagerToDirectsVL.xml • PoHeaderToLinesVL.xml • ItemToApprovedSuppliersVL.xml
.xml, .java	Application Module	<ModuleName>AM The AM name should convey the purpose of the UI module it services (which is either a shared region, or a discrete "task" ranging in scope from one to several related pages). **Note:** If you opt to generate an interface for your application module, BC4J will automatically assign this name as the interface name.	• EmployeesAM.xml EmployeesAM.java (optional interface) EmployeesAMImpl.java • ItemCatalogAM.xml ItemCatalogAMImpl.java • PoSummaryAM.xml PoSummaryAMImpl.java • ApprovedSupplierListAM.xml ApprovedSupplierListAMImpl.java
.xml, .java	[Validation] Application Module	<TopLevelEntityName>VAM Since VAMs are associated with the top-level object in a composition (or an individual entity) they should be named for the EO noun(s), or the composition. For example, a "purchase order" is comprised of a PurchaseOrderHeaderEO with a PurchaseOrderLineEO and a PurchaseOrderShipmentEO. In this case, the expert is named "PurchaseOrderVAM."	• EmployeeVAM.xml EmployeeVAMImpl.java • SupplierVAM.xml SupplierVAMImpl.java • PurchaseOrderVAM.xml PurchaseOrderVAMImpl.java
Defined as part of the SAM; may have a .java implementation	Service	<ServiceName>Service For **business object services**, the interface's name should be the singular name of the associated business object.	• PurchaseOrderService PurchaseOrderService.java • SupplierService SupplierService.java • EmployeeService EmployeeService.java • PriceListService PriceListService.java

.xml, .java	[Service] Application Module	<ServiceName>SAM The service implementation's name should match the service bean interface name plus the "SAM" suffix instead of the "Service" suffix. **Note:** Since preexisting application modules may be used for service bean implementations, it is not necessary to change the AM's original name to comply with this naming convention. **Important:** When you define the data model for your service application module, be sure to specify a singular view instance name, without a VO or SVO suffix. For example, if you select the SuppliersSVO to add to the SupplierSAM data model, you must rename the view instance to Supplier, rather than Suppliers or SuppliersSVO.	• EmployeeSAM.xml EmployeeSAMImpl.java • SupplierSAM.xml SupplierSAMImpl.java • PriceListSAM.xml PriceListSAMImpl.java
.xml, .java	[Service] View Object	<PluralObjectName>SVO View objects created to correspond directly to individual entities in a business object should be named for the corresponding entity. View objects created to support specific tasks or data views should be named with a description of the associated data. For example, if a view object includes candidate new hires for reference during an interview/approval process, the view object might be named CandidatesSVO. Top-level SVOs in a composite object should be given a business object name. For example, the top-level table in the ToolBox purchase order is FWK_TBX_PO_HEADERS, but the associated SVO is PurchaseOrdersSVO since the business object name is "Purchase Order." **Note:** Since preexisting view objects may be used in service bean implementations, it is not necessary to change the VO's original name to comply with this naming convention.	• PurchaseOrdersSVO.xml PurchaseOrdersSVOImpl.java • PurchaseOrderLinesSVO.xml PurchaseOrderLinesSVOImpl.java • RejectedRequisitionsSVO.xml RejectedRequisitionsSVOImpl.java • CurrentEmployeesSVO.xml CurrentEmployeesSVOImpl.java
.xml, .java	[Service] Domain Set	<PluralObjectName>DVO Domain set view objects created to correspond directly to individual entities in a business object should be named for the corresponding entity with additional descriptive information if required to identify a specific data set. For example, a DVO that simply queries all employees would be called EmployeesDVO, while one that queries only temps would be called TemporaryEmployeesDVO.	• EmployeesDVO.xml EmployeesDVOImpl.java • ManagersDVO.xml ManagersDVOImpl.java • ActiveEmployeesDVO.xml ActiveEmployeesDVOImpl.java • CandidateNewHiresDVO.xml CandidateNewHiresDVOImpl.java • TemporaryEmployeesDVO.xml TemporaryEmployeesDVOImpl.java

.xml	[Service] View Link	<Master>To<Detail>SVL The view link name should convey the relationship between the master and detail view object. **Important:** When you define the properties of your service view link, be sure to specify singular accessor names for the view link source and destination. For example, when defining a SuppliertoSitesSVL, specify the Source accessor name as Supplier and the Destination accessor name as SupplierSite.	• ManagerToDirectsSVL.xml • PoLineToShipmentsSVL.xml • ItemToApprovedSuppliersSVL.xml
Defined as part of the SVO; may have a .java implementation	[Service] Data Object	<SingularSVOName> The typed data row should be named for the associated view object. For example, if the SVO is named **PurchaseOrderLinesSVO** (plural), the corresponding typed data row should be named **PurchaseOrderLine** (singular).	• PurchaseOrder PurchaseOrder.java • SupplierSite SupplierSite.java • Employee Employee.java • Patient Patient.java
Defined as part of the SVO; may have a .java implementation	[Service] Filter	<DataObject>Filter A filter should be named for the associated data object. For example, if the data object is named **PurchaseOrderLine**, the corresponding filter should be named **PurchaseOrderLineFilter**. Since an SVO can have multiple associated filters, feel free to choose a descriptive name that captures the primary purpose of the filter.	• PurchaseOrderFilter PurchaseOrderFilter.java • ApprovedPurchaseOrderFilter ApprovedPurchaseOrderFilter.java • EmployeeFilter EmployeeFilter.java • ManagerFilter ManagerFilter.java
.xml	[Service] Test Suite	<ServiceName>Test The test suite should be named for the associated service. If your test suite spans multiple services, choose a descriptive name without the "Service" name component. For example: ProcureToPayTest.mxl.	• EmployeeServiceTest.xml • SupplierServiceTest.xml • ProcureToPayTest.xml
.xml, .java	[Validation] View Object	<DescriptiveName>VVO The VVO name should convey, to the extent that is practical/possible, the specific validation that it performs.	• EmpIdForNameVVO.xml EmpIdForNameVVOImpl.java • EmpNameForIdVVO.xml EmpNameForIdVVOImpl.java
.xml, .java	[Application Properties] View object	<AssociatedApplicationModuleName>PVO The PVO name should clearly identify its affiliation with a single application module. For example, if an application module named EmployeeAM has an associated application properties view object, it should be named EmployeePVO.	Application properties view object associated with a PurchaseOrderAM: • PurchaseOrderPVO.xml • PurchaseOrderPVOImpl.java Application properties view object associated with a SupplierAM: • SupplierPVO.xml • SupplierPVOImpl.java

.java	Entity Expert	<TopLevelEntityName>Expert Since experts are associated with the top-level object in a composition (or an individual entity) they should be named for the EO noun(s), or the composition. For example, a "purchase order" is comprised of a PurchaseOrderHeaderEO with a PurchaseOrderLineEO and a PurchaseOrderShipmentEO. In this case, the expert is named "PurchaseOrderExpert."	• EmployeeExpert.java • SupplierExpert.java • PurchaseOrderExpert.java
.xml, .java	UI Controller	<Object><Function>CO or <Object>CO The CO name should convey its usage. Ideally, this should be similar to its associated region name.	• EmpSearchCO.java • EmpResultsCO.java • EmployeeCO.java • EmpFooterCO.java
.xml, .xss	Look-and-Feel (LAF)	<lookAndFeelFamilyName>-<device> The LAF name should convey the LAF family and the device for which the LAF is intended. The Custom Look-and-Feel (CLAF) UI automatically generates .xml and .xss files for the LAF name specified.	• custom-desktop.xml • custom-desktop.xss • custom-pda.xml • custom-pda.xss
.uit	Custom Renderer	<webBeanType> The custom renderer name should convey the web bean component for which you are customizing the rendering. Custom renderer files must reside under `$HTML_TOP/cabo/templates/<lookandFeelId>/` where `<lookandFeelId>` would be a LAF name, such as 'custom-desktop'. Custom icon files must reside under `$HTML_TOP/cabo/images/<lookandFeelId>/` where `<lookandFeelId>` would be a LAF name, such as 'custom-desktop'.	• sideBar.uit • tabBar.uit • pageLayout.uit

Standard File Contents / Organization

Java Files

This section focuses on the structure of your Java files. All Java classes and interfaces that you code must also comply with the various Java coding standards published in this Developer Guide.

Comments

All files should have a standard Oracle E-Business Suite copyright/history comment at the top of the page. Javadoc-style methods should be used for public and protected interface/class declarations, methods and fields. All internal comments should use the double slash `//` which allows for the use of `/* ... */` to comment out chunks of code.

Example of a standard copyright/history comment:

```
/*===========================================================================+
  |   Copyright (c) 2000 Oracle Corporation, Redwood Shores, CA, USA          |
  |                    All rights reserved.                                   |
  +===========================================================================+
  | HISTORY                                                                   |
  | 22-Jan-00 lxharris Created.                                               |
  | 25-Apr-00 mmnakamu Modified to user cabo controls                         |
  | 31-Aug-00 lewe Added initializeChild()                                    |
  | 27-Oct-00 shira Overhauled for MarlinMerge                                |
  | 03-Apr-02 pambale Enhancement 2293247: Added getTotalValues()             |
  | 04-Apr-02 nbajpai Bugfix 2252743: Added setWrapEnabled()                  |
```

```
   | 06-May-02 pambale Doc bug 2311433: Modified method javadocs               |
   | 28-Jun-02 nbajpai deprecated setWrapEnabled()                             |
   | 09-Jul-02 pambale ER 1930333: Added getter and setter for                 |
   | DETAIL_VIEW_ATTRIBUTE_NAME                                                |
   | 18-Sep-02 pambale ER 1930333: Added getter and setter for                 |
   | ROW_HEADER_VIEW_ATTRIBUTE_NAME                                            |
   | 08-Oct-02 pambale ER 2502892: Bypass processForm* for inner               |
   | tables                                                                    |
   | 08-Oct-02 nbajpai Bug 2637673. Added getFWKColumnHeaderData()             |
   | & isCallfromFWK().                                                        |
   +===========================================================================*/
```

Imports

Per the Oracle corporate Java coding standards, you may not use wide imports (in other words, a package import like `oracle.apps.fnd.framework.*` unless you are referring to a standard Java package). Each class/interface that you require must be explicitly imported. Remove any imports that you do not actually use in the code.

Source Control Header

All files in an OA Framework application must include an ARCS source control header.

You must add the constants **RCS_ID** and **RCS_ID_RECORDED** to all Java files as shown below (these should be added at the top of your class in accordance with the Content Order guidelines below). Remember to import the oracle.apps.fnd.common.VersionInfo class.

Note the standard Javadoc comments used by the OA Framework for these public constants.

```
package oracle.apps.fnd.framework.toolbox.webui;


import oracle.apps.fnd.common.VersionInfo;


public class ExampleClass
{
 /**
  * Oracle E-Business Suite internal source control identifier.
  */
  public static final String RCS_ID="$Header$";

 /**
  * Oracle E-Business Suite internal source control identifier.
  */
  Public static final boolean RCS_ID_RECORDED =
    VersionInfo.recordClassVersion(RCS_ID, "oracle.apps.fnd.framework.toolbox.webui");
}
```

Variable Names

Member variables should have names that are nouns, noun phrases, or abbreviations for nouns.

- Local variable and parameter names should be short, yet meaningful.
- In order to distinguish member variables from variables local to a method, all member variables must be prefixed with the letter "m". All remaining words in the name should have their first letter capitalized. For example:
  private int mMaxLines
- Constant variable names should be in upper case. Different words should be separated by an underscore. Constant names should be descriptive and not unnecessarily abbreviated. For example: MIN_VALUE, MAX_VALUE and BUTTON_SUBMIT_BEAN.

Method Names

The first letter of a method name should be in lower case. All other words in the same name should have their first letter capitalized. For example:

public void prepareForRendering(OAPageContext pageContext);

- Method names should be verbs or verb phrases.
- Methods to get and set an attribute for the variable X should be named getX and setX.

- A method that tests a Boolean condition X about an object should be named isX.
- A method that converts its object to a particular format F should be named toF.
- Whenever possible and appropriate, the names of methods in a new class should be based on names of an existing class that is similar, especially a class from the standard Java Application Programming Interface classes.

Content Order

Contents should be added to Java files in the following order from the top of the page to the bottom.

1. (Required) Oracle E-Business Suite copyright comment
2. (Required) Package declaration
3. (As Needed) Import statements in ascending alphabetical order within each package Packages should be listed in the following order:
    1. Java
    2. Oracle classes/interfaces from outside Applications (BC4J, cabo, JDBC, and so on)
    3. Oracle E-Business Suite AOL
    4. Oracle E-Business Suite OA Framework
    5. Other Oracle E-Business Suite products
    6. Your product
4. (Required) Class/interface Javadoc
5. (Required) Class declaration
6. (Required) ARCS source control headers w/ standard Javadoc comments
7. (As Needed) Public fields w/ Javadoc
8. (As Needed) Protected fields w/ Javadoc
9. (As Needed) Private fields
10. (As Needed) Public constructor w/ Javadoc
11. (As Needed) Protected constructor w/ Javadoc
12. (As Needed) Private constructor
13. (As Needed) Public methods w/ Javadoc *****
14. (As Needed) Protected methods w/ Javadoc *****
15. (As Needed) Private methods *****

*** Note:** For the BC4J classes with generated methods, it is appropriate (and recommended for ease of use) to organize your file so that the methods with code are located at the top, and the template methods are located at the bottom after a delineating comment like the one shown below. Within these two regions, you should organize your methods as described above.

```
/*===========================================================================+
 | Begin BC4J Generated Code Here
 +===========================================================================*/
```

OA Component XML Files

Source Control Header

JDeveloper will add the ARCS source control header automatically when you create page, region, attribute set and package files. The value is stored in a special XML attribute named file-version that can be viewed using the JDeveloper UI Component Property Inspector (the property name is **File Version**).

AutoPatch Driver Commands

Since the XML page definition files are essentially loader files that must be loaded into a database, they require AutoPatch driver dbdrv commands. For these files, JDeveloper automatically adds the dbdrv commands as comments when the files are created. An example dbdrv command for this file type is shown below:

```
<!-- DBDRV: exec Java oracle.jrad.tools.xml.importer.XMLImporter jdk118=y
&fullpath/~PROD/PATH/~FILE &phase=dat+24 checkfile:~PROD:~PATH:~FILE
username=&un_apps password=&pw_apps dbConnection=&jdbc_db_addr userId="1"
rootPackage=/oracle/apps/~PROD rootdir=&fullpath/~PROD/mds -->
```

Region Names

This section describes the naming standards for region IDs *inside* the OA component definition XML file.

Note: As a reminder, region names should not exceed 30 characters, and they should be as short as possible. Abbreviations (which would be understood by consultants and customers) are encouraged.

Generally, region names used as IDs within a file should follow the naming conventions described above for shared region XML file names with the following exceptions:

- The top level page region (which has a style of **pageLayout**) must be named **PageLayoutRN**.
- If the top-level page region has one main content region which holds other content regions and items, this region must be named **MainRN**.
- For pages which include UI abbreviation or icon keys at the top of the page (for example, the "indicates a required field" key), this first key region must be named **KeyRN**. Any subsequent key regions added further down the page should be qualified with its parent region name (for example, **ResultsKeyRN** in a **ResultsRN** region).
- Any contextual information regions at the top of a page should be named **InfoRN**.
- Certain standard navigation regions are given special names: **PageActionButtonsRN**, **TrainRN**
- Regions added only for the purpose of affecting layout should be named with a noun that concatenates a short reference to the parent content region name *if needed*, location indicator *if needed* and a short name of the layout style used with the region. Layout region names shouldn't be suffixed with RN. Location indicators, if needed to avoid confusion, could use relative position indicators such as Top, Bottom, Middle, Left, Right, Center and TopLeft or sequence numbers like in the case of a series of rows. When using a sequence number, the location indicator follows the short name of the layout style like in Row1, Row2 and Row3. See the page hierarchy below the following sample uses cases.

Example Use Cases	Region Names
Employee search criteria	EmpSearchRN
Employee search results	EmpResultsRN
Primary employee information (for update or view)	EmployeeRN
Contextual information region	InfoRN
Employee contact information (for update or view)	EmpContactsRN
Employee direct reports (for update or view)	EmpDirectsRN
Key region added below the EmpResultsRN region.	ResultsKeyRN
Table listing employees	EmpTableRN
Tree listing employees	EmpTreeRN
HGrid listing employees	EmpHGridRN

The following partial page hierarchy illustrates the application of the layout region standard:

```
CustDetailsPG
  PageLayoutRN
    MainRN (default single column header)
      InfoRN (default double column header)
        ... (region items)
      ContactsRN (default single column header)
        ContactsTableLayout (table layout)
          ButtonsRow (row layout)
            ButtonsCell (cell format)
              ContactCreate (submit button)
          SpacerRow (referenced spacer for vertical spacing)
          ResultsRow
            ResultsCell (cell format)
              ContactsTableRN (table)
                ... (region items)
      ContractsRN (default single column header)
        ContractsTableLayout (table layout)
          ContractsButtonsRow (row layout)
            ContractsButtonsCell (cell format)
              ContractCreate (submit button)
          ContractsSpacerRow (referenced spacer for veritical spacing)
         ContractsResultsRow (table layout)
           ContractsResultsCell (cell format)
             ContractsTableRN (table)
               ... (region items)
```

Item Names

This section describes the naming standards for item IDs inside the OA component definition file.

Note: Region item names should also not exceed 30 characters, and they should be as short as possible. Abbreviations (which would be understood by consultants and customers) are encouraged.

The item label (also known as its prompt) is typically a user-friendly name for the underlying data object name, so you should use this label as the item's name. For objects with no label, use the name of the associated object; use a made-up unique name as a last resort to substitute for the label. See the example use cases below.

Warning: Never use the reserved word **name** on its own.

Warning: Item names must be unique within a single page regardless of the region hierarchy. This is an HTML limitation. For now, when you encounter a duplicate, prefix the item name with a short version of the parent region name. For example in an employee search/results page, use **EmpNum** in the search region and ResultsEmpNum in the results region. Please don't use notations like **EmployeeNum** and **Employee_Num** to work around duplicates, because this does not comply with the standard and it's hard to debug if you mistakenly refer to the wrong item.

Example Use Cases	Item Names
A text field with the prompt **Supplier Name**	SupplierName
A text field with the prompt **Salary**, followed by a second text field with the prompt **Salary** under a **Manager** header.	• Salary • MgrSalary
A poplist with the prompt **Employee Type**	EmpType
A display-only column for item description	Description (if no duplicates, otherwise ItemDesc).
An **Apply** button	Apply
A **Go** button, followed by a second **Go** button in the **Results** region.	• Go • ResultsGo

OA Component Packages

For OA components, you have the option of grouping discrete tasks comprised of more than one page in a physical XML file called a *package* (not to be confused with packages corresponding to the directory structure for storing individual files).

Note: Oracle E-Business Suite developers may use OA component packages ONLY for attribute sets. Objects included in OA component packages cannot be personalized.

Attribute Sets

Attribute sets are created in OA component package files named for the associated table as illustrated in the summary table above. Additionally, the following naming standards apply:

- The individual attribute sets should be named for the columns they represent. When derriving the `Document Name` value (attribute set name) from a column name, remove underscore characters and capitalize initial letters. Example: the **PARTY_NAME** column name maps to the `PartyName` Document Name.

  Warning: Never abbreviate the table column name when creating the `Document Name`; reproduce it exactly without the underscores.

  Note: In the OA Component XML files, "Name" is a reserved word. If you have a column named "name," prefix it with the object type to avoid compilation errors. For example, in the ToolBox Tutorial we have a table (FWK_TBX_ADDRESSES) with a column named NAME. The corresponding attribute set `Document Name` for this column is therefore **AddressName**.

- When more than one attribute set maps to the same column, follow the conventions above and add a suffix to the `Document Name`. The suffix is composed of an underscore and the prompt name (or a meaningful abbreviation of the prompt name). Example: for the column **PARTY_NAME** the first attribute set will have a `Document Name` of **PartyName** (note that this should be created for the most popular use/case and prompt). Subsequent attribute sets (if required) will have the `Document Names` **PartyName_Supplier**, **PartyName_Emp** and so on.
- When creating a button attribute set, base the `Document Name` property on the button's label using the same conventions described for column-based attribute sets. For example, a button with the label Add More Rows would have the attribute set name **AddMoreRows**.
- When creating attribute sets for transient UI components (like a search field), the `Document Name` property should convey the meaning/use of the component in accordance with the naming standards described here.

For additional information, see Creating Attribute Sets.

BC4J XML Files

Source Control Header

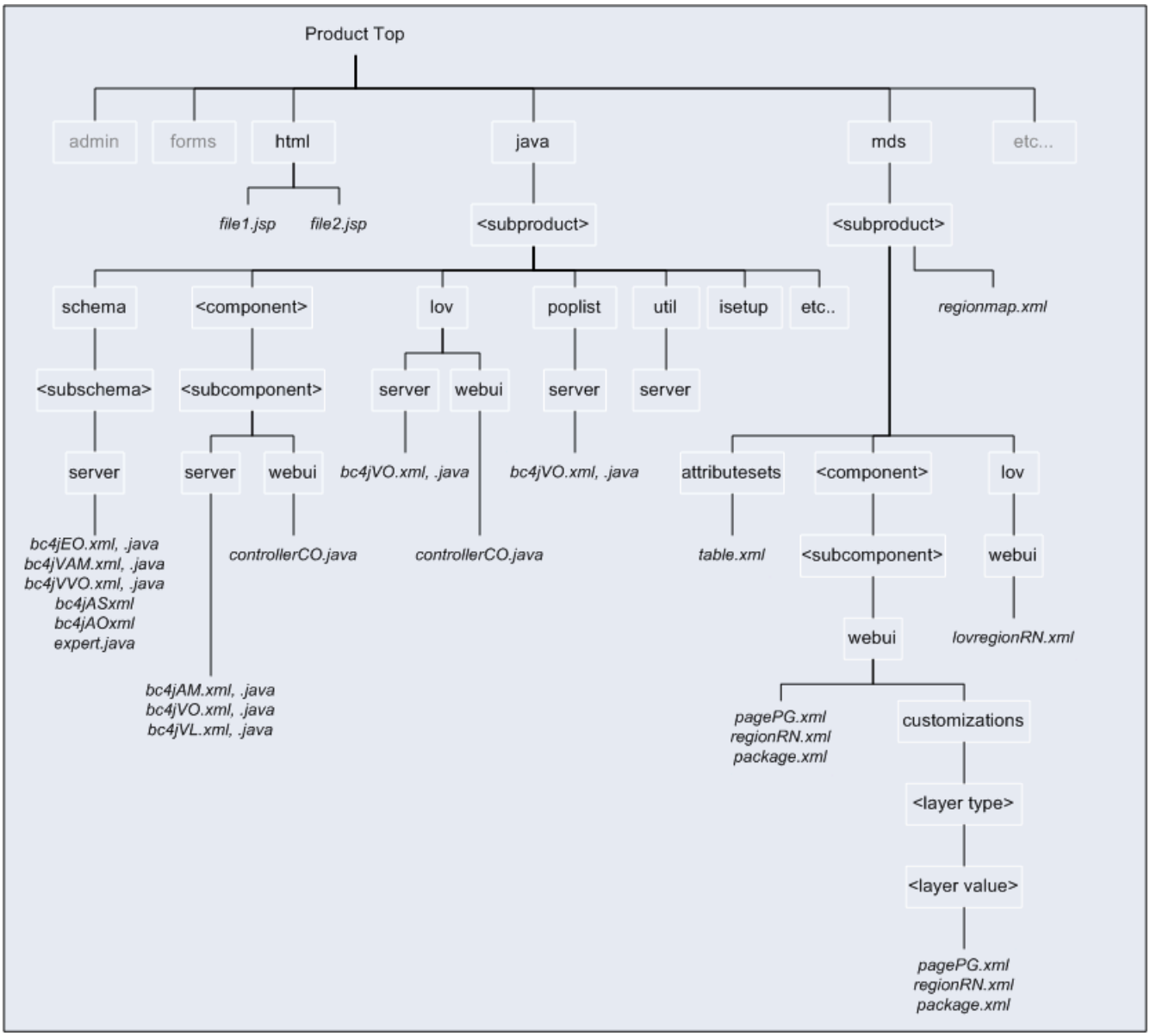
All source controlled BC4J XML files (which currently includes the object metadata files and the package-level server.xml files) require an ARCS source control header. You must add this information at the third line after the DOCTYPE declaration statement using the following format:

```
<?xml version="1.0" encoding='WINDOWS-1252'?>
<!DOCTYPE AppModule SYSTEM "jbo_03_01.dtd">
<!-- $Header$ -->
```

Package / Source Control Directory Structure

When you create and source control your files, you will define packages and directories that comply with structure illustrated in Figure 1 below.

Figure 1: Illustration of OA Framework application file directory structure.



Directory Definitions

..java/*

This directory and its subdirectories contain all the Java and BC4J XML files for your product. These untranslated files will be deployed in the Apps.zip that is created for OA Framework application ARUs.

..java/[<*subproduct*>]

This directory is optional and exists for historical reasons to support product teams that already have a subproduct grouping.

Note: This subdirectory is neither required nor recommended for most product teams.

..java/schema/[<*subschema*>]

This directory is optional and allows for the grouping of all related "schema objects" under a subschema directory (anything related to entity objects). Thus, all your related entity objects (EOs), validation view objects (VVOs) and validation application modules (VAMs) would come under the same subschema folder. Many product teams do not create the optional subschema directories and instead use just the `java/schema/server` directory.

..java/schema/server

This directory contains BC4J XML and Java files for entity objects (EOs), including the EOs themselves, validation application modules (VAMs), validation view objects (VVOs), [entity] association objects (AOs), entity experts and other associated Java files.

This directory does **NOT** hold BC4J XML and Java files for application modules and view objects intended to support the user interface (see .../java/<component>/server).

..java/<*component*>/[<*subcomponent*>]

This level in the directory structure is inteded to define a functional unit. The subcomponent directory is optional; teams are free to define the level of granularity that makes the most sense (if you create subcomponents, however, we recommend that you create subcomponents around small functional units to simplify the logistics around parallel coding and patching).

Example components include:

`.../java/vacancy` for creating and editing vacancies

`.../java/applicant` for displaying/processing applicants

Any classes, interfaces and exceptions which can be used by *both* server and webui components should be included at this level of the directory structure. For example, in the OA Framework, the `OAException` class is included in the `oracle.apps.fnd.framework` package.

Service (application module) interfaces and associated data object and filter implementations would also be included here.

..java/<component>/[<subcomponent>]/test

Includes XML test scripts associated with services. Each `oracle.apps.<product>.<component>.[<subcomponent>]` with a service interface should have an associated `test` directory.

..java/<*component*>/server

This directory contains BC4J XML and Java files associated with functional (user interface) components and services including root (UI) application modules, nested application modules, view objects and view links.

This also includes service application module implementations, domain sets, view objects and view links.

This directory does **NOT** hold BC4J XML and Java files for entity objects and their associated validation application modules, validation view objects, entity experts, and so on (see .../java/schema/server).

..java/<*component*>/webui

This directory contains Java UI controllers associated with functional components.

This directory does **NOT** contain the XML page or region definitions (see .../mds/<component>/webui)

..java/lov/server

This directory contains BC4J XML and Java files (if any) associated with Lists of Values (LOVs).

..java/lov/webui

This directory contains any Java UI controllers associated with LOVs.

..java/poplist/server

This directory contains BC4J XML and Java files (if any) associated with poplists.

..java/util/server

This directory contains miscellaneous utility files referenced by other objects in the server name-space. Typically, these are Java files, however, it may contain BC4J view objects used by the utility to query the database.

..mds/*

This directory and its subdirectories contain all XML page, region, attribute set and package definition files. These files will be translated and deployed separately from Apps.zip. At the customer site, these files will be loaded into a set of mds repository tables. Because the files are loaded into the repository at the customer's site, you are able to create granular patches containing just the affected files.

..mds/[<*subproduct*>]

This directory is optional and exists for historical reasons to support product teams that already have a subproduct grouping.

Note: This subdirectory is neither required nor recommended for most product teams.

..mds/attributeset

This directory contains AttributeSets organized by database table name. See Creating Attribute Sets for additional information.

..mds/<*component*>/webui

This directory contains XML files associated with functional components, tasks or page flows. The `<component>/[<subcomponent>]` name in the mds directory should match the corresponding `<component>/[<subcomponent>]` name in the java directory.

..mds/<*component*>/webui/customizations

This directory contains seeded XML customization files associated with files in the webui directory. See the OA Frameweork Personalization Guide for additional information.

..mds/<*component*>/webui/customizations/<*layer type*>

This directory contains seeded XML page customizations by layer type, which can be either a user or a function name. See the OA Frameweork Personalization Guide for additional information.

..mds/<*component*>/webui/customizations/<*layer type*>/<*layer value*>

This directory contains seeded XML page customizations by layer value, which can be either a function name or the name seededdeveloper. At a customer site the directory name seededcustomer may be created automatically as well. The names of the customized files in is directory should match the corresponding names in the webui directory. See the OA Frameweork Personalization Guide for additional information.

..mds/lov/webui

This directory contains region definition XML files for LOVs.

Example: Differences Between Packages and Source Control Directories

When you define packages for your Java UI controllers, BC4J model objects/metadata, and view metadata files they all begin with oracle.apps.<*product short name*>. For example, a Java UI controller named **MessageSearchCO.java** (for searching in Message

Dictionary) might belong to the following package. Note in this example that "messages" is the component.

oracle.apps.fnd.messages.webui

When it's time to check this file in, it should be added to the directory:

$FND_TOP/java/messages/webui/MessageSearchCO.java

Similarly, the page definition for the Message Dictionary search (named **MsgSearchPG.xml**) would be added to the same package as the UI controller:

oracle.apps.fnd.messages.webui

But at checkin time, it would be added to the following directory:

$FND_TOP/mds/messages/webui/MessageSearchPG.xml

The root application module for the Message Dictionary application (files comprising the application module are named **MessageDictionaryAM.xml** and **MessageDictionaryAMImpl.java**) would be added to following pacakge:

oracle.apps.fnd.messages.server

And checked in to:

$FND_TOP/java/messages/server/MessageDictionaryAM.xml
$FND_TOP/java/messages/server/MessageDictionaryAMImpl.java

The entity object for this same application (**MessageEO.xml** and **MessageEOImpl.java**) would be added to the following package:

oracle.apps.fnd.messages.schema.server

And checked in to:

$FND_TOP/java/messages/schema/server/MessageEO.xml
$FND_TOP/java/messages/schema/server/MessageEOImpl.java

Finally, the validation application module and a validation view object used by the entity objects would be added to the following package:

oracle.apps.fnd.messages.schema.server

And checked in to:

$FND_TOP/java/messages/schema/server/MessageVAM.xml
$FND_TOP/java/messages/schema/server/MessageForCodeVVO.xml
$FND_TOP/java/messages/schema/server/MessageForCodeVVORowImpl.java

In summary, when reviewing the diagram above, remember that the `<product_top>/java` and `<product_top>/mds` source control directories correspond to the `oracle.apps.<product short name>` package name. All subdirectories beneath this level are the same in both use cases.

OA Framework Model Coding Standards

Overview

This document lists the coding/declarative definition standards for the *model* in an OA Framework Model-View-Controller application. It is organized into the following categories:

- Universal (Applies to All "Model" Code)
  - General
  - Performance
  - State Management
- Application Modules
  - Performance
  - Service Interface
- View Objects, View Rows and View Links
  - General
  - Service Interface
  - Performance
  - State Management
  - Internationalization
- Entity Objects, Association Objects and Entity Experts
  - General
  - Performance
- Standard Code/Validation Tasks

Note: Before building your application, please review the corresponding view and controller standards also. These three documents combine with the administrative OA Framework File Standards (Naming, Package Structure and Standard Content) and the Oracle E-Business Suite Java Coding Standards to provide a complete picture of OA Framework coding/declarative definition standards.

Standards Column Description Key

Column Header	Description
No	The standard's number. Note that, for convenience, some standards appear in multiple documents. In these cases, the duplicate references the original standard.
Rule	The standard or guideline content.
Reason	Provides additional explanation for why the standard or guideline is recommended.

Universal (Applies to All "Model" Code)

No	Rule	Reason
General		
M1	Never import any classes or interfaces from the framework.webui.* packages in your server code. For example, importing and using the following is prohibited: oracle.apps.fnd.framework.webui.xxxxx oracle.apps.fnd.framework.webui.xxxxx	The OA Framework is designed to support separate client and server tiers. As such, the code for each of these tiers must remain distinct. See the oracle.apps.fnd.framework package for interfaces which can be safely referenced by code on both tiers.

M121	Use "UTF-8" for encoding attribute in xml tag of BC4J xml files.	Java supports various encodings in two groups. One is Basic encoding set and another is Extended encoding set. Basic encoding set supports some European and American local encoding and Unicode encodings, implemented by lib/rt.jar file. Extended encoding set supports Asian local encodings, vendor-specific encodings, minor European encodings and so on, implemented by lib/charsets.jar. In some customer environments, that do not require extended encoding support, lib/charsets.jar may not exist in the Java class path causing a java.lang.NoSuchMethodError. This standard avoids the error.
M122	Never include schema references in BC4J (View Object, Entity Object, Association Object and View Link) definition files.	Not all customers use the default ORACLE schema names.

Performance

M15	Never use JDBC directly. Always use a view object instead, and the view object should be defined declaratively and not programmatically if possible. If you must call a PL/SQL routine, use the `oracle.apps.fnd.framework.OADBTransaction` to create a `CallableStatement`.	Encapsulation allows us to put in diagnostics, add features, and improve performance in one place, rather than thousands.
M2	If you must issue prepared statements, then call the proper `defineColumnType()` on all the columns in the SELECT with proper sizes.	There are two advantages in doing this. First, this eliminates an extra round trip that the JDBC will need to do to describe the columns. Second, this reduces the prepared statement memory footprint. Without a set precision, for example, all VARCHAR2 columns default to 4KB. **GSCC warning File.java.21**
M3	Always call registerOutParameters() with proper precision for callable statements	This reduces the callable statement memory footprint. Without a set precision, for example, all VARCHAR2 columns default to 32KB. **GSCC warning File.java.22**

State Management

M4

Use BC4J custom properties (those that you create by calling `setProperty(String, Object)`) only for caching objects as a convenience/performance optimization.

M5

Never add member variables to BC4J objects, controllers or any supporting utility classes that you create unless unless they are *transient* or *final*.

Note: For final member variables, it is common practice to also declare them *static* because you only need one instance per JVM for the constant; however, it does not violate the state management coding standard to declare a member variable *final* without declaring it *static*.

If you use transient member variables in your BC4J objects, then you need to clean these values up when the application module is released for use by another thread. To do this, override `OAApplicationModuleImpl.beforeRelease()` or `OAViewObjectImpl.resetSession`() to perform your custom cleanup. For more information, see the corresponding Javadoc for these methods. (Values that you save by calling `pageContext.putTransactionValue()` and `pageContext.putTransactionTransientValue()` from your controller are automatically cleaned up by the OA Framework before the application module release.)

Valid Standard Exception Cases:

- For classes that implement the `oracle.apps.fnd.framework.webui.OAReleaseListener` interface, you can include stateful member variables as long as the class correctly implements the `java.io.Serializable` interface.
- For member variables generated by the BC4J wizards, the OA Framework skips this check because these variables are used for caching purposes.

M67

If you need to manipulate an entity object's state as described in Implementing Java Entity Objects, do not call `setNewRowState(byte state)` on an `OAEntityImpl` instance. Call `setNewRowState(byte state)` on an `OAViewRowImpl` instance instead.

Also see related standard M69.

Application Modules

No	Rule	Reason
Performance		
M6	Do not indiscriminately retain more than one application module while navigating between pages. See State Management in the OA Framework for additional information.	A root application module is tightly coupled with a database connection. Retaining multiple application modules will increase the number of connections required to support multiple users, which adversely effects product scalability.
M7	If you implement the `oracle.apps.fnd.framework.webui.OAReleaseListener` interface to programmatically determine whether an application module should be released, be sure to implement a condition in the release listener under which the AM is actually released.	Otherwise, this is a potential application module memory and connection leak.

M70	For all application modules, enable the Lazy Loading option as described in Implementing the Model (for new application modules, this is enabled for you by default). See Application Modules in Detail for additional information about this feature, including usage considerations. **Tip:** For existing application modules that were created before this feature was introduced, you may use a mass conversion script.	The Lazy Loading option defers instantion of view instances and nested application modules until they are needed. This is more performant than choosing to instantiate all referenced objects at AM instantiation time regardless of whether or not they are needed.
Service Interface		
M74	For Service Interface only: When you define the data model for your service application module, you must specify a **singular** view instance name, without a VO or SVO suffix. For example, if you select the SuppliersSVO to add to the SupplierSAM data model, you must rename the view instance to Supplier, rather than Suppliers or SuppliersSVO.	This is required to support the generation XML Schema per changes for the new Service Data Object (SDO) standard.

View Objects, View Rows and View Links

No	Rule	Reason
General		
M11	If a WHERE clause should always be enforced, such as the multi-org security WHERE clause, create that clause as part of the view object at design time.	No user of your view object can override/replace that WHERE clause, although they can add their own supplemental conditions.
M12	You should NOT generate an `oracle.apps.fnd.framework.server.OAViewObjectImpl` unless you intend to specialize its behavior or write code to initialize/execute the view object's query.	
M13	Encapsulate the WHERE clause and bind parameter settings or SQL modification logic in a custom method named initQuery() (or a variant if you have several of these: initManagersQuery(), initNewEmployeesQuery(), initAllEmployeesQuery() and so on).	
M14	Trivial view objects (like those used for poplists, Lists of Values and validation view objects) should be based on a plain SQL statement, and not on entity objects. A LOV view object must have a primary key associated with it. All other view objects (including read-only view objects) should be based on entity objects *unless* you specifically want to avoid the inherent coordination benefits whereby changes to an entity object are reflected in all the view objects that reference it. **Note:** For entity object-based view objects, the OA Framework recommends -- but does not require -- the use of associations for references objects. They facilitate view object and view link creation.	There are several reasons for this: • It is necessary if you are performing DML • Multiple view objects based on the same entity object can automatically see updates made by any one of them • At some point, if the view object is large enough with enough foreign keys, you might see more efficient memory usage

M66

Always use unique bind targets in your SQL statements, even if they bind to the same value. Always use ascending bind targets, and do not skip numbers in the sequence.

Note that this assumes you are using Oracle-style binding as required in OA Framework Model Coding Standard M23.

This is actually a JDBC coding standard violation that may work when you initially code it, but may fail at the customer site (particularly when personalizations are applied).

Incorrect

```
select emp_name from employees
where emp_id = :1
and manager_id = :1

select emp_name from employees
where emp_id = :1
and manager_id = :1
and deptno = :4
```

Correct

```
select emp_name from employees
where emp_id = :1
and manager_id = :2
and deptno = :3
```

M31

You must expliclity initialize all view objects that are not used to query data from the database to suppress any efforts by BC4J to query the database.

For detailed instructions on how to address this in different scenarios, see View Objects in Detail: Initialization Guidelines.

This unwanted query may result in a SQL exception due to unbound WHERE clause parameters, or the loss of newly inserted rows that are not properly activated.

M71

When accessing an application module, use `getApplicationModule()` instead of `getRootApplicationModule()` whenever possible.

Also see guideline C11 in the Controller Coding Standards.

This ultimately improves modularity and reuse.

Service Interface

M72

For Service Interface only: If a service view object supports partial failure, then the service view object can only be based on one updateable entity object.

Changes posted in a service view object, are actually posted in the underlying entity object. However, if a post to an entity object fails, then the corresponding view object row needs to be reverted. If the view object is based on more than one updateable entity object, it becomes more complicated to revert all entity objects if only some entity objects fail.

M73	For Service Interface only: If a row modifies other rows that are neither its parent nor its children, then the corresponding SVO is is recommended not to support partial failure. If the SVO has to enable partial failure, then: 1. The API needs to make sure that all the changes made to the other rows are reverted back before throwing an exception. 2. If the modification of other rows occur in the validation cycle, then the modifications should be addressed after validation. This avoids the situation where a row fails and is rolled back to its original state, but the other rows are left in an inconsistent state. 3. If an entity needs to modify other entities that are neither its parent nor its children in the commit cycle, the modifications should be done after the entity itself has been posted successfully. This requirement avoids the situation where an entity modifies other entities, but the entity itself fails later and is reverted leaving the other entities in an inconsistent state. 4. If the modification of other rows occur in the validation cycle, and the underlying entity object fails during `postChanges` or commit, then the underlying entity object should revert all changes made to the other rows before throwing the exception.	If a row makes changes to other rows, and the row fails and needs to be reverted, then changes made to all the other rows also need to be reverted. Since OA Framework does not know how to revert the other rows, your API needs to take care of this before throwing an exception.
M75	For Service Interface only: When you define the properties of your service view link, you must specify **singular** accessor names for the view link source and destination. For example, when defining a SuppliertoSitesSVL, specify the Source accessor name as Supplier and the Destination accessor name as SupplierSite.	This is required to support the generation XML Schema per changes for the new Service Data Object (SDO) standard.
Performance		
M16	Always create view objects declaratively if you can. Create view objects programmatically ONLY if you CANNOT create them declaratively. **Note:** If you must create a dynamic view object, we recommend that you use an `oracle.apps.fnd.framework.server.OAViewDef` instead of a SQL statement to define the view object. The `OAViewDef` is a more performant option, and it allows you to create an "OA" view object which is essential for UI component binding. If you create a dynamic view object from an `oracle.apps.fnd.framework.server.OAViewDef`, you must follow the usage rules documented in the `OAViewDef` Javadoc.	Dynamic view objects have to be described by BC4J through an additional execute call for the VO prepared statement, and they potentially result in non shareable SQL.

M17	If you must programmatically create a view object or view object attribute: • Do not create dynamic view objects without a name (and do not use a null name like ""). • Make sure you avoid a naming collision by checking to see whether a VO or attribute of that name already exists or before trying to create it. • Be sure to remove any view objects which are no truly longer required (however, you should use AM.findViewObject() to reuse one that has already been created; don't recreate it unnecessarily as the VO is cached on the AM and automatically cleared when the AM is released).	Naming the dynamic view object gives you the ability to look it up next time you need it instead of recreating it.
M18	Avoid creating generic view objects just so you can reuse them in different places. Create dedicated, task-specific view objects instead that are as small as possible.	The "one VO fits all" approach adds complexity to your SQL, increases number of attributes and VO memory footprint. You should consider view objects to be UI/task-specific.
M19	If you create your view object declaratively, try to avoid changing the WHERE clause definition or order by clause at runtime. If you can, create multiple definitions of the same view object, each with a different WHERE clause (note that it is appropriate to change the WHERE clause if you have complex search criteria).	These actions invalidate the cached statement and require reparsing.
M20	Always generate an oracle.apps.fnd.framework.server.OAViewRowImpl subclass for the view objects (VOs) that you create. Furthermore any code you write to call accessors on the view row should use the named accessors (for example, getSupplierName()) on the VO row instead of the generic accessors.	Accessing the VO attributes through the index is significantly faster; especially for VOs with a large number of attributes (10+). • Avoid using getAttribute("<name>") which performs a expensive scan to lookup the index.
M21	If possible, define your view object as "Read Only" and "Forward Only." Note that this standard makes the most sense for view objects that are not tied to the UI.	No validation and smaller memory footprint. Prevents BC4J from buffering all of the result set in the middle tier.
M22	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.	This improves performance by reducing the primary key comparison logic for lookups, and it also reduces the cost of encryption when each primary key value has to be encrypted in your application for security purposes. The also reduces HTML footprint, since the primary key is sent to the browser.
M23	ALWAYS use Oracle Positional style binding (:1, :2) in your SQL. Do NOT use ANSI JDBC-binding (?) or Oracle-Named binding styles.	This avoids parsing SQL at runtime to do String replacement. It's also necessary for any view objects with a DATE value in the WHERE clause due to a mismatch between the Java and Oracle Date formats when the month format mask is MON. Due to that, to_date() function might return a error. **Warning:** ANSI JDBC-style binding will be desupported. **Proposed GSCC Warning File.Java.24**

M24	Always use bind variables for any values that are passed from the client. Do not use literal values (for example, do not simply concatenate a value to the WHERE clause).	
M25	Always make sure that precision is specified for all String columns in the view object (including both declarative and programmatically defined view objects). Furthermore, the precision needs to match the database column - if you change the column size or generate the VO against an invalid database, the VO definition can be out of synch. **Note:** This is specified automatically for any VO defined since Oracle JDeveloper.	This reduces the view object memory footprint. Without a set precision, for example, all String columns default to 4KB in size each.
M26	Do not execute searches by default nor allow blind queries. Remember that the **FND: View Object Max Fetch Size** profile option is set to "200" by default for Oracle E-Business Suite development. Even with search criteria, you cannot query > 200 rows without first presenting your case to the Applications Performance Tuning team for approval.	
M27	Do not use select* in the view object definition.	Data model changes will break the code.
M28	For view objects which are not bound to a UI component, fetch only the number of rows you want to display; no more. Set the VO prefetch size to an appropriate value. **Note:** This is handled for you automatically for view objects used with UI components (for example, if the VO is serving up rows to a table displaying 10 rows, the prefetch value is set to 10).	
M29	Avoid calling vo.getRowCount() to perform a simple row existence check. For VOs that you query, use vo.first() or vo.hasNext() and check for a null result. For VOs that you populate programmatically without performing a query, you could also use: `if (vo.getFetchedRowCount == 0)`	getRowCount() causes the entire view object row set to be fetched to middle tier. **GSCC Warning File.Java.19**
M30	Never call VO.getEstimatedRowCount().	getEstimatedRowCount() executes select count(*) on the VO. This count is not guaranteed to stay the same, thus the method name. **GSCC Warning File.Java.20**
M32	This standard has been revised for clarity and moved to the controller coding standards for a better fit with the revised content. See OA Controller Coding Standard C32.	
M33	Use view links carefully (note that they are required when performing DML on master/detail entity objects related by a composite association, and by certain UI components like the Tree and HGrid). In cases where they are not required, do not use them if you don't want to buffer data for the life of the transaction.	View links buffer data as you navigate from master to master. If practical from a design perspective, code the master on one page and the detail on a second page (pass the search criteria on the request).
M34	When creating a RowSet or RowSetIterator make sure you name and close them when you're done using closeRowSet() and closeRowSetIterator() respectively.	You can (and should) reuse them using the ViewObjectImpl.findRowSetIterator() method.

M36

Tune your SQL!!!! Fix all SQL statements with:

- Full table scans/full index scans
- Hash joins
- Non-mergable views
- Parse time > 0.3 seconds
- Sharable memory >= 100K

Do not add decode or nvl conditions in WHERE clauses to control join conditions or filters.

See Enabling a Database Trace in the Testing OA Framework Applications chapter for instructions on checking your OA Framework SQL statements.

If the SQL is poorly written, response time and scalability will be adversely affected.

Additional information on SQL tuning for all audiences is also available in your Oracle 10*g* Database documentation (also published on the Oracle Technology Network site).

M61

If you have a dynamic view object created from an `OAViewDef`, and if no database query is needed for the view object (you typically call `ViewObjectImpl.setMaxFetchSize(0)` for these view objects), then make sure to set the expert mode (full SQL) property to true with `OAViewDefImpl.setExpertMode(true)` or `OAViewDefImpl.setFullSql(true)`.

If the `FullSql` property is false (the default), then BC4J framework tries to build (derive) the SQL from the EO attributes. Setting FullSql to true optimizes the performance and allows BC4J to save the VO state properly.

State Management

M37

Whenever you dynamically set a WHERE clause or bind WHERE clause parameters, always clear any possible preexisting values *before* executing the query as shown:

```
vo.setWhereClause(null);
vo.setWhereClauseParams(null);
...
vo.executeQuery();
```

Note: Upon setting up query criteria and executing the view object query, make sure your view object query criteria remains consistent with the executed query. In other words, do not change or reset query criteria unless the new criteria will be consumed within a request (for JSP forward case, before the page boundary is reached).

The preexisting WHERE clause parameters must be cleared to avoid exceptions like "java.sql.SQLException: ORA-01008: not all variables bound", in case the previous where clause had more parameters.

The view object query criteria must remain consistent with the executed query because an inconsistent view object state could cause undesired side effects. Note that OA Framework has the right to call ViewObject.executeQuery() for a view object that is already prepared for query execution, and this may not be what you intended if you redundantly set up query criteria and, for instance, cleared the view object cache.

M38

Don't just assume your view object exists, or that it contains the expected data; you could get a NullPointerException. Always check for null and handle this case.

M39

All view objects must have a primary key (this is essential for UI components bound to view objects to work properly).

If you want to perform DML operations (insert, update, delete) on a view object row, always use the primary key instead of the row index. In a Real Application Cluster with database partitions, the same database row can have a different ROWID in different partitions. Similarly, if a table is partitioned, the ROWID can change inside a transaction if the row is migrated. This leads to problems if you rely on the ROWID as your primary key.

See the View Object topic in Implementing the Model for information about defining primary keys.

Exception: This isn't necessary for view objects that truly do not have a logical primary key (for example, a view object that simply selects the sum or average of something).

M65

When you iterate view object rows to perform some operation on selected rows, be sure to evaluate all the view object ranges, and not just the current view object range.

When BC4J rebuilds a view object during activation, row positions may change as the refreshed result set may include newly inserted rows in the database and/or reflect the removal of deleted rows in the database. As a result, the rows that were in the displayed range may no longer be in the current range after activation.

M69

Whenever you create and insert a new row into a transactional view object, always immediately set its row state to **STATUS_INITIALIZED** after you perform the row insert. View objects rows should not appear to have pending changes immediately after they are created and defaults are applied. For example:

```
Row row = vo.createRow();
vo.insertRow(row);
row.setNewRowState(Row.STATUS_INITIALIZED);
```

Exception: If you explicitly require that the row remain "dirty" so it is posted and committed regardless of further user changes, you may avoid making this state change.

Note: You cannot use this method for `OAPlsqlViewObjectImpl` view objects (they are not based on entity objects, and ultimately, BC4J manages this state at the entity object level). Please see the `OAPlsqlViewRowImpl.setNewRowState()` Javadoc for additional information about controlling the state for this special case.

Also see standard M67.

Doing this ensures that rows that have not been edited by the user are not validated or posted to the database (even if you set defaults for the row). For example, if you have a table of expenses that is prepopulated with 5 "temporary" rows, but the user only needs to record 2 expenses, you want the 3 untouched rows to be ignored when you validate / commit the changes. Setting the row state as described here achieves this.

Note this also ensures that "temporary" rows do not trigger the save model warning message if the user tries to abandon the transaction without specifying any values for these rows.

Internationalization

M41 If you want to use a date value in your WHERE clause, you must convert a String date to a Date object using OANLSService.stringToDate(), and then use oracle-style bindings to set VO's where clause with DATE data (don't try to use a to_date function with a String directly in the WHERE clause).

Java and Oracle abbreviate months differently so user to_date in the WHERE clause of a VO returns an error.

M42	Don't use `substr(column)` in the select of a view object. You should use substrb or if you must use substr, multiply the length in the XML by 4.	The database uses byte semantics, but substr is measured in characters. The value in the view object XML is essentially the byte length allowed, which breaks down when running against a multi-byte database.

Entity Objects, Association Objects and Entity Experts

No	Rule	Reason
General		
M43	All validation that is generally applicable to a given table should be captured in the entity object (which can delegate to its entity expert and validation view objects). Do NOT code validation/business logic in view objects or application modules unless you face a special case that precludes the use of entity objects (or PL/SQL entity objects if necessary).	Since all database access to that table will occur through this entity object, using entity object's validation only needs to be defined a single place in the code. Concurrent programs, HTML and Java UIs, and any external programs will all interact with a given table through it's EO, and will all be forced to use the EO's validation.
M44	When you define an entity object, include all table columns. **Note:** For Release 12.2, make sure the entity object omits the `ZdEditionName` attribute, as it is only for internal use by Online Patching.	
M45	Base your entity objects on the _ALL synonyms rather than on organization-restricted views.	Entity objects need full, unrestricted table access to do validation. For example, if your business rule tests that supplier name is unique, your EO will need to query across all organizations to confirm uniqueness. It is not sufficient to check within the current organization.
M46	Always use named setters/getters for all EO attributes. All column-level validation should be written in the named setters.	This yields a number of benefits: • Single point of validation logic is the setter in the EO. • Downstream users can override or extend the validation logic by overriding the setXXX method for a given attribute, and know that they can always call the super.setXXX to extend the existing logic. • Compile time type checking.
M47	Cross-attribute validation (within a single entity object) must be written in the validateEntity() method (or in other entity event points like `beforePost()` as needed -- see Implementing Entity Objects for standard validation patterns). DO NOT write any cross-attribute validation in the attribute setters.	You have no control over the order in which setters are called, so you cannot be sure of the attribute state for any attribute EXCEPT for the one you are currently setting. Therefore, you cannot write reliable cross-attribute validation in the setter.
M48	Entity objects have a create(attributeList) method which is called automatically by the BC4J framework when the entity is instantiated. All defaulting should be done in this method. For example, you can assign default values to attributes in your entity.create(attributeList). You typically assign primary key sequences in the create method.	

M50	Do NOT use BC4J validators	Placing logic in validators which are then only specified in the XML metadata breaks object encapsulation, and is hard to debug.
M51	Do NOT use BC4J custom domains.	This breaks the entity encapsulation, the exceptions are thrown in a disassociated fashion from the entities, and you can't control the exception content that is displayed to the user.

M52	Avoid calling `OADBTransaction.executeCommand("rollback / commit")` *for Java entity objects*.
	
	Remember that the OA Framework already handles commit/post failure when working with entity objects; you do not need to issue a rollback. When you want to commit (or if you think you need to execute a rollback manually) for your entity objects, use `OADBTransaction.rollback()` and `OADBTransaction.commit()`. Furthermore, do not attempt to perform a rollback or clear BC4J caches from within an entity object; all such logic should be written at the application module/transaction level (see Inappropriate Validation Failure Handling in Java Entity Objects).
	
	Note it is acceptable to use this for PL/SQL transactions (using a `CallableStatement`) that do not involve entity objects, however, you should always use this in conjunction with a save point as shown:
	
	`/* Use a product prefix in the savepoint name to avoid name collisions as shown for "PO" */`
	`txn.executeCommand("SAVEPOINT poset_context");`
	`// Issue your PL/SQL callable statement.`
	`...`
	`if (<error condition detected upon post and before commit>)`
	`txn.executeCommand("ROLLBACK TO SAVEPOINT poset_context");`
		When you call `executeCommand("<command>")`, this leaves the BC4J entity cache out of synch with the database.
		
		Calling the rollback without the savepoint for your PL/SQL callable statements could lead to invalid cursors.

M53	When you initialize an attribute value in your entity.create(attributeList), call the attribute setter method (set<AttributeName>).	This is equivalent to calling setAttribute("<AttributeName>", val) but the performance is slightly better as the lookup step is bypassed. You can optionally use the setAttributeInternal() method to skip the custom validation in the attribute setter method. setAttributeInternal() will still fire declarative validations.

M54

All Java entity objects *except* the ones subclassing OATLEntityImpl must have the following attributes to support standard WHO handling (the OA Framework will automatically maintain these values).

- CreatedBy
- CreationDate
- LastUpdatedBy
- LastUpdateDate
- LastUpdateLogin

If your entity object does NOT have the standard WHO attributes, then just provide a no-op implementation for the standard setters.

Read more about WHO columns in Java entity objects, and in PL/SQL entity objects.

Performance

M49

Children entity objects in composite entity associations should not set the parent's foreign key attribute values.

These entity objects can expect that their parent primary key attribute values are passed through the attributeList parameter in create(attributeList). The call to super.create(attributeList) will populate these foreign key attribute values. Repopulating the foreign key attribute values unnecessarily has a performance cost.

M56 Use (Java) entity objects for DML. Use PL/SQL entity objects if you absolutely cannot move to Java entity objects.

- Automatically issues updates, inserts and deletes
- No need to use callable statements (which are always reparsed)
- Executing DML through PL/SQL results in added round-trips as compared to native DML through BC4J
- Supports batching; much more efficient than calling PL/SQL APIs row by row:
  - Reduces round trips
  - Executes bulk DML

M57

Create an entity expert for each standalone entity object, and for the root-level entity object in a composition. Also create a "validation application module" for these same entity objects. Add basic validation methods to the entity expert that client entity objects can use. These basic validation methods should make use of "validation view objects" added to the corresponding "validation application module."

See additional information about creating entity experts, validation application modules and validation view objects.

Any foreign key objects that you access for validation will be instantiated, which can be quite expensive for large objects. You should not instantiate foreign key objects unless you need to call additional methods on these classes. For example, if you just want to find out if a particular supplier is valid, there is no need to instantiate a supplier entity object. The entity expert singleton solution offers a lightweight alternative.

M58

Use declarative "validation view objects" for lightweight validation (or query) SQL that you need *within* an entity object. Do NOT write JDBC code or create programmatic view objects.

See additional information about creating entity experts, validation application modules and validation view objects.

See the view object performance guidelines above for information about JDBC and programmatic view objects.

M59	Do not access foreign key entity objects indiscriminately. Instead, call lightweight validation methods on the foreign key entity object's entity expert as described above.	
M63	Set the `Change Indicator` property if you have an OBJECT_VERSION_NUMBER (or equivalent) column in your table. Note that change indicators for all updateable EOs must be included in your query.	
M68	Enable batch updates for your entity objects by selecting the `Use Update Batching` property on the Entity Object Wizard **Tuning** tab. Also set the `Threshold` property to **1**. This is particularly important for _TL multilanguage entities. **Note:** there are several cases where this feature cannot be used: • You override the DML (PL/SQL entity objects are in this category) • You have one or more streaming attributes (CLOB or BLOB). • You do not have a primary key. • One or more attributes is marked as retrieve-on-insert or retrieve-on-update	When enabled, BC4J combines multiple DML operations and executes them in a single round trip. Modified rows are grouped into batches, each with a maximum number of records less than or equal to the `Threshold` value and executes each batch in one round trip.

Standard Entity Object Validation Patterns / Examples

For examples of standard validation patterns and examples, see Implementing Entity Objects.

OA Framework View Coding Standards

Overview

This document lists the general coding/declarative definition standards for the *view* in an OA Framework Model-View-Controller application. It is organized into the following categories:

- General
- Attribute Sets
- Performance
- Back Button / State Management
- Internationalization
- Component-Specific Rules
  - Page Layout
  - Table Layout
  - Tables
  - Lists of Values (LOVs)
  - BI Beans
  - Form Parameter / Form Value
  - Poplists
  - Images
  - Search
- Accessibility
- Parameters

Note: Before building your application, please review the corresponding model and controller standards also. These three documents combine with the administrative OA Framework File Standards (Naming, Package Structure and Standard Content) and the Oracle E-Business Suite Java Coding Standards to provide a complete picture of OA Framework coding/declarative definition standards.

Standards Column Description Key

Column Header	Description
No	The standard's number. Note that, for convenience, some standards appear in multiple documents. In these cases, the duplicate references the original standard.
Rule	The standard or guideline content.
Reason	Provides additional explanation for why the standard or guideline is recommended.

General

No	Rule	Reason
V1	Always define your UI declaratively. Resort to programmatic definition ONLY if your design CANNOT be implemented declaratively.	Programmatically created web beans cannot be personalized, reused or extended easily. See the topic, Relationship Between Controllers and OA Personalization Framework for more information. Furthermore, some hand-coded layouts may stop complying with the BLAF UI Guidelines over time.

V2	Each page must have AT MOST one form bean. Only your **pageLayout** region should have its `Form` property set to **Yes**.	Multiple form beans on a page are not supported and can result in odd runtime behavior. **Note** the Developer Mode check throws a "warning" if there are multiple form instances in OA Framework page. The parsing is done on OA Framework web bean tree so it won't throw a warning for any forms outside the web bean tree to support the "embedded" mode.
V3	IDs must be unique for each web bean on a page. **Note:** This is essential for correct PPR behavior.	The OA Framework assumes that each component in the web bean hierarchy (in a page) is uniquely identified by its ID.
V4	Once you ship a product to a customer, DO NOT change the IDs of your web beans, or change the document the item resides in (in other words, don't break a region out into a subdocument).	The ID is the item's primary key. Even if you don't have any references to this item, any personalizations (or translations) on the item will be broken if the key changes.
V5	When extending objects, you should only reference them at the document-level. In other words, don't try to extend a region contained within another page. If a region should be shared, it should be created within its own XML file. **Note:** Within a package, you may extend a shared region that is shipped with the package.	
V27	If you dynamically change a page's UI when the user presses a Go button, you must enable PPR for the driving component (poplist, radio button and so on) and remove the Go button. Do not introduce a Go button for this purpose with any new pages. **Note the this standard does not apply to Go buttons used to initiate a query in a Search page. All Search page Go buttons must remain for explicit user selection.** See the Dynamic User Interface document for additional information.	This ensures a consistent experience for the user.
V33	All regions in your page must have the `Add Indexed Children` property set to **True**.	Doing this ensures that your page is personalizable / extensible.
V38	Do not reference a region more than once in the same page, that is, no two regions in a page can extend the same region.	When two regions extend the same region, the two extended regions will have web beans with duplicate IDs. This violates the coding standard V3, which states that IDs must be unique for each web bean in a page.
V41	The unique ID for each web bean on a page may not contain more than two underline '_' characters.	A web bean ID containing more than two underline characters is invalid and will be flagged as not unique. For example, a web bean ID of "Table___1" is invalid and should be changed to "Table__1" or "Table_1".

Attribute Sets

No	Rule	Reason
V6	You must create an attribute set package for each table your product team owns. This attribute set package should include: • (**Required**) An attribute set for each *displayable* column (_TL table translatable columns should be included with the base table). You may create multiple attribute sets for a single column if it has multiple usages. • (**Required**) An attribute set for each lookup code column in your table (in place of the corresponding display value). For example, if you have a column called FOB_CODE, you need to create an attribute set with a matching name as you would for a regular display column. • (**As Needed**) An attribute set for each *reusable* action button for the data in this table (prompts should comply with the UI guidelines). Do NOT create buttons for common actions like **Go**, **Apply**, **Cancel**, and so on as they are published in the following common attribute set package: oracle.apps.fnd.attributesets.Buttons • (**As Needed**) An attribute set for each *reusable* header related to the presentation of data in this table (text should comply with the UI Guidelines). If you have commonly used attributes that have no associated base table, you may also create a special attribute set package named **Transient** to include associated attribute sets. See Creating Attribute Sets in Chapter 3 for instructions on how to create attribute sets. Note that the templates of required attribute set properties in *Creating Attribute Sets* should be considered part of this coding standard. See OA Framework File Standards (Naming, Package Structure and Standard Content) for information on naming conventions (including mapping multiple attribute sets to a single column).	The reuse of UI elements provides a significant cost savings both to Oracle and its customers. Oracle will save a great deal in translation and maintenance costs. Customers will be able to make quick and global personalization of the E-Business Suite (EBS) UI. Additionally, having fewer UI elements translates to smaller middle-tier memory consumption (better performance).
V7	Before creating a new attribute set, ensure that no other corresponding attribute set exists per the guidelines above. All duplicates must be approved by the OA Framework team.	
V8	As a developer, you MUST use attribute sets whenever possible (for transient attributes like search criteria, try to use the attribute set for the associated data item). See the Implementing the View: Attribute Sets document for additional information on using attribute sets.	
V9	When using attribute sets, you should generally avoid overriding attribute set translatable properties as this defeats their purpose.	
V36	For foreign key column references, you should leverage the column attribute set associated with the base table. If an additional attribute set is warranted for generic usage (because other product teams may have a need for it), then the base table owner is required to make that attribute set available. If an additional attribute set is warranted for a specific usage (for a single product team), then the consumer product team can add the attribute set in the table that has the foreign key reference.	

Performance

No	Rule	Reason
V11	Page response time should be < 5 seconds.	
V29	Blind queries are not allowed. All Search regions must have: a) a default query that is tuned with built-in query constraints not specified by the user or b) at least one user-specified search value (which would result in a performant query) must be designated as being "selectively required" See the Simple Search and/or Advanced Search sections in the Chapter 4 Search document for additional information about how to implement this in a query bean region, or in a manually constructed Search region.	

Back Button / State Management

No	Rule	Reason
V14	If you want to add hidden fields to a page (whose values will be added to the request in a form submit), add **formValue** items; do not use **formParameter** items. Remember to encrypt these values if they are sensitive (users can see the values if they opt to view the page source).	
V15	If you need to manually submit a form (for example, when a user selects an image or link), do NOT use a Javascript `submitForm` function. Instead, you should configure a **fireAction** event to declaratively submit the form (see the Declarative Submit Form documentation for instructions).	

Internationalization

Alignment

No	Rule	Reason
V16	Always specify layout component alignment as "Start" and "End" (never "Left" and "Right"). If you specify alignment programmatically in a layout bean, always use (`OAWebBeanConstants.H_ALIGN_START` and `OAWebBeanConstants.H_ALIGN_END`). **Tip:** If you need to right-align number values in a form bean, set the bean's `CSS Class` to **OraFieldNumber**.	Components will align properly in a bi-directional session.

Maximum Length Property

No	Rule	Reason

V39	Always specify a value for the `Maximum Length` property of OAMessageTextInput when OAMessageTextInput is bound to a database column. The maximum length value of OAMessageTextInput must equal that of its corresponding VO attribute.	The OA Framework length validator is invoked only when the `Maximum Length` property of OAMessageTextInput is set. Without the validator, non-ASCII text insert/update to the database may cause an unexpected exception and/or result in string truncation.

Translation

In general, you should give well-formed, meaningful, translatable text values for all properties that will be displayed in the UI. The text should clearly convey its intent (as concisely as possible!) to both the user, and the translation team so they can property translate it.

The following illustrates both correct (well-formed, translatable) and incorrect examples:

Example Context	Correct (Translatable)	Incorrect
Text field prompt	Supplier Number	SUPPLIER_NUM
Button short description *	Select to save this purchase order.	SELECT TO SAVE
Button label	Apply	APPLY
Region prompt (see below)	Status	STATUS_FLOW_LAYOUT_REGION
Form value prompt (see below)	Delete Event	_DELEVENT

*** Note:** The `Short Description` property is used as the ALT text, meaning it's used by assistive technologies and displayed as tooltip text when the mouse moves over region items like buttons and images. This property is not relevant for regions. See the Accessibility section for additional information.

While it's self-evident that a text field prompt and a button label (to name just a few cases) should satisfy the criteria described above, there are several cases where it's easy to overlook text values which must also comply with these guidelines. In each of the following special cases, *if you specify any non-null text value*, this value must be translatable.

Default Single/Double Column Regions

Rule	Reason
The `Text` property must comply with the Translation guidelines described above.	Even if the `Header Disabled` property is set to **True**, the `Text` value (if specified) is always translated because it could be toggled to display at runtime.

Content Container Regions

Rule	Reason
The `Text` property must comply with the Translation guidelines described above.	When specified, this value is used as the **contentContainer's** header text. Even if you have code that sets this value programmatically, if specified declaratively, the `Text` property is always translated.

Form Value Region Items

Rule	Reason
The `Prompt` property must comply with the Translation guidelines described above.	Even though **formValue** items are hidden and never displayed to the user, the `Prompt` value is used as a label in the message box if an item-level message (error or warning) is associated with the **formValue**. For example, your hidden primary key may have a unique constraint violation when set.

Valid Table Child Regions

The following region types can be nested beneath a table region (a data table, not a table layout):

- advancedSearch
- flowLayout
- hideshow
- query
- rowLayout
- stackLayout
- switcher

If you add any of these region types beneath a table, the `Prompt` property must comply with the translation guidelines described above. This value is used as a table column header if any of these regions is added as an indexed child of a table. Since regions can be shared, this value must be set properly even if you think you're creating a region that won't be used in a table.

Component-Specific Rules

In addition to the broad rules described above, you must comply with the following detailed rules when working with specific components.

Page Layout

No	Rule	Reason
General		
V18	If a page is intended to be "bookmarkable" (or if a shared region might be used in a bookmarkable page) its `securityMode` property must be set to **selfSecured**.	Pages cannot be accessed across user ICX sessions without following these insructions.
V31	If a page is updateable, set its `Warn About Changes` property to **True** to ensure that users do not inadvertantly lose pending changes when they try to leave the page. See the Save Model document for additional information including how to disable this property for selected page items.	This is required by the BLAF UI Guidelines.

Table Layout

No	Rule	Reason
General		
V40	1. tableLayout must always have only rowLayouts or message web beans as its children. 2. rowLayout must always have cellFormats as its children. Furthermore, rowLayouts always need to be under a tableLayout. 3. cellFormat can have any other web bean except rowLayout and cellFormat under it.	Any variation of these guidelines can lead to cases where a browser may not render the page correctly. For example, fields in the layout get misaligned or render with incorrect spacing. The following error may also result when submitting the page: "You are trying to access a page that is no longer active. - The referring page may have come from a previous session. Please select Home to proceed."

Tables

See the View Objects, View Rows and View Links standards in the corresponding "Model" document for information related to query execution.

No	Rule	Reason
Performance		
V19	Number of rows displayed should be kept to 10 (this is the default display value). 25 rows is the maximum.	Fewer rows increases response time.

Lists of Values (LOVs)

No	Rule	Reason
Performance		
V20	Group all *related* LOV view objects in a single package into one application module. In this case, "related" means associated with the same product or logical application within a product area. You should also consider packaging and patching realities when choosing your structure.	This loads view definitions (which are static in the JVM), so view objects can be shared between users.
V21	If the potential number of displayed values is small (see the UI Guidelines for a specific recommendation), use a poplist instead of an LOV.	The LOV requires several trips to the database while a poplist -- particularly if using cached values -- is more efficient.
V28	Unless an LOV field allows partial values (in a "Search" region, for example), you must enable validation (also known as "autocompletion"). See List of Values document for additional information.	This ensures a consistent experience for the user.

BI Beans

No	Rule	Reason
Performance		
V22	Use PNG instead of JPEG format for graphs.	Faster response time and less memory consumption (factor of 3x).
V24	Do NOT render graphs on high-traffic pages.	Latest performance benchmark: a graph consumes 4-5MB.

Form Parameter/Form Value

No	Rule	Reason
State Management		
V25	Never add a **formParameter** bean directly to your page; these may be added only by the declaratively configured **fireAction** and **firePartialAction** events. Use the **formValue** for hidden field values.	See Supporting the Browser Back Button for additional information.

Poplists

No	Rule	Reason
Performance		
V26	Use poplists with caching for static lists of data. **Note:** These caches are striped by language, organization, and bind variables. If your values can vary across other parameters, you should turn off the OA Framework caching.	This is cached on the JVM and shared.

Images

No	Rule	Reason

General		
V30	Whenever you explicitly add an image to your page (for example, a **Delete** icon in a table), you must specify its `Height` and `Width` properties. If you add an image programmatically, call its `setHeight()` and `setWidth()` methods.	Browsers can more efficiently allocate space for images not yet downloaded. This is particularly important for application access via low bandwidth networks (for example, wireless access). Also, if size tags are not present when using a low bandwidth network, the page may wiggle around while loading.

Search

No	Rule	Reason
General		
V34	You must populate the `Securing Function` property for any "Oracle-seeded" view you create, and grant this function to the appropriate users/responsibilities. You must also document these secured views in their corresponding User's Guides.	Personalization administrators at the customer site can use this feature to selectively hide or show these "Oracle-seeded" views for their users.

Accessibility

Note: You must also test your product for accessibility compliance as described in Testing OA Framework Applications.

Regions

You must specify any "Required" region properties for your product to be accessible.

Region Style	Additional Text	Prompt	Window Title	Row Header Col	Explanation
table hGrid	Required				`Additional Text` (shortDesc) must be set because it acts as the table summary.
table advancedTable				Preferred	Set the `Row Header Col` (String) property to the ID of the column web bean that should act as the row header column. The underlying UIX `oracle.cabo.ui.beans.table.TableBean` supports accessibility by defaulting a table's first column as the row header. However, you can set a different column as the row header, especially if a user is likely to personalize the table by turning off rendering of the first column or if the first column can return no data. For classic tables, you may only set an immediate child under the table as the row header column, and not a recursive child item under an immediate child.
pageLayout			Preferred		`Window Title` must be set to identify the browser window, but setting the `Page Title` is acceptable.
switcher		Conditionally Required			If used as a table column, then the prompt is displayed as the column header and must be specified.
advancedSearch					No special actions required.
tree					No special actions required.

Items

You must specify any "Required" item properties for your product to be accessible. If an item style has "Preferred" and "Available" properties (as opposed to a single "Required" property), you must implement one or the other.

Item Style	Additional Text	Prompt	Text	Explanation
List (as a shuttle subelement)	Required			The Additonal Text serves as the label for the list element. This is normally set the same value as the Available / Selected Header for the shuttle.
messageCheckBox messageChoice messageTextInput messageFileUpload messageLovInput messageRadioGroup messageRadioButton	Available	Preferred		`Additional Text` is set as the `TITLE` HTML attribute and the `Prompt` is set as the `LABEL FOR` HTML attribute. If the `Prompt` is null, then `Additional Text` is also used as the `LABEL FOR` attribute in accessible mode.
button submitButton	Available	Preferred		If `Additional Text` is set, then this will be read as the `ALT` text for the element. If the `Additional Text` is not set, then the `Prompt` will be read as the `ALT` text. **Note:** If the prompt is meaningful on its own, do not specify `Additional Text` if it is redundant. Specify `Additional Text` only if additional information is required.
exportButton resetButton	Available		Preferred	If `Additional Text` is set, then this will be read as the `ALT` text for the element. If the `Additional Text` is not set, then the `Text` value will be read as the `ALT` text. **Note:** If the Text value is meaningful on its own, do not specify `Additional Text` if it is redundant. Specify a `Additional Text` only if additional information is required.
image	Required			An image must always have `ALT` text associated with it. If the image is purely decorative, then this alt text must be an empty string which you can set by entering a value of **DECORATION** in the `Additional Text` property. **Note:** When using a bound value for an Image at runtime you must ALWAYS bind the ShortDesc also - a change in image requires a change in the `ALT` text!
formattedText	Available (currently not read by screen readers)		Required	The `Text` property must contain only 100% W3C compliant HTML (no missing closing tags and so on). The `Additional Text` (shortDesc) is shown as the HTML `TITLE` attribute.

rawText			Use this web bean with caution; the OA Framework does not validate your HTML. • Customers may follow the Section 508 Standards and the Web Content Accessibility Guidelines (WCAG).
messageStyledText	Conditionally Required		If used as a table column, then the prompt is displayed as the column header and must be specified. **Note:** if using as a link programmatically (so a destination is set), then the `SHORT_DESC_ATTR` must be set to the `TEXT_ATTR` as shown in the code example below.
staticStyledText			No special actions required.
spacer tip formParameter formValue			No special actions required.
link		Conditionally Required	If the `Icon URI` property is set on the link web bean, then the `Text` property is required to make it accessible.
attachmentLink attachmentImage messageInlineAttachment	Available		If no prompt is specified for attachmentLink or messageInlineAttachment, then no default prompt appears. If no prompt is specified for attachmentImage, then the prompt defaults to "Attachments".
attachmentTable		Available	Text appears as the display name for the Attachment table.

Note: In any layout in which prompts and form fields are not associated with one another, you must specify `Additional Text` (shortDesc) values for the fields.

Code Example: Setting the SHORT_DESC_ATTR

```
OAWebBean desc = table.findIndexedChildRecursive("Desc");
desc.setAttributeValue(DESTINATION_ATTR,
   new OADataBoundValueViewObject(desc,"ItemDetailLink", voStr));

//Insert the following line to set the title attribute to the text of the item
desc.setAttributeValue(SHORT_DESC_ATTR, new NodeAttributeBoundValue(desc, TEXT_ATTR));
```

Parameters

No	Rule	Reason
General		
V37	If you are creating any new parameters in your application for passing via the request or through the URL, then you must avoid the parameters already used by OA Framework as listed in the Oracle Application Framework URL and Request Parameters appendix.	Using parameter names that conflict with the parameters used internally by OA Framework can lead to unpredictable failures in several web beans.

OA Framework Controller Coding Standards

Overview

This document lists the coding standards for the *controller* in an OA Framework Model-View-Controller application. It is organized into the following categories:

- General
- Performance
- Back Button / State Management
- Internationalization
- Component-Specific Rules
  - Table
  - Form Bean
  - Dialog Page
- Accessibility

Note: Before building your application, please review the corresponding model and view standards also. These three documents combine with the administrative OA Framework File Standards (Naming, Package Structure and Standard Content) and the Oracle E-Business Suite Java Coding Standards to provide a complete picture of OA Framework coding/declarative definition standards.

Standards Column Description Key

Column Header	Description
No	The standard's number. Note that, for convenience, some standards appear in multiple documents. In these cases, the duplicate references the original standard.
Rule	The standard or guideline content.
Reason	Provides additional explanation for why the standard or guideline is recommended.

General

No	Rule	Reason
V1	Always define your UI declaratively. Resort to programmatic definition **ONLY** if your design **CANNOT** be implemented declaratively.	Programmatically created web beans cannot be personalized, reused or extended easily. See the topic, Relationship Between Controllers and OA Personalization Framework for more information. Furthermore, some hand-coded layouts may stop complying with the BLAF UI Guidelines over time.

C1

If you must create web beans programmatically, never instantiate them using their constructors (`new OA*Bean()`). Always use the `createWebBean()` factory methods published on the `oracle.apps.fnd.framework.webui.OAControllerImpl` class.

Note: Specifically, you should use always use the `createWebBean()` methods that let you specify a component `ID`.

Some bean properties aren't initialized correctly when you use the constructor.

You must specify a component ID because:

- This is the only way to identify components that are PPR targets, or are referenced by Javascript.
- The OA Framework assumes that IDs are the same between requests, and this is true only if they are explicitly assigned by you (either programmatically or declaratively).
- You can use the ID further down in the bean hierarchy processing to call findIndexedChild or findIndexedChildRecursive.

V3

IDs must be unique for each web bean on a page (NEVER programmatically create multiple items with the same item name).

Note: IDs should comply with the OA Framework File standards, which include naming standards that help address the question of uniqueness. If you're creating a shared, reusable region, be very careful to use ID values that would work in any page that includes the region.

The OA Framework assumes that each component in the web bean hierarchy (in a page) is uniquely identified by its ID.

Failure to assure this can lead to subtle, idiosyncratic behavior that is difficult to debug since, even if the items aren't "rendered" at the same time, they both exist in the web bean hierarchy.

C2

Never add the same web bean instance twice to a region. For example, do NOT do the following:

```
OAWebBeanContainer region = ...


OAWebBean someItem = ...
region.addIndexedChild(someItem);
region.addIndexedChild(someItem); // Not allowed!!
```

M1

Never import any classes or interfaces from the `framework.server.*` packages in your UI controller.

For example, importing and using the following in a controller is prohibited:

oracle.apps.fnd.framework.server.OADBTransaction
oracle.apps.fnd.framework.sever.OAViewObjectImpl

The OA Framework is designed to support separate client and server tiers. As such, the code for each of these tiers must remain distinct.

See the oracle.apps.fnd.framework package for interfaces which can be safely referenced by code on both tiers.

C3 There are two way in which you can invoke typed methods:

- You can generate an application module interface so you can invoke typed methods directly (with compile-time checking) instead of calling invokeMethod(). Refer to Generating Application Module Interfaces for additional information.
- You can avoid referencing any model objects (*except* for application modules which should always be referenced using the interface `oracle.apps.fnd.framework.OAApplicationModule`) in your UI controller. For example, if you need to call an initQuery method on a view object to execute a query, call a corresponding method on the associated application module which can, in turn, call initQuery on the view object. Consider the following example where the application module has a search(String name, String number) method that calls initQuery on an "employees" view object.

```
import java.io.Serializable;
import oracle.apps.fnd.framework.OAApplicationModule;
...
processFormRequest(OAPageContext pageContext,


                   OAWebBean webBean)
{
  ...
  OAApplicationModule am =
    (OAApplicationModule)pageContext.


       getApplicationModule(webBean);
String empName = pageContext.getParameter("empName");
String empNum = pageContext.getParameter("empNum");
Serializable params = { empName, empNum };
am.invokeMethod("search", params);
...
```

The following part of this rule is a **Guideline**:

Furthermore, the methods that you create in the application module should be named to correspond with the UI action/event (for example: search(), apply(), deleteOrder(), init(), init<*Page*>(), approve(), hire(), `createEmployee()`, and so on).

Warning: Be careful not to inadvertently override methods in the base class. For example, do not create a method named `create()` to handle a "Create" button press in your page since the application module already includes a `create()` method. Use the more explicit `create<Object>()` instead to avoid a conflict.

This coding style will be easier to covert to a ServiceBean architecture, and promotes code reuse.

C4 Never modify or manipulate parent/grandparent web beans from child/grandchild controllers.

This is a poor design practice that hampers reuse while introducing fragile code (particularly if the child code executes too late in the page rendering cycle to properly affect the parent). Furthermore, this can incur costly overhead if the parent/grandparent bean rendering preparation must be repeated.

C5 Never use index numbers to find web beans when you want to change their properties. Always search by bean name.

Index numbers can change during processing; they aren't a reliable mechanism for identifying individual beans.

C6	If you have a requirement to modify web bean properties in `processFormRequest()` or `processFormData()`, make sure you either: • Bind the property to some implementation of `oracle.apps.fnd.framework.webui.OAAbstractBoundValue` in `processRequest()` (preferably `oracle.apps.fnd.framework.webui.OADataBoundValueViewObject`) and then modify the underlying view row attribute value to ensure that the property value changes. or • Explore declarative binding using SPEL as outlined in the Changing UI Properties section of the Dynamic User Interface document. In particular, since SPEL is supported on the rendered, disabled, readOnly and required properties, if you want to change any of these properties in `processFormRequest()` for any web bean, you must use SPEL. **Note:** The old standard of writing event handling logic in processFormRequest() to redirect back to the same page is deprecated.	This ensures that the web bean hierarchy is in a stable state that ensures back-button compliance when it renders.
C7	Javascript is prohibited without explicit permission from the OA Framework team, and verification with the corporate UI team that it is essential for your product design. • If you need to make images, links and so forth submit the form when selected, you may configure a **fireAction** event (see Declarative Submit Form for additional information). You may *not* use the formerly approved UIX `submitForm` Javascript for this purpose; this is no longer a valid exception to the "no Javascript" rule. **Note:** If you have permission to ship custom Javascript, all Javascript APIs must be in separate JS files. Do *not* add them to the HTML content (this keeps the page definitions as small as possible, and the JS file itself will be fetched only once).	Javascript needs to be certified on different platforms and it increases the page size.
C8	For web beans that provide multiple versions of a method such as setText(String text) and setText(OAPageContext pageContext, String text), always select the one that has the OAPageContext parameter.	Access or modifications to OAWebBean objects must be performed in a thread-safe manner, and the use of an OAPageContext ensures this.
C9	Don't programmatically depend on "implicit" structures created by the OA Framework. Doing a findChildRecursive() to find beans within these structures is allowed.	Several beans - defaultRenderers, PageLayoutBean, and TableBean - create additional beans when prepareForRendering is called. These beans may change between versions; you cannot reliably parse this web bean structure.

C10	Always check the request parameters for the event you want to handle. Don't simply assume that your processFormRequest() code is being executed because of an expected form submit.
	
	Incorrect Code
	
	`processFormRequest(...)`
	`{`
	`  super.processFormRequest(...)`
	
	`  // Your event handling code...`
	
	Correct Code
	
	`processFormRequest(...)`
	
	`{`
	`  super.processFormRequest(...)`
	`  // Always check for the event you want`
	`  if (pageContext.getParameter("Go") != null)`
	
	`  {`
	`  // Your event handling code...`
		The OA Framework may issue form submits that you aren't expecting, and your code would be executed at an inappropriate time.

C11	When accessing an application module, use `getApplicationModule(webBean)` instead of `getRootApplicationModule()` whenever possible. *Also see guideline M71 in the Model Coding Standards.*	This ultimately improves modularity and reuse.
C12	When forwarding to the first page of a new flow (a page directly attached to a menu), always use a menu function instead of a direct page reference. For subsequent pages, you may navigate using the page name.	This allows you to change destination by changing Function definition. No code change or recompilations.
C31	Always call `super.processRequest()`, `super.processFormRequest()` and `super.processFormData()` first in each of the respective methods as shown in C10 above.	OA Framework will be relying on these calls to super -- particularly in processFormData.

Performance

No	Rule	Reason
C13	Avoid multiple levels of forwarding (and avoid querying needlessly when forwarding). See the View Objects, View Rows and View Links standards in the corresponding "Model" document for additional information relating to query execution.	

C34	If you need to get or set attribute values, use the accessor methods with the following signature:
	
	`setAttributeValue(AttributeKey attrKey, Object value)`
	
	`getAttributeValue(AttributeKey attrKey)`
	
	Do not use the signatures that accept a String as the atrribute key identifier.
	
	Tip: The list of available `AttributeKeys` is defined in `OAWebBeanConstants`.
		Using the `AttributeKey` constants greatly reduce the synchronization overhead, thereby improving scalability.

Back Button / State Management

For additional information, see Supporting the Browser Back Button.

No	Rule	Reason
C15	Any information used to alter beans when rendering a page must be added to the request as a URL parameter. See Supporting the Browser Back Button.	If the OA Framework must rebuild the web bean hierarchy, this ensures that it can be done correctly.
M6	Do not indiscriminately retain more than one application module while navigating between pages. See State Management in the OA Framework for additional information.	A root application module is tightly coupled with a database connection. Retaining multiple application modules will increase the number of connections required to support multiple users, which adversely effects product scalability.
C17	Do not set your pages to expire as a way of handling the Back button without reviewing your case with the OA Framework team. Otherwise, avoid this completely.	
C19	This standard has been removed as it is covered in C20.	n/a
C20	To ensure that your pages gracefully support browser Back button navigation, you must comply with the published instructions for correctly communicating state between pages: See Supporting the Browser Back Button for specifics.	
C21	Standard has been removed as it is no longer applicable.	n/a
M5	Never add member variables to your controller unless they are transient or final. See the M5 standard for additional information.	
C24	Standard has been removed as it is no longer applicable.	n/a
C32	Avoid *unconditional* view object and transaction state initialization in your `processRequest()` methods. For detailed instructions on how to satisfy this requirement in your code, see Supporting the Browser Back Button: Avoid Unconditional view Object/Transaction State Initialization.	The `processRequest()` method may be re-entered to recover a lost web bean hierarchy, or to synchronize client UI state with middle tier web bean state. It is important that any initialization logic that you implement in your `processRequest()` methods anticipate this possibility to avoid unexpected application behavior. For example, if you unconditionally execute a query in a view object that is used for updates, you may inadvertently lose the user's pending changes.

Internationalization

No	Rule	Reason
C25	Use `OAUrl.encode` to encode URL parameters. Do NOT use `java.net.URLEncoder` or `java.net.URLDecoder`.	To handle non-ASCII characters based on the encoding of the URL, developers should use `OAUrl.encode()` and `OAUrl.decode()` which encodes/decodes based on the ICX_CLIENT_IANA_ENCODING profile option.

V16	If you specify alignment programmatically in an `oracle.apps.fnd.framework.webui.beans.layout.OACellFormatBean`, always use "Start" and "End" (`OAWebBeanConstants.H_ALIGN_START` and `OAWebBeanConstants.H_ALIGN_END`) instead of "Right" and "Left."	Components will align properly in a bi-directional session.

Component-Specific Rules

In addition to the broad rules described above, you must comply with the following detailed rules when working with specific components.

Table

No	Rule	Reason
General		
C27	Never access the table's single or multiple selector as an indexed child.	During `prepareForRendering()`, OA Framework converts the selector from an indexed child to a UIX named child (which is how it should be accessed).

C28	To enable sorting in a table with a selector:
	
	Define a transient view attribute for the selector item using the BC4J view object wizard in JDeveloper. Do not add a dynamic attribute by calling `ViewObject.addDynamicAttribute()`. In your `OAViewObjectRowImpl`, override your `set<SelectorAttributeName>()` method as shown below:
	
	`public void setSelectFlag(String val)`
	`{`
	`  populateAttribute(getAttributeIndexOf("SelectFlag"), val);`
	`}`
		By default, when the selector item is checked in the UI, the underlying view object attribute value is updated. This action leaves the view object with pending changes (even for a read-only table). When the view object is in this state, table sorting is disabled. You must follow these steps to set the selection without making the view object "dirty."

C33	Avoid redundant query execution for tables. For detailed instructions on how to address this in different scenarios, see View Objects in Detail: Initialization Guidelines.	The OA Framework needs an event point for applying personalization changes to a view object, and as a consequence, will execute the query against a table's VO when it gains control after the developer's code executes. This guideline is intended to help you avoid redundant query execution.
C35	Code event handling for table rows in a controller associated with the table region.	In an advanced table under a layout region, a queued event may throw an error in some cases. To mitigate this issue, code any event handling pertaining to table rows in a separate controller associated with the table.

Form Bean

No	Rule	Reason
General		
V2	Never create an `oracle.apps.fnd.framework.webui.beans.form.OAFormBean` programmatically.	Your page should have only 1 form, and that should be specified declaratively for the **pageLayout** region. **Note:** The Developer Mode check throws a "warning" if there are multiple form instances in OA Framework page. The parsing is done on OA Framework web bean tree so it won't throw a warning for any forms outside the web bean tree to support the "embedded" mode.

Dialog Page

No	Rule	Reason
State Management		
C29	Standard has been removed as it is no longer applicable.	n/a
C30	Always configure the dialog page buttons to POST if you want to perform an action in the calling page (for example, committing a transaction).	

Accessibility

See the View Coding Standards for individual bean attributes which must be specified to satisfy accessibility requirements.

Oracle E-Business Suite Java Coding Standards

Overview

This document describes general Java coding standards that apply to any Java code you write. It is organized into the following categories:

- General
- Performance
- Internationalization
- Patching Binary Compatible Java

If you need information about file naming conventions, standard file contents/organization, or directory (package) structures for any OA Framework application file types (including Java files), see OA Framework File Standards (Naming, Package Structure and Standard Content).

For information about coding standards (including performance guidelines) related to specific areas of OA Framework application coding, see the Model, View and Controller coding standards. For example, for all the standards related to the definition and implementation of view objects, see the OA Framework Model Coding Standards.

Standards Column Description Key

Column Header	Description
No	The standard's number. Note that, for convenience, some standards appear in multiple documents. In these cases, the duplicate references the original standard.
Rule	The standard or guideline content.
Reason	Provides additional explanation for why the standard or guideline is recommended.

General

No	Rule	Reason
A1	You should have only one top level "outer" class per Java file. Inner classes are permitted, but remember that the concatenated name (combination of the outer and inner class names) can't exceed 50 characters.	The Java Release Infrastructure (JRI) requires each Java file to have a valid package declaration and to contain exactly one top-level class declaration.
A2	Public constants should be used sparingly, and should never be changed after code is shipped.	The Java compiler does not force recompilation of files that reference constants when those constants change (the Java compiler optimizes reference to public static final variables to their actual value in the code that refers to them). Rather than reuse a constant name, create a new constant to maintain backward compatibility with existing constants. For all classes that have an Interface defined, all public constants should be declared in the interface as opposed to any implementation of that interface.

A3	Member variables may have the public, protected, private or default visibility, however, member variables that are "public" should usually be made private with public accessors defined for them: `setXX()`, `getXX()`	It is a poor design practice to publish public member variables allowing direct access by other classes. This is fragile, and it breaks the object's encapsulation. If the accessors are overridable, the class code must use the accessor instead of using the private member variables directly. This will allow subclasses to easily add behavior to the accessors.
A4	Avoid client-side Java (Swing, EWT, applets and so on)	All client-side Java in Oracle E-Business Suite is planned for replacement. No new client-side Java code should be written without senior management approval.
A5	Never use Java in the database.	No robust patching infrastructure, undesirable overhead of supporting another JVM and performance.
A6	Never catch and "swallow" exceptions unless you can fix the cause of the exception, and no user action/notification is required.	While writing initial code, it is often tempting to "swallow" exceptions (catch an exception and do nothing with it, or simply write a message to the console). It is very likely that if you do this frequently, you will overlook some catch block, and fatal exceptions could be thrown that are not handled properly.
A23	Do not use the browser cookie to store state information.	Browser cookies are reserved for OA Framework internal use only. They add request bloat, and there is a corporate initiative to reduce cookie usage.

Performance

No	Rule	Reason
A7	Use a `StringBuffer` for concatenation.	Strings are immutable objects, so when used for concatenation, unnecessary temporary objects will be created.
A8	Size your `StringBuffer` properly.	If initial StringBuffer size is not enough, the original char[] will be discarded and moved to a newly allocated one, thus creating unnecessary temporary objects.
A9	Use `String.equals()` for String equality testing.	This defaults to object reference equality and then to char by char comparison.

A10	Do not use exceptions for regular code path execution. For example, some OA Framework developers have used `ViewObjectImpl.findAttributeDef()` which throws an exception when the attribute is missing. Instead, you should use the JDeveloper method `ViewObjectImpl.lookupAttributeDef()` which returns null when the attribute is missing.	To fill the stack trace for an exception, the JVM has to deoptimize the code to report a correct trace.
A11	Size your collections properly.	This avoids resizing at runtime and rehashing for hashtables and hashmaps.
A12	Do not use synchronized collections for thread local or read only objects. For example: 1. Avoid using Vectors for objects that do not need thread safety (that is, only used by one thread at any point of time). Use ArrayLists in this case. 2. Avoid using Hashtable for objects that do not need thread safety (that is, only used by one thread at any point of time). Use HashMaps in this case.	Synchronization adds unnecessary overhead in this case since it needs to acquire the monitor, execute the method and release the monitor.
A13	Minimize the size of synchronized code blocks.	This reduces the wait time for other threads contending for the same code block.
A14	Avoid writing finalizers. Finalizers are unpredictable, often dangerous and generally unnecessary. They make it hard to debug memory leaks, prolong the lifetime of an object and place extra burden on the garbage collector.	This prolongs lifetime of an object and places extra burden on the garbage collector since it runs the finalizer.
A15	Do not call `System.gc`.	It should be left to the discretion of the JVM to run garbage collection. **GSCC Error File.Java.8**
A16	Do not use your own debug classes. Use the standard `writeDiagnostics()` method.	
A17	Do not call debug routines (or write diagnostics) without first checking if debug/diagnostics is enabled for the level at which you want to record a log entry. Avoid adding *too many* calls to the debug class. In this case, "too many" is is proportional to the amount of work the method is doing. For example, logging in a given method should not be doing more work than the method itself (this usually happens when extensive logging is added during development to debug some error, and never removed).	Logging overhead is significant (even if the debug mode is silent). You should check that logging is enabled at a given level, because we recommend to customers that the highest (most restrictive) level of logging (unexpected exceptions) always be turned on so that OAM can collect information on these errors.
A18	Avoid calling `System.err.println` or `System.out.println` in the normal code path execution. Use standard debugging or logging instead.	Developers often forget to remove these calls which can slow the system significantly. There is only one output stream and one error stream in the JVM. Multiple threads calling those methods have to be synchronized, and will potentially block. Also in production (iAS 1,0.2.2.2), the std out is not redirect to a file, and is lost. **GSCC warning File.Java.17.**

A19	Avoid doing a lot of string manipulation in PL/SQL when you can do it in Java instead.	Multibyte string manipulation in PL/SQL is very slow.
A20	Use Buffered I/O classes such as `BufferedReader` and `BufferedWriter`.	I/O devices are optimized to work with bulk data. For example, without buffering, each invocation of a print method on a writer would cause characters to be converted into bytes and written immediately to the file, which is inefficient.

Internationalization

No	Rule	Reason
A21	Do not use resource bundles. Use Message Dictionary to programmatically obtain translated values for any strings that will display in the UI. Note that declarative data intended for UI display (OA Framework UI component XML metadata, menu definitions, profile options and so on) are all directly translatable.	Resource bundles cannot be used across technology stacks and are not as reliable / available as data in the database.

A22	Do not use Java APIs that rely on the platform encoding or locale.
	
	Bad Code Examples:
	
	`byte[] bytes = ....`
	`new String(bytes)`
	
	`byte[] b = str.getBytes();`
	
	`OutputStream out = ...`
	`new PrintWriter(out)`	

Patching Binary Compatible Java

Standard (New Code)

Any of the following changes to shipping code impact link compatibility of referencing classes. *If you do not have a complete account of the products or code that may be referencing your classes*, you should NEVER make any of the following changes in an ARU.

Select a rule link below for additional information. See Sun Microsystems' Patching Binary Compatible Java document for a detailed list of their rules for ensuring binary compatibility.

Rule	Reason
Changes to a method signature	• Changing the return type or parameter type requires redelivery of the associated class. • Changing the return type from String to void, even if no classes use the return type, requires redelivery of the associated class. • Changing a type to a super class (or to a subclass, interface, implementation class) also requires redelivery.
Changes to final variable values	Constant field values (String constant, primitive type constants) are in-lined into the classes. You must recompile and redeliver all the classes that use this constant.
Changes to a field signature (type, modifier)	You must recompile and redeliver all classes that reference this field.
Change a static method to an instance method or vice versa	You must recompile and redeliver all classes that call this method.

Change of a method location within a class hierarchy (specifically, if you move it *down* the hierarchy)	You must recompile and redeliver all classes which reference the method you move.
Change to an interface definition	If you change an interface, any class implementing or referencing the interface change will need to be recompiled and redelivered.

Chapter 9: Extending and Deploying OA Framework Applications

Deploying Extensions

As explained in earlier sections of this Developer's Guide, *extensibility* refers to the ability to programmatically extend an application's functionality, while *personalization* refers to the ability to declaratively alter the UI. For a detailed description of how to deploy personalizations, refer to the chapter on deployment in Oracle Application Framework Personalization Guide.

The Extending OA Framework Applications document describes how you can add custom business logic to the products Oracle E-Business Suite ships with. The document describes how to deploy such customer-built extensions to your Release 12 environment, as well as what you need to do to deploy custom OA pages you have developed to add functionality to an existing OA Framework-based application.

Additional Information: For additional steps on how to deploy your custom extensions in conjunction with the Release 12.2 Online Patching feature in Release 12.2 and above, refer to Section 6.3 of My Oracle Support Knowledge Document 1577661.1, *Deploying Customizations in Oracle E-Business Suite Release 12.2*.

The discussion below assumes that you have set up your development environment as described earlier in this book. The setup steps should have resulted in a working development environment. You may wish to verify this by testing your setup. The following documents how to use this environment to deploy an extension created in Oracle JDeveloper to your Release 12 environment.

Deployment at a Glance

Business components you extend consist of the following:

- XML files that provide the declarative properties for your extended component.
- Possibly one or more Java files where you have overridden methods from the base class for the component, to provide custom business logic programmatically.

At design-time, Oracle JDeveloper reads the component's declarative metadata definition from its respective XML file that resides on your file system and runs the associated Java classes from your JDEV_USER_HOME. As explained in the substitution instructions under the topic Extending OA Framework Applications, the substitutions you specify corresponding to your extended BC4J objects are stored in the **.jpx** definition file in your Oracle JDeveloper project.

For run-time use, the BC4J XML for your extended business components and the corresponding compiled Java code will need to be deployed to the file system of your application tier server. The substitutions specified in the **.jpx** definition file will need to be deployed to the MDS repository on the database of your target Release 12 instance.

Custom OA pages you develop consist of the following;

- XML files that define the OA components and declarative properties specified in your pages
- One or more OA Extension controller Java files

To deploy a custom OA page, the OA component definitions (in XML files) will need to be deployed to the MDS repository on the database of your target Release 12 instance. The corresponding OA Extension Controller classes will need to be deployed to the file system of your Release 12 application tier server.

In addition, you will need to deploy any custom BC4J components that model data for your custom page. These BC4J XML files and the corresponding compiled Java code will need to be deployed to the file system of your application tier server. If you have extended existing BC4J components (shipped with an OA Framework-based self-service product) for use with your page, you will also need to deploy the corresponding substitutions specified in the **.jpx** definition file of your Oracle JDeveloper project. You deploy the **.jpx** file to the MDS repository on the database of your target Release 12 instance.

Note: OA Framework does not currently support the use of the BC4J deployment wizards provided with Oracle JDeveloper. Although Oracle JDeveloper uses a local OC4J instance to allow you to test your work locally, the Release 12 technology stack employs an Apache JServ Servlet engine to process servlet requests, and does not currently use or support OC4J.

Use the instructions in the following section to deploy the constituent parts of your extension or custom OA page to your Release 12 environment. Oracle recommends that you always deploy your extensions to a test instance first and verify the changes before deployment to a production server.

Deploying Your Business Logic Extensions

Note: The following instructions assume your extensions comply with the naming standards described in Extending OA Framework Applications.

Step 1: Compile your Java in Oracle JDeveloper and zip up your Java classes

Create a zip of `<JDEV_USER_HOME>\myclasses\<CompanyIdentifier>`, electing to preserve the directory structure. You will be picking up both BC4J files and MDS XML in your zipped file.

Note: Remember to remove any compiled JSPs from your zip as compiled JSPs will be retrieved from the server. Also remove .jpx files from your zip, as they are not needed in the deployed environment. (Oracle JDeveloper copies the jpx file into the myclasses directory upon project compilation.)

Expand your zip file to the middle-tier of your Release 12 instance under `$JAVA_TOP`, ensuring that the directory structure of your packages is preserved. Your directory structure should now resemble the following:

`<$JAVA_TOP>/<CompanyIdentifier>/oracle/apps/<AppsProductShortName>/server` (BC4J files)

So, if you extend a "PO" view object, and your company's name is "Abc," a subclass of the `RequisitionsVOImpl` view object would be in the following directory:

`<$JAVA_TOP>/abc/oracle/apps/po/server/AbcRequsitionsVOImpl.java`

Set the permissions on `$JAVA_TOP/<CompanyIdentifier>` using: `chmod -R 775 <directory_name>`

Note: Depending on the packages you have created and the naming conventions you have used, you may have additional directories in your tree.

Step 2: Copy the .jpx definition file to the $APPL_TOP staging area

Copy your **.jpx** file, which should be located under `<JDEV_USER_HOME>\myprojects`, to the following staging area:

```
$APPL_TOP/<CompanyIdentifier>/<AppsProductShortName>/<product-version>/java/
```

Step 3: Run the jpx import utility to import substitutions specified in the .jpx definition file to the MDS repository

The import utility, **jpximport.bat** or the **jpximport** shell script is located under the `jdevbin\oaext\bin` directory of your Oracle JDeveloper installation area. You run this utility from the command line by specifying:

- The database credentials of the MDS repository on your target instance
- The fully-qualified location of your **.jpx** file, which should be located under `$APPL_TOP/<CompanyIdentifier>/<AppsProductShortName>/<product-version>/java/`

Running the utility without specifying any parameters will display its usage options and format information.

The import utility parses your **.jpx** XML file for specified substitutions, and transforms each substitution into a separate site level customization document which is then imported into the MDS repository. For example, assume that you are following the Extensibility Standards described under the section on Extending OA Framework Applications. Using the example on substituting the PoSummaryVO view object from the OA Framework ToolBox Tutorial `ExtendLabSolutions.jpx`, your command line would look something like the following:

```
java oracle.jrad.tools.xml.importer.JPXImporter
$APPL_TOP/mycompany/fnd/12.0.0/java/ExtendLabSolutions.jpx
-username <username>
-password <password>
-dbconnection "(description = (address_list = (address = (community = tcp.world)
(protocol = tcp)(host = <host.example.com>)(port = <port#>)))(connect_data = (sid = <SID>)))"
```

The MDS reference to this imported document would be:

```
/mycompany/oracle/apps/fnd/framework/toolbox/tutorial/server/customizations/site/0/PoSummaryVO
```

You can verify the above using the **listContents** procedure of the **JDR_UTILS** package. Please refer to the section on Inspecting the MDS Repository content for an example of using this procedure to list customization documents.

Note: Each Oracle JDeveloper project has one **.jpx** file. If you have more than one Oracle JDeveloper project where you have created substitutions, you can import the substitutions specified in the respective **.jpx** file from each project, one after another. If you deploy substitutions for the same component more than once, the latest substitution will take precedence, replacing the existing substitution for that component.

Note: Since .jpx files contain BC4J extensions, you should maintain only one .jpx file per product. If you maintain different **.jpx** files for different *projects* (as mentioned in the previous note), but they all belong to the same *product,* then you should merge all the substitutions from the multiple **.jpx** files into a single product **.jpx** file and copy it to the `$APPL_TOP/.../java` staging area for that product.

For example, if you have three products, FND, PO and MFG, and each product has BC4J extensions, then you should have three separate .jpx files and each should be copied to the appropriate staging area:

- $APPL_TOP/mycompany/ fnd/12.0.0/java/FNDExtendLabSolutions.jpx
- $APPL_TOP/mycompany/ po/12.0.0/java/POExtendLabSolutions.jpx

- $APPL_TOP/mycompany/ mfg/12.0.0/java/MFGExtendLabSolutions.jpx

Step 4: Bounce the web server

Step 5: Review your deployed extensions

At this point the deployment of your extension is complete, and you should be able to login to your application to verify that your changes have successfully taken effect.

Note: If you unloaded your BC4J XML and Java to a location other than `$JAVA_TOP`, or a location not already in your classpath, you will need to modify your classpath to add that location to it and bounce the web server before logging in to view your changes.

Deploying a Custom Page

Note: The following instructions assume your custom components comply with the naming standards described in Extending OA Framework Applications.

Step 1: Compile your Java in Oracle JDeveloper and zip up your Java classes

Follow the instructions described under Step 1 of the preceding section on deploying business logic extensions. Your zip file will pick up XML files and Java classes corresponding to your extended or custom BC4J components, as well as those corresponding to your OA components. This includes OA Extension Controller Java classes that you would typically have located under the corresponding webui directory. At the end of this step your directory structure should resemble the following:

`<$JAVA_TOP>/<CompanyIdentifier>/oracle/apps/<CustomProductShortName>/server` (BC4J files)
`<$JAVA_TOP>/<CompanyIdentifier>/oracle/apps/<CustomProductShortName>/webui` (OA Extension controller files)

Then set the permissions on `$JAVA_TOP/<directory_name>` using: `chmod -R 775 <directory_name>`

Step 2: Copy your JRAD XML files files to the $APPL_TOP staging area.

Copy the JRAD XML files from the `webui` sub-package after

`<JDEV_USER_HOME>\myprojects\<CompanyIdentifier>\oracle\apps\<CustomProductShortName>\`

to

`$APPL_TOP/<CompanyIdentifier>/<CustomProductShortName>/<product-version>/mds/`

For example, you would copy the files in the package `<JDEV_USER_HOME>\myprojects\abccompany\oracle\apps\xxpo\webui` to the staging area under:

`$APPL_TOP/abccompany/xxpo/12.0.0/mds/webui`

Note: You cannot simply extract the JRAD XML files from the .zip file into the `$APPL_TOP` staging area because the directory structure in the .zip file is different from the directory structure of the staging area.

Note: There is no need to copy the compiled BC4J files from the `<$JAVA_TOP>` to the `$APPL_TOP` staging area, as the compiled files are already in the classpath (`$JAVA_TOP`).

If in the process you had to extend an Oracle business object like an Application Module, then you should also copy the **.jpx** file, which is located under `<JDEV_USER_HOME>\myprojects`, to this staging area:

`$APPL_TOP/<CompanyIdentifier>/<AppsProductShortName>/<product-version>/java/`

Step 3: Run the jpx import utility to import substitutions specified in the .jpx definition file to the MDS repository

If in the process you had to extend an Oracle business object like an Application Module, follow the instructions described under Step 3 of the preceding section to import the corresponding substitutions to the MDS repository. Otherwise, proceed to the next step.

Step 4: Import the OA component definitions into the MDS repository

Import your XML OA component definitions into the MDS repository for the database against which you are running your application. Assuming that you are following the page-building standards described in this guide, your OA XML file `<CompanyIdentifier><YourPagePG>.xml` would be located in the directory:

`<JDEV_USER_HOME>\myprojects\<CompanyIdentifier>\oracle\apps\<CustomProductShortName>\webui`

You would then copy your OA XML file to the `$APPL_TOP` staging area under:

`$APPL_TOP/<CompanyIdentifier>/<CustomProductShortName>/<product-version>/mds/webui`

You may use **import.bat** or the **import** shell script, located under the `jdev\bin` directory of your Oracle JDeveloper installation area to import your XML file or package directory into the MDS repository of the target instance.

We provide an example below to import XML files for the package specified by the directory `$APPL_TOP/<CompanyIdentifier>/<CustomProductShortName>/<product-version>/mds/webui`, using the JDK 1.3 style of the Import tool. Follow the detailed instructions provided for the Import Tool to understand the options provided and those used in the example below:

```
java oracle.jrad.tools.xml.importer.XMLImporter
$APPL_TOP/<CompanyIdentifier>/<CustomProductShortName>/<ProductVersion>/mds/webui
-jdk13
-mmddir "<JDEV_USER_HOME>\myhtml\OA_HTML\jrad"
-username <username>
-password <password>
-rootdir $APPL_TOP/<CompanyIdentifier>/<CustomProductShortName>/<ProductVersion>/mds/
-rootPackage /<CompanyIdentifier>/oracle/apps/<CustomProductShortName>
-validate
-dbconnection "(description = (address_list = (address = (community = tcp.world)
  (protocol = tcp)(host = <host.example.com>)(port = <port#>)))(connect_data =
  (sid = <SID>)))"
```

In this example, the Import tool first looks for the presence of the package `$APPL_TOP/<CompanyIdentifier>/<CustomProductShortName>/<ProductVersion>/mds/webui`. It then truncates this package path up to the **-rootDir** provided, which would leave `webui` as the remaining package path. Finally the tool prepends the **-rootPackage** path to the remaining package path to generate the final package path: `/<CompanyIdentifier>/oracle/apps/<CustomProductShortName>/webui`

Step 5: Stop and restart the web server

Step 6: Review your deployed custom OA page

Note: Ensure that the necessary AOL objects referenced from your custom page (such as Functions, Menus, and Responsibilities) have been created or exist on the Release 12 instance to which you are deploying your custom page. For a detailed description of these objects and how they are created, please refer to the *Oracle E-Business Suite Security Guide*, in the chapter called, "Oracle Application Object Library Security".

If you unloaded your BC4J XML and Java class files to a location other than `$JAVA_TOP`, or a location not already in your classpath, you will need to modify your classpath to add that location to it, and bounce the web server before logging in to view your changes.

At this point the deployment of your custom page is complete, and you should be able to login to your application to verify that your changes have successfully taken effect.

Other Resources

Use the following resources to assist with troubleshooting issues related to deploying extensions.

- Debugging OA Framework Applications
- Logging
- Troubleshooting OA Framework Applications

Extending OA Framework Applications

Overview

- Introduction
- Extensibility Standards
  - Naming Conventions
  - BC4J Objects
- Common BC4J Extensions
  - Add Additional View Object Attributes
  - Modify Entity Object Attribute Default Values
  - Modify Entity Object Validation Logic
  - Substitute Extended BC4J Objects for Parent BC4J Objects
- Controller Code Extensions
- Troubleshooting Extensions

Introduction

OA Framework provides robust support for personalizing and extending the E-Business Suite user interface and underlying business logic. The capabilities are largely achieved by leveraging the OA Framework's declarative architecture, and the object-oriented features of Java.

As a reminder, we begin with a distinction between the concepts of **"personalization"** and **"extensibility:"**

- **Personalization** refers to the ability to declaratively alter the UI to suit user or business needs.
- **Extensibility** refers to the ability to programmatically extend an application's functionality.

This document describes how to add your own custom business logic to the products that the Oracle E-Business Suite ships. To find all dependent objects of the base Oracle Application page you wish to extend, use the **About this page** link shown at the bottom-left corner of the base page. The link renders the About page which displays page definition, current session, and technology stack information for that base page. (If you do not see the **About this page** link, verify that the FND_DIAGNOSTICS profile option is enabled.)

If you need to make user interface changes, see the OA Framework Personalization Guide to see if your changes can be implemented using personalizations. If possible, user interface changes should always be made using the personalization utilities.

Note: Once you extend an OA Framework page, you can access the extended page from an external application.

Extension Standards

When creating your custom objects, please observe the following standards.

Note: *To facilitate transparent upgrades and new feature uptake, custom code must also comply with the Oracle E-Business Suite OA Framework coding standards described in Chapter 8 of the OA Framework Developer's Guide.*

Naming Conventions

Custom Application Short Name

If you want to create new pages, business logic, or whole applications and deploy your code with existing Oracle E-Business Suite applications, you must define a new application and use its short name for any related package and seed data definitions. For example, if you want to create a new, custom Procurement application, you might create a product with the short name **XXPO** (the "XX" prefix ensures that your selected product short name never conflicts with any future Oracle E-Business Suite product names).

See the *Oracle E-Business Suite Developer's Guide* for additional information on creating a custom application.

Package / Directory Structure

Any objects that you create -- either by extending (subclassing) existing Oracle E-Business Suite objects or by creating new objects from scratch -- must be added to a package that starts with your company's identifier: `<myCompanyName>.oracle.apps...`.

- If you are creating new objects from scratch, you should put them in a package that also includes your custom application short name as follows: `<myCompanyName>.oracle.apps.<customProductShortName>...`. For example, assuming your company is called **Abc Corporation** and your custom product short code is **XXPO**, any new classes and OA Components that you create should be added to the following package: `abc.oracle.apps.xxpo`....
- If you are extending existing Oracle-Business Suite objects, you may add your files to a package that corresponds directly to the original Oracle package (in this case, you don't need to add your files to a package including a custom application short code). For example, if you were to extend files that are shipped in the package shown on the left, you would add your new files to the package shown on the right.

Oracle Package	Custom Package
oracle.apps.po.server.schema	**<*myCompanyName*>**.oracle.apps.po.server.schema
oracle.apps.po.server	**<*myCompany*Name>**.oracle.apps.po.server
oracle.apps.po.webui	**<*myCompany*Name>**.oracle.apps.po.webui

File Names

As a rule, all class and OA Extension file names should follow the OA Framework File Naming Standards.

For any subclasses that you create, you should differentiate them from the superclass with your company identifier as follows: `<myCompanyIdentifier><ParentClassName>`. For example, if the parent view object (VO) is called `RequisitionsVO`, and your company's name is **Abc**, then you should name the corresponding view object subclass **`Abc`**`RequisitionsVO`.

So, if an Oracle view object class is `oracle.apps.fnd.po.RequisitionsVOImpl.java`, and your company identifier is **Abc**, then the fully qualified name of your custom class would be:

`abc``.oracle.apps.po.server.`**`Abc`**`RequisitionsVOImpl.java`.

Extending BC4J Objects

- Never work with a copy of an object; always extend an existing object or add a new object.
- Use the JDeveloper wizards to extend and create new business objects. This guarantees that the extensions are visible to other objects at the metadata layer.
- Do not make any changes to the base object as these changes do not survive an upgrade. Always extend objects and add your logic to the new classes to ensure your work survives upgrades.
- Use the substitution mechanism described below to ensure that the OA Framework uses your new classes. Do not change base object references to existing objects (the substitution mechanism handles this automatically for you).
- In general, children inherit their parents' BC4J properties, and you can override these declarative settings if needed.

Common BC4J Extensions

This section provides step-by-step instructions for completing common BC4J code extensions.

Note: *All BC4J objects (class files) are available in the Oracle E-Business Suite runtime environment or can be copied from $JAVA_TOP. There is no need to export XML pages from the MDS repository to perform BC4J extensions.*

Tip: If you are unfamiliar with JDeveloper, the OA Framework ToolBox Tutorial includes an Extending OA Framework Applications lab that provides detailed instructions for each task described below.

Add Additional View Object Attributes

In order to add additional attributes to view objects, follow these steps:

1. Analyze the page and its regions to determine which view objects to extend.

   Tip: Run the page you want to change and select the **About this Page** link found in the footer of the running page. The "About" page that renders displays the full name of the current page (for example `oracle/apps/fnd/framework/toolbox/tutorial/webui/PurchaseOrder.xml`).

   The **Page** subtab of the "**About**" page displays a Page Definition HGrid that lists the controller, application module, view object and view attribute associated with each web bean within the page hierarchy. Within the HGrid, look for the item in the region you want to change and identify the view object associated with that item. Also, expand the Business Component References Details section and make note of the application module (AM) that contains the view object for this page.

   Once your analysis is complete, you should have the name of the view object you need to extend. We will call this the *parent view object*.

2. If you have not already done so, in OA Extension, create a new empty BC4J package to hold your custom BC4J objects.

   Note: This package must adhere to the naming standard described above.

3. Create a new view object that extends the parent view object (you create the extended view object just as you would any other view object with the following difference: specify the name of the parent view object in the **Extends** field in the first page of the wizard). Be sure to follow the naming standard described above.

   Note that the parent view object package must be included in your project before you can extend the parent view object.

4. After extending the new object from the parent, you can do any of the following:

   - Add an existing Oracle entity object to your extended view object
   - Add an existing custom entity object to your extended view object
   - Add attributes from already included entity objects to your extended view object

   Note: Refer to My Oracle Support (formerly Oracle *MetaLink*) Knowledge Document 353443.1, *Oracle Application Framework: Allowing an Extended VO to See Subsequent Parent VO Changes*, to enable subsequent changes made to the parent view object to be visible to the extended view object.

5. **If the view object you extend is a non-expert mode view object:**

   - Select **Next** until you get to the **Attributes** page. Select the attribute(s) from the **Available** list that you want to include in this new

view object, to the **Selected** list. Note that the following warning message appears when you shuttle an attribute to the **Selected** list: "To add the selected attributes, the following entity usage(s) will need to be overridden from the base view object...." Select **OK**.

If the parent view object was created in expert mode (meaning it uses handwritten SQL), then the extended view object will also be created in expert mode.

- *Your new query should include the original SQL SELECT statement plus whatever changes you need to make (so you effectively override the base class query).* Specifically, as a safe and convenient starting point, you should:
  - Copy the query from the superclass **Query** page's **Query Statement** field before you create the subclass.
  - Then, create your subclass and paste the superclass query into the subclass **Query** page's **Query Statement** field.
  - Make your changes to the subclass query. Be sure to test your SQL statement before proceeding!

Note: Subclassing expert mode view objects is inherently fragile; if the base class SQL SELECT statement changes, your subclass could break after an upgrade. Always test cases like this carefully after upgrading.

6. In the Java page of the view object creation wizard, *uncheck* **View Object Class - Generate Java File** checkbox. *Check* the **View Row Class - Generate Java File** and **Generate Accessors** checkboxes.

   Note: There is no need to have a redundant *View Object Class* Java file if you don't anticipate adding any view object logic for your new attribute. By default, the BC4J framework will use the base view object's class. However, you should still generate the *View Row Class* to optimize the performance of attribute name lookup.

7. Save your view object.

Finally, you need to define a substitution so your new view object is used in place of the parent at runtime (see the substitution section below).

Modify Entity Object Attribute Default Values

In order to add or change entity object attribute default values, follow these steps:

1. Analyze the page and region that contains the item whose default value you wish to modify to identify the associated view object and entity object.

   Tip: Run the page you want to change and select the **About this Page** link found in the footer of the running page. In the **Page** subtab of the "**About**" page, look for the item you want to change within the Page Definition HGrid and identify the view object associated with that item. Select that view object's link to display the **About View Objects** page. Note the entity object (EO) that it uses for this item. We will call this the *parent EO*.

2. If you have not already done so, create a new, empty BC4J package to hold your custom BC4J objects.

   Note: This package must adhere to the naming standard described above.

3. Create a new entity object that extends the parent EO (you create the extended EO just as you would any other entity object with the following difference: specify the name of the parent EO in the **Extends Entity** field in the first page of the wizard). Be sure to follow the naming standard described above.

   Note that the parent EO package must be included in your project before you can extend the parent EO, and that all entity objects are included in a `oracle.apps.....schema.server` package.

4. For the rest of the steps, accept all the defaults and proceed until you get to **Generate Default View Object** page where you should uncheck the **Generate Default View Object** checkbox as you do not need a new view object for this extension. Then choose **Finish**.

5. Edit your new ***EOImpl** class and add a `create()` method like the one shown below (all defaulting is done in this standard method that BC4J calls when an entity object is instantiated).

```
/**
  * Initializes a new purchase order header.
  */
  public void create(AttributeList attributeList)
   {
      super.create(attributeList);

     // Add or change the default values here.
     // EXAMPLE: setPaymentTermsCode("NET_60");
   }
```

   Note that you must also include the following import statement to make this example compile successfully:

```
import oracle.jbo.AttributeList;
```

Finally, you need to define a substitution so your new entity object is used in place of the parent at runtime (see the substitution section below).

Modify Entity Object Validation Logic

In order to extend (add or modify) BC4J validation logic, follow the steps listed above in the Modify Entity Object Attribute Default Values section. Remember to define a substitution for your new entity object so that it is used in place of the parent at runtime (see the substitution section below).

Attribute-Level Validation

For validation to be performed whenever a single attribute value is set, you need to override the corresponding attribute's setter as illustrated in the `setUnitPrice()` example below. Note that we call `super.setUnitPrice()` AFTER performing our custom validation to avoid setting a bad value on the entity object.

```
public void setUnitPrice(Number value)
{
   // Add validation before calling the super class method call.
   // The super class method will perform some validation and set the attribute
   // and hence, additional validation needs to be added before the super class
   // method call.

   if (value != null)
   {
     // Verify value is <= 10000.
     // Proper message with the message name should be seeded.

     If (value.compareTo(10000) > 0)
     {
       throw new OAAttrValException(OAException.TYP_ENTITY_OBJECT,
                                    getEntityDef().getFullName(), // EO name
                                    getPrimaryKey(), // EO PK
                                   "UnitPrice", // Attribute Name
                                    value, // Attribute value
                                   "AK", // Message product short name
                                   "FWK_TBX_T_PO_PRICE_EXCEEDED"); // Message name

     }
    }


   super.setUnitPrice(value);
 }
```

Note that you must also include the following import statements to make this example compile successfully:

```
import oracle.jbo.domain.Number;
import oracle.apps.fnd.framework.OAAttrValException;
import oracle.apps.fnd.framework.OAException;
```

Entity-Level Validation

If you want to perform cross-attribute validation or any other entity-level validation, you need to override the `validateEntity()` method. For entity-level exceptions, you should throw an `oracle.apps.fnd.framework.OARowValException` instead of the `oracle.apps.fnd.framework.OAAttrValException` shown in the attribute-level example above.

Translated Messages for Exceptions

As mentioned in the example code comment above, all exceptions should throw a translatable message as defined in the Oracle E-Business Suite Message Dictionary. At runtime, OA Framework will display the message text in the appropriate runtime language (assuming a translation is available).

Substitute Extended BC4J Objects for Parent BC4J Objects

After you finish extending the business objects, you need to configure the BC4J factory methods to instantiate your objects in place of the original objects.

The existing application code uses the factory method APIs provided by BC4J to construct business object instances. The application code passes in the fully qualified XML file name to the factory method, and then the BC4J framework returns the appropriate Java object at runtime. With the factory method, the BC4J framework can flexibly change the type of the object that gets returned to some other subclass object types. To accomplish this, a set of rules must be specified so the BC4J framework can perform this exchange.

Follow these steps to define a single substitution so BC4J *can substitute the extended business object for the base object throughout the application.*

1. After creating and saving your new view object following the instructions above, select the new view object in the **Applications Navigator** panel and the select the **Project Properties** icon.
2. In the **Project Properties** navigation tree, select **Business Components**, then **Substitutions**.
3. In the **Available** list, expand the package that contains the base view object that you want to replace and select that base view object (for example, **POSummaryVO**). In the **Substitute** list, expand the package that contains the new extended view object and select that extended view object (for example, **MyCompanyPoSummaryVO**). Select **Add**, then choose **OK** to create a new substitution rule.
4. Save your project.

When you save your changes, the BC4J framework updates the project's `.jpx` file (each project `.jpr` has a corresponding `.jpx` file of the same name). The `.jpx` is an XML document that includes a "Substitutes" section as shown in the following example from the OA Framework ToolBox Tutorial `ExtendLabSolutions.jpx`:

```
<Substitutes>
  <Substitute OldName ="oracle.apps.fnd.framework.toolbox.tutorial.server.PoSummaryVO"
       NewName ="mycompany.oracle.apps.fnd.framework.toolbox.tutorial.server.MyCompanyPoSummaryVO"/>
</Substitutes>
```

Deploying Substitutions

Normally, the BC4J runtime does not read a project's `.jpx` file. To override this default behavior in JDeveloper so your substitutions can be used, you must do the following to ensure that the classpath contains the `.jpx` file.

1. In the **Project Properties** navigation tree, select **Run/Debug**. Select **Edit** for the **Default Run Configuration**. In the **Launch Settings** page of the **Edit Run Configuration** window, *add* the following option to the existing **Java Options**:

   `-Djbo.project=<name>`

   Note that the `<name>` value is the `.jpx` file name *without* the `.jpx` extension (for example, **ExtendLabSolutions**). Be sure to include a space between any existing options and the new option. For example, `-Djbo.project=ExtendLabSolutions.`

   Warning: If you forget to add this Java option, you might encounter a runtime exception like **oracle.apps.fnd.framework.OAException: oracle.jbo.NoDefException: JBO-25002: Definition SiteName of type Attribute not found**. Without this setting, BC4J does not read the substitution rules, and therefore looks to the base objects without consideration for your new objects.

2. When you run the page that references the view object, in the **About this Page** link for the page, you can confirm your substitution by navigating to **Personalization** sub tab, where the substitution should be listed in the **Business Component Substitutions** section.

3. Using the Admin Personalization user interface, you can update your page UI to reference your new substitutions. For example, if you added a new attribute to your new extended view object, you can add a new column to a table that references the original view object, by simply specifying the old view instance name (which has now been substituted with the new extended view object) and the newly added attribute name.

4. To deploy substitutions for use outside JDeveloper, see Deploying Customer Extensions. In general, the project is imported to a designated environment using the **jpximport** utility.

Known Issue

OA Framework does not support substituting a root application module due to a known limitation in MDS and personalizations.

A root application module substitution has no runtime effect in a deployed environment outside JDeveloper or could lead to a runtime error with a "Cannot Display Page" message in a LOV modal window in JDeveloper (bug 4410729).

Controller Code Extensions

As described above, if you need to make UI changes, you should make them using the personalization utilities. In release 12, controller code extensions should be avoided.

- The `oracle.apps.fnd.framework.webui.OAControllerImpl` class methods, while marked as public, should be treated as if they are private (they should not be considered part of a supported public interface). These methods are likely to change.
- There is no guarantee that controller extensions will survive an upgrade (you should assume that they will not).

Troubleshooting Extensions

Before you even deploy your extension, if it does not render properly within JDeveloper, check the following:

- **Personalizations:**
  - If your extension is not appearing, but the page renders fine, check the **About this Page** link. If the correct substitution appears in the **Personalization** tab of the **About Page**, it is likely that a personalization is masking your extension. Deactivate all personalizations for the page.
  - If the page itself cannot render, you can launch the Personalization UI for that page using the Functional Administrator responsibility and navigate to the Manage Personalization Levels page to deactivate its personalizations.
- **Extended View Object Structure:**
  - If the parent view object was created in expert mode, make sure you copy the entire query from the parent view object to the extended view object before you add your new attributes, if any, to the end of the copied SELECT clause.
  - Check the attribute mappings for both the parent and extended view object to verify that there are no conflicts.

If you encounter problems with your extension *after* deployment, use the following steps to help troubleshoot the problem:

1. Make sure you deactivate all personalizations on the deployed environment and if the extended view object is an expert mode view object, verify that the query structure is correct.
2. If the extension still does not render properly, go back and run the page in JDeveloper.
3. If the extension renders fine, then in JDeveloper, edit the project that contains the extension. Remove the substitution and run the page to verify that the page itself also renders fine in JDeveloper.
4. Deploy the BC4J extension to your database using **jpximport.bat**. This utility transforms each substitution from your project's .jpx file into a separate site level customization document which it then imports into the database MDS repository.
5. Bounce your web server, then login to your application and run the page.
6. If the extension still does not render in the deployed environment, use the following APIs from the JDR_UTILS package to help you troubleshoot and isolate the problem:
    - `JDR_UTILS.listcustomizations(<regionName>);` - list all the customization documents for a region
    - `JDR_UTILS.printdocument(<docName>);` - print out the substitution document content.
    - `JDR_UTILS.deletedocument(<docName>);` - remove the substitution

Accessing an OA Framework Page From an External Application

Overview

To allow an external application to integrate with or access an OA Framework application, you need to construct a link to an OA Framework page using an entry point JSP page. When a user selects the link from the external application, the user gets redirected to the Oracle E-Business Suite login page for authentication. After successful authentication, the OA Framework page renders.

Your link should be based on one of three possible URL constructs:

- Run Function URL (based on `RF.jsp`)
- OA Bookmarkable URL (based on `OA.jsp`)
- OA Drill Down URL (based on `OAD.jsp`)

Run Function URL (based on RF.jsp)

A Run Function URL is by default a bookmarkable URL that acts as the entry point to any OA Framework application. It must be of the following form:

```
RF.jsp?function_id=<func_name>&resp_id=<resp_name>&resp_appl_id=<appl_name>&security_group_id=0
   &[param1=value1&param2=value2&...]
```

where

- `function_id` is the ID/name of the function to run, as defined in the Oracle E-Business Suite Functions form. The Web HTML call of this function encapsulates the OA Framework page.
- `resp_id` and `resp_appl_id` represent the ID/name of the Responsibility and Application, respectively, for the security context used to run the function.
- `security_group_id` is the ID of the security group for the security context used to run the function.

For example:

```
http://<SERVER>:<PORT>/OA_HTML/RF.jsp?function_id=UMX_SEARCH_PERSON&resp_id=UMX&
   resp_appl_id=FND&security_group_id=0&lang_code=US
```

Note: If the URL is to be accessed from Microsoft Excel or Microsoft Word, then replace `RF.jsp` with `OAD.jsp` to create an OA Drill Down URL.

OA Bookmarkable URL (based on OA.jsp)

An OA Bookmarkable URL is an OA Framework page that is implemented as bookmarkable according to the guidelines defined in the Accelerated Validation topic. An external application can use an OA Bookmarkable URL as an entry point for OA Framework integration or access.

Note: If the URL is to be accessed from Microsoft Excel or Microsoft Word, then replace `OA.jsp` with `OAD.jsp` to create an OA Drill Down URL.

OA Drill Down URL (based on OAD.jsp)

In Release 12.2, `OAD.jsp` is introduced as the entry point JSP page to drill down to from a Microsoft Excel or Microsoft Word document. Simply replace `RF.jsp` or `OA.jsp` with `OAD.jsp` in your existing Run Function or OA Bookmarkable URL if the external application you are integrating with is Microsoft Excel or Microsoft Word.

As a rule, use `OAD.jsp` only when you drill down to an OA Framework page from Microsoft Excel or Microsoft Word. Use `RF.jsp` or `OA.jsp` for all other cases of external applications.

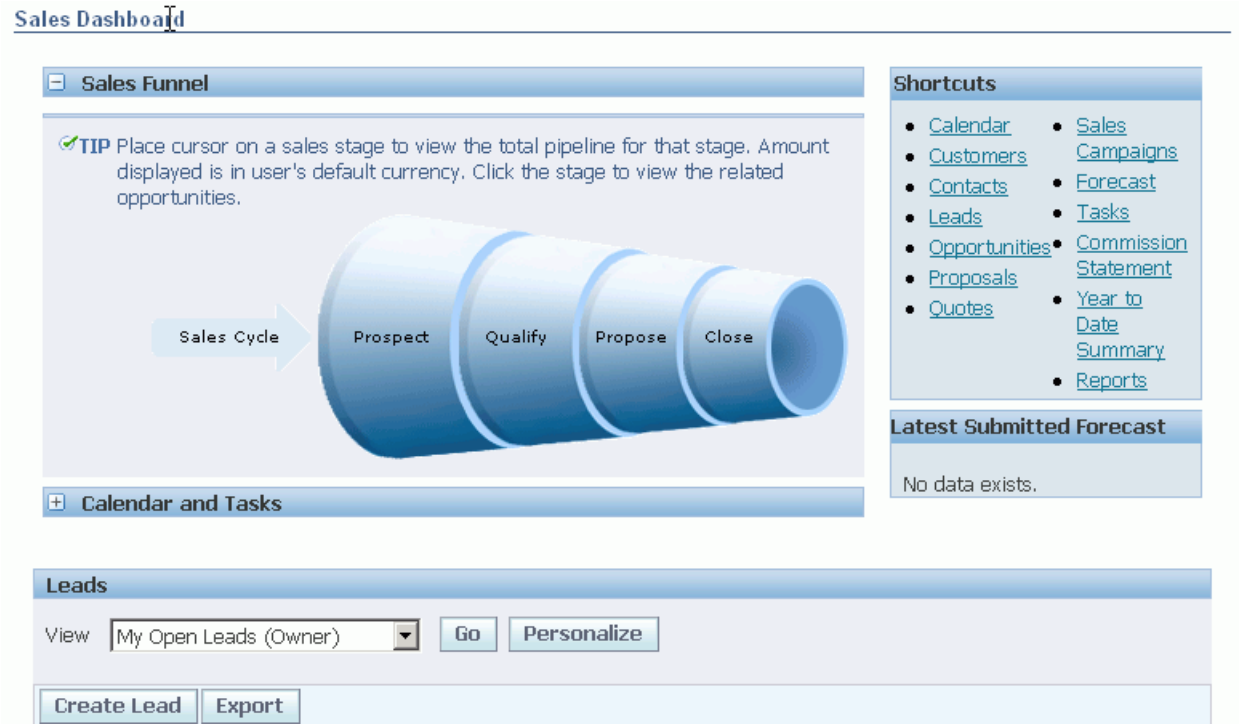
Additional Information: Refer to My Oracle Support Knowledge Document 1077728.1, Microsoft Office Integration with Oracle E-Business Suite 11i and R12.

Creating a Configurable Page

Overview

A Configurable Page is a page designed with built-in layout personalization and content selections capabilities. It is composed of personalizable layout components called flexible layout regions and self-contained content components called flexible content regions.

As of Release 12.1 RUP 2, a page's flexible layout or flexible content region may be defined to display a Hide/Show header that allows users to expand or collapse that region on the page. The figure below shows the Sales Dashboard configurable page, with the "Sales Funnel" flexible content region expanded and the "Calendar and Task" flexible content region collapsed.



An administrator can also further personalize the page by rearranging the layout and visibility of information to suit specific user audiences. OA Personalization Framework provides a user interface, called the Page Layout Personalization screen, specifically designed for personalizing configurable pages.

Contents

- Flexible Layout and Flexible Content
- Declarative Implementation
  - General Instructions
  - User-Personalizable Regions in flexibleContent Regions
  - Creating Configurable Page Templates
  - Flexible Layout Examples
- Runtime Control
- Personalization Considerations
- Known Issues
- Related Information

Flexible Layout and Flexible Content

A configurable page contains at least one flexible layout.

Note In Oracle E-Business Suite, a configurable page is composed of a hierarchy of flexible layout and flexible content regions, starting with a flexible layout region right below the page layout region. Other region styles are only allowed as leafs within the flexible layout hierarchy or within a flexible content region. Please refer to the coding standards for more detail.

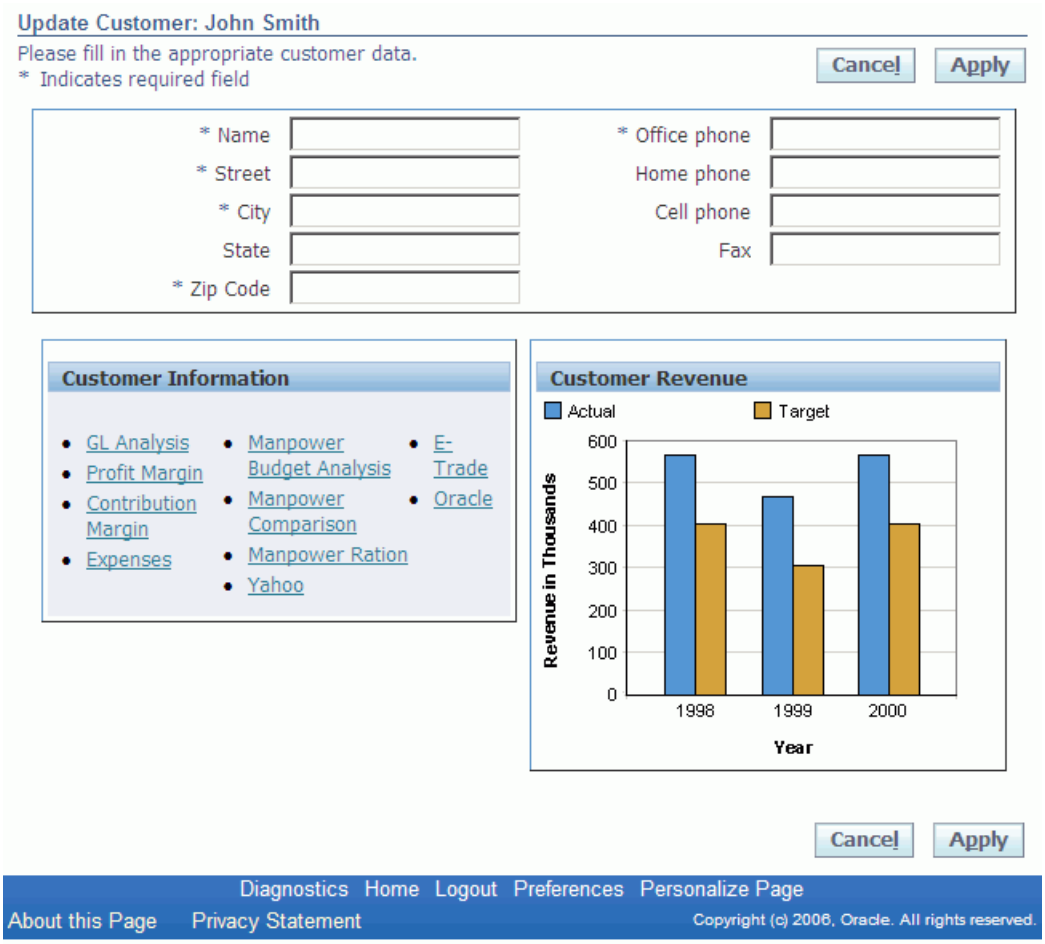
The layout is "flexible" because it can be personalized by a user or administrator to take on a different shape.

A configurable page also consists of pieces of content, also referred to as resources, that can be rearranged, hidden or shown. These resources are displayed within the cells of a layout region. When you create a configurable page in OA Extension, the resources are

known as flexibleContent regions, and the layout regions are known as flexibleLayout regions.

Specifically, a flexibleContent region defines the content itself - how the content appears and what personalizations a user or administrator can make to the content. A flexibleLayout region defines the area in which the content is placed when the page is run. When you define a flexibleContent region, you have the option of assigning it to a specific flexibleLayout region, so that the content is displayed in that flexibleLayout when it is rendered. If you do not assign it to a flexibleLayout region, the content becomes a shared resource that the user can later choose to add to any flexibleLayout region from a resource catalog displayed in the OA Personalization Framework Add Content page.

The figure below is an example of a configurable page. The blue/black bordered areas represent the flexibleLayout regions in the page, whereas the content within the borders represent the flexibleContent regions.



The number of cells within a flexibleLayout region is determined by the value of its 'rows' and 'columns' attributes. These attributes determine how many flexibleLayout or flexibleContent children the current flexibleLayout region has. For example, if a flexibleLayout region has one column and one row, it has one cell as illustrated in the figure below, on the left. If a flexibleLayout region has 1 row and 2 columns, it displays 2 cells side by side. Each cell (or flexibleLayout region) can also have multiple rows and columns and therefore be split into additional cells. In the figure on the right, below, the flexibleLayout starts with 1 row and 2 columns, but the cell on the far right is further divided into 1 column and 2 rows, resulting in the three cells shown.



Note In order to be able to take full advantage of page layout personalization capabilities, it is important when you create your configurable page, to use flexibleLayout and flexibleContent regions at all levels when you create the page hierarchy structure under your page layout region. In Oracle E-Business Suite, this is required, as it is the coding standard.

Note: If the content shown within a flexibleLayout region exceeds the pixel height or pixel width specified for the flexibleLayout region, then OA Framework displays a horizontal or vertical scroll bar in the flexibleLayout region, respectively. If the content in the flexibleLayout region exceeds both the specified pixel height and pixel width, then OA Framework displays both a horizontal and a vertical scroll bar in the flexibleLayout region. The availability of scroll bars allows dashboard-type pages to utilize the screen area better and makes it possible to embed more portlet-type content.

Declarative Implementation

General Instructions

The following steps describe generally, how to implement a configurable page in OA Extension. Note that the steps may vary depending on the type of page layout that you want to achieve. Refer to the Flexible Layout examples for suggestions on some basic layouts you can implement in a configurable page.

Step 1: In your project, create a new page and in that page, create a new region of style **pageLayout**. Set the necessary properties on the pageLayout region.

Step 2: Select the pageLayout region and choose **New** > **Region** from the context menu. Set the `Region Style` property for this new region to **flexibleLayout**, to create a flexibleLayout region that you can use to layout flexibleContent or additional flexibleLayout regions.

Step 3: Using the Property Inspector, you can set the following properties on the **flexibleLayout** region (* indicates a required property):

- `ID`* - Specify an identifier that uniquely identifies the region in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Attribute Set` - specify an attribute set if you want to maintain a standard look and feel for your region.
- `Title` - specify the title for the region. The title is displayed only when the region is rendered in the collapsed state.
- `Disclosed` - specify **True** or **False** to indicate if the region is to be initially expanded (disclosed) or collapsed, respectively. The default is True.
- `Rendered` - specify **True** or **False** to indicate if the region is to be rendered. Setting `Rendered` to False is equivalent to hiding the region in the configurable page (or selecting Remove Content in Page Layout Personalization screen). The default is True.
- `Layout Style` - specify **vertical** or **horizontal** to indicate the direction that the content should be laid out. The default is vertical. This property takes effect only for those flexibleLayout regions that are leaf nodes, where the `Rows` and `Columns` properties are "1" and flexibleContents are added.
- `Columns` - specify the number of columns in the flexibleLayout region. The default is 1.

- `Rows` - specify the number of rows in the flexibleLayout region. The default is 1.

  Note The number of columns and rows indicate the number of cells within the flexibleLayout region. OA Extension automatically creates a child flexibleLayout region for each cell. For example, if `Columns`=2 and `Rows`=3, the flexibleLayout would contain 6 cells total, 2 columns going across and 3 rows going down, and OA Extension would create 6 flexibleLayout children for the flexibleLayout region.

- `Height` - specify the display height of the flexibleLayout in pixels or as a percentage (by including the % sign). If you specify the height in pixels and the flexibleLayout content's height exceeds the specified pixel height, a vertical scrollbar will render.
- `Width` - specify the display width of the flexibleLayout in pixels or as a percentage (by including the % sign). If you specify the width in pixels and the flexibleLayout content's width exceeds the specified pixel width, a horizontal scrollbar will render.
- `Show Border` -specify **True** or **False** to indicate whether or not to display a border around the flexibleLayout region. The default is False.
- `Show Header` - specify **True** or **False** to indicate whether or not to display a header for the flexibleLayout region. The default is True. As of Release 12.1 RUP 2, if this property is set to **True**, and the properties `User Personalization` is set to **True** and `User Operations` is set to **disclose**, a Hide/Show header renders above the region allowing users to hide or show the region. Prior to Release 12.1 RUP 2, instead of a Hide/Show header, **Expand** (plus sign) and **Collapse** (minus sign) icons would render near the top left corner of the region to allow users to expand or collapse the region. If this property is set to **False**, then no Hide/Show header or Expand/Collapse icon renders regardless of the values set for the `User Personalization` and `User Operations` properties.
- `Admin Personalization` - specify **True** or **False** to indicate if administrators are allowed to perform the personalization operations listed under the `Admin Operations` property. The default is True.
- `User Personalization` - specify **True** or **False** to indicate if users are allowed to perform the personalization operations listed under the `User Operations` property. The default is True.
- `User Operations` - specify the personalization operations that a user can perform on the component, if `User Personalization` is set to True. The choices are **(Default)** or **disclose**. The default value is (Default), which implies "null". As of Release 12.1 RUP 2, when you set this property to **disclose** and set `Show Header` and `User Personalization` both to **True**, the user can either hide or show the region in the page using the rendered Hide/Show header.
- `Admin Operations` - specify the personalization operations that an administrator can perform on the component if `Admin Personalization` is set to True. The choices are "null" or **add**. The default is add. If you set this property to add and set `Admin Personalization` to True, an Administrator can add predefined flexibleContent to the flexibleLayout region when personalizing the page in the Page Layout Personalization screen or the Page Hierarchy Personalization screen.

Note The order in which the flexibleLayout regions within the Structure pane appear is the order in which they display in the configurable page.

Step 4: To create flexibleContent, select the **pageLayout** region and choose **New** > **flexibleContents** from the context menu. OA Extension automatically creates a flexibleContentList named child in the pageLayout Components folder. There are no properties to set on the flexibleContentList.

Step 5: Select the **flexibleContentList** named child and chose **New** > **flexibleContent** from the context menu.

Step 6: Set the following properties on the **flexibleContent region**. In particular, specify the `Flexible Layout Reference` property to the **ID** of the flexibleLayout region in which you want this flexibleContent region to reside. (* indicates a required property)

- `ID`***** - Specify an identifier that uniquely identifies the region in the page. Refer to the OA Framework File / Package/ Directory Standards for guidelines on defining an ID.
- `Attribute Set` - specify an attribute set if you want to maintain a standard look and feel for your region.
- `Flexible Layout Reference` - specify the **ID** of the flexibleLayout region in which you want this flexibleContent region to reside. You may use the list of values to select a flexibleLayout region ID. The list includes all leaf flexibleLayout regions (that have no children) defined in the current page structure. Leaving this property null indicates that the flexibleContent does not have to be associated with any particular flexibleLayout region and can later be added to any flexibleLayout region using the Add Content control on the Page Layout Personalization screen or the Page Hierarchy Personalization screen.
- `Title` - specify the title for the region. The title is displayed only when the region is rendered in the collapsed state.
- `Description` - specify a brief description of the items that are contained in this element. Use short phrases as this description is used to list the predefined content that can be manipulated from the Page Layout Personalization screen or the Page Hierarchy Personalization screen.
- `Disclosed` - specify **True** or **False** to indicate if the region is to be initially expanded (disclosed) or collapsed, respectively. The default is True.
- `Rendered` - specify **True** or **False** to indicate if the region is to be rendered. Setting `Rendered` to False is equivalent to hiding the region in the configurable page (or selecting Remove Content in Page Layout Personalization screen). The default is True.
- `Show Border` -specify **True** or **False** to indicate whether or not to display a border around the flexibleContent region. The default is False.
- `Show Header` - specify **True** or **False** to indicate whether or not to display a header for the flexibleContent region. The default is True. As of Release 12.1 RUP 2, if this property is set to **True**, and the properties `User Personalization` is set to **True** and `User Operations` is set to **disclose**, a Hide/Show header renders above the region allowing users to hide or show the region. Prior to Release 12.1 RUP 2, instead of a Hide/Show header, **Expand** (plus sign) and **Collapse** (minus sign) icons would render near the top left corner of the region to allow users to expand or collapse the region. If this property is set to

False, then no Hide/Show header or Expand/Collapse icon renders regardless of the values set for the `User Personalization` and `User Operations` properties.

- `Admin Personalization` - specify **True** or **False** to indicate if Admin level users are allowed to perform the personalization operations listed under the `Admin Operations` property. The default is True.
- `User Personalization` - specify **True** or **False** to indicate if users are allowed to perform the personalization operations listed under the `User Operations` property. The default is True.
- `User Operations` - specify the personalization operations that a user can perform on the component, if `User Personalization` is set to True. The choices are **(Default)** or **disclose**. The default value is (Default), which implies "null". As of Release 12.1 RUP 2, when you set this property to **disclose** and set `Show Header` and `User Personalization` both to **True**, the user can either hide or show the region in the page using the rendered Hide/Show header.
- `Admin Operations` - specify the personalization operations that an administrator can perform on the component if `Admin Personalization` is set to True. The choices are "null" or remove. The default is remove. If you set this property to remove and set `Admin Personalization` to True, an Administrator can remove flexibleContent so that it does not render on the page, when personalizing the page in the Page Layout Personalization screen or the Page Hierarchy Personalization screen. Removed regions can be added back using the Add Content control.
- `Resource Description` - specify a user friendly, concise description of the items contained in this flexibleContent region and optionally, its usage. In the Page Layout Personalization screen, when you select the Add Content control to navigate to the Add Content page, a catalog of predefined flexibleContent regions for this page appears. The Content Name, as well as Content Type is listed for each flexibleContent region. The Content Name is the value specified in the Title property of this region and the Content Type is the value specified in the Resource Description property of this region.

Step 7: Create the actual content for the flexibleContent region. Select the **flexibleContent region** in the Structure pane and select **New** > **Region** or **Item** from the context menu. Set the style for the region or item as desired and set the properties that are appropriate for that style.

Note To specify a predefined region as the content of a flexibleContent region, create a new child region under the flexibleContent region and specify the ID of the predefined region in the **Extends** property of the newly created child region.

Note The order in which the flexibleContent regions appear within the Structure pane is the order in which they display in the configurable page.

Step 8: Depending on your layout needs, you can also embed a flexibleLayout region under a flexibleContent region. See the Rack Layout example.

Select the **flexibleContent** region and choose **New** > **Region** from the context menu. Set the `Region Style` to **flexibleLayout**, and set the remaining properties on this nested flexibleLayout region as appropriate.

Step 9: Create other flexibleContent regions, as described above, to display in the embedded flexibleLayout region. For these flexibleContent regions you create, be sure to set the `Flexible Layout Reference` to the **ID** of the nested flexibleLayout region you created in Step 8 and set the remaining properties on the flexibleContent region as appropriate.

User-Personalizable Regions in flexibleContent Regions

Recall that a query region that contains a results table can be defined as user-personalizable. A user can create multiple "views" of the query region. The views define saved, named queries that contain instructions on how to display and sort the columns in the results table. When you define a flexibleContent region, you can add as its content, a user-personalizable region. Refer to the Chapter 4 topic, Creating End User Personalizable Pages for information on how to define a user-personalizable region.

As mentioned in the Declarative Implementation of a configurable page, when you define a flexibleContent region, you can specify a value for the `Flexible Layout Reference` property, so that the flexibleContent region is rendered in that flexibleLayout region. If you do not specify a value for the property, the flexibleContent region becomes a shared resource that a user can later add to any flexibleLayout region from a resource catalog displayed in the OA Personalization Framework Add Content page.

If you define a flexibleContent that contains a user-personalizable region, a user may create multiple personalized views of that region. When a user selects Add Content in the Page Layout Personalization screen to add content to a flexibleLayout region, you can allow the user to not only add a specific flexibleContent to a flexibleLayout region, but also add flexibleContent with a **specific personalized view** applied to the user-personalizable region within that flexibleContent.

Note: A flexibleContent region can contain more than one region and the content of a region can include other nested regions. However, due to the characteristics of the resource catalog, a user can only add to a flexibleLayout region, a flexibleContent region with one personalized view applied to the flexibleContent at any given time.

The following optional steps describe how to enable users to select personalized views of a flexibleContent region in the resource catalog:

Step 1: Follow the General Instructions described above to create a configurable page with one or more flexibleContent regions.

Step 2: In the Structure pane of OA Extension, select a flexibleContent region and choose **New** > **Region** from the context menu. Set the `Region Style` property of this new region to **flowLayout**. Create a user-personalizable query region as a child of this flowLayout region.

Note: Typical usage extends this flowLayout from a flowLayout defined elsewhere. Be sure to set the `Extends` property on this flowLayout region to the full region reference of the flowLayout region you wish to extend so that the entire hierarchy of the flowLayout's children (query, query/stackLayout, query/stackLayout/table) appear in the current flowLayout region.

Step 3: Select the flexibleContent region again in the Structure pane. In addition to setting the properties described in Step 6 of the General Instructions, set the following two properties:

- `View Source Region ID` - set to the **ID** of the region for which personalized views may be defined. If the user-personalizable region is defined outside the configurable page (extends), use the ... button to display the package browser window so you can select the **complete region reference**.
- `View Source Target ID` - set to the **ID** of the query region for which to apply a personalized view. If the user-personalizable region is defined outside the configurable page (extends), use the ... button to display the package browser window so you can select the **complete region reference**.

Note: A user-personalizable region always has a table region embedded in a query region, so in typical usage, the `View Source Region ID` is always set to the **ID of the table region** and the `View Target Region ID` is always set to the **ID of the query region**.

Creating Configurable Page Templates

If you plan to create multiple configurable pages that share some of the same flexible layout or flexible content, you can save yourself time and maintain consistency by using templates to create your new pages. Following are instructions describing how to create a template:

Step 1: In your project, create a new page and in that page, create a new region of style **pageLayout**. Set the necessary properties on the pageLayout region.

Step 2: Select the pageLayout region and choose **New** > **Region** from the context menu. Set the `Region Style` property for this new region to **flexibleLayout** to create a flexibleLayout region. Refer to the General Instructions or Flexible Layout Examples for further instructions on how to define the flexibleLayout and flexibleContent regions that you want to create as templates

Step 3: Set the `Scope` property for the pageLayout region to indicate the base packages that are allowed to reuse the page and its components.

Step 4: Save the page.

Step 5: To reuse the contents of the page you just saved in Step 4 as a template for other pages, create another new page.

Step 6: Create a pageLayout region and under that pageLayout region, create a flexibleLayout region. Set the flexibleLayout region's `Extends` property to the fully qualified name of the template page flexibleLayout region you wish to share. Similarly, if you wish to use a flexibleContent region from the template page in this new page, create a new flexibleContent region and set its Extends property to the fully qualified name of the template page flexibleContent region you want.

Note You cannot edit the children of the template region/item you are extending from your current page, so be careful how you define a region/item as a template. You can only edit the children in their original source page, and any edits you make propagate to all pages that extend the region or item.

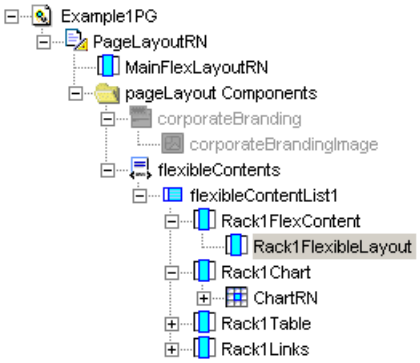
Flexible Layout Examples

The following examples illustrate some typical page layouts:

- Example 1 - Creating a Rack Page
- Example 2 - Creating a Table with One Row and Two Columns
- Example 3 - Creating an Asymmetric Layout
- Example 4 - Creating a Projects Home Page

Example 1 - Creating a Rack Page

A Rack Page is a simple one box layout whose content is laid out in a vertical direction. The Rack page should be constructed as follows:



Step 1: In the Structure pane, select the pageLayout region and select **New** > **Region** from the context menu.

Step 2: In the Property Inspector, set the `Region Style` of the new region to **flexibleLayout.**

Step 3: Ensure that the properties of the flexibleLayout region are as follows: `Rows`=**1,** `Columns`=**1**, `Layout Style`=**vertical,** and `ID`=**MainFlexLayoutRN.**

Step 4: Create one flexibleContent region for each rack required, and set the `flexibleLayoutRef` property to **MainFlexLayoutRN** and `ID` property to **Rack<N>FlexContent** where N is a number that identifies this rack uniquely.

Step 5: For each flexibleContent region (**Rack<N>FlexContent**) that represents a rack, create a child flexibleLayout region. Select the **flexibleContent** region and choose **New** > **Region** from the context menu. Set the following properties on the child region: `ID`=**Rack<N>FlexLayout,** `Region Style`=**flexibleLayout,** `Rows`=**1**, `Columns`=**1**, and `Layout Style`=**horizontal**.

Step 6: For each item (content) that you want to display in a rack (usually a table, a chart, and links), create a new flexibleContent region by selecting the **flexibleContentList** and choosing **New** > **flexibleContent** from the context menu. Set the `ID` property appropriately to describe the content and set the `Flexible Layout Reference` property to **Rack<N>FlexLayout**.

Step 7: For each flexibleContent region, select that **flexibleContent** region and choose **New** > **Region** or **Item** to create the actual content (chart, table, links).

Example 2 - Creating a Table with One Row and Two Columns

A flexibleLayout region with one row and two columns creates a 'table' with one row and two columns.

To control the characteristics of each of the cells in this table, one nested flexibleLayout is needed for each cell.

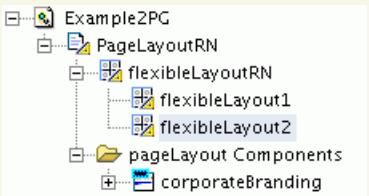
To create this layout :

Step 1: In the Structure pane, select the pageLayout region and select **New** > **Region** from the context menu.

Step 2: In the Property Inspector, set the `Region Style` of the new region to **flexibleLayout.**

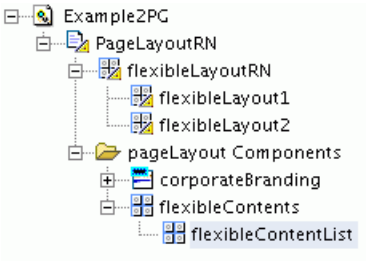
Step 3: Set the properties of the flexibleLayout region as follows: set `Rows` to **1**, and `Columns` to **2**. Since there are two columns, two nested flexibleLayouts are needed. OA Extension automatically creates these for you when you set the `Rows` and `Columns` properties.

Step 4: Set the `width` property for the newly created flexibleLayout1 and flexibleLayout2 regions to achieve a look similar to the figure on the left, otherwise both columns will be of equal width.



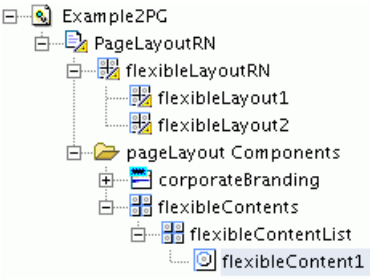
Step 4: In the Structure pane, select the pageLayout region and select **New** > **flexibleContents** from the context menu.

Step 5: This creates a new **pageLayout Component** called **flexibleContents** with one child of **flexibleContentList**. The **flexibleContentList** is where you define the content:



Step 6: Select **flexibleContentList1** and choose **New** > **flexibleContent** from the context menu.
This creates a region with `Region Style` set to **flexibleContent.**

Step 7: In the Property Inspector, set the `ID` property of the new region to **flexibleContent1** and set the `flexibleLayoutRef` property to **flexibleLayout1**. This indicates that the content should appear in the flexibleLayout with the `ID` of **flexibleLayout1**.

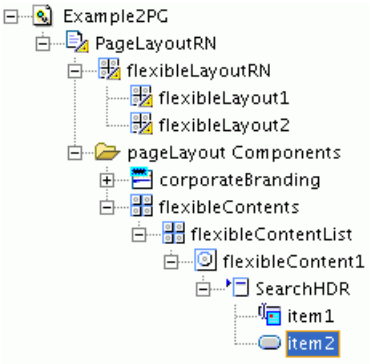


Step 8: Add content to the region called **flexibleContent1**. You can add regions or items. As an example, the following steps add a Header region with a messageTextInput item inside.

Step 9: Select **flexibleContent1** and choose **New** > **Region** from the context menu. Set `Region Style` property to **header**, `ID` property to **SearchHDR** and `Text` property to **Search**.

Step 10: Select **SearchHDR** and choose **New** > **Item** from the context menu. Set the `Item Style` property to **messageTextInput** and set the `Prompt` property to **Search**.

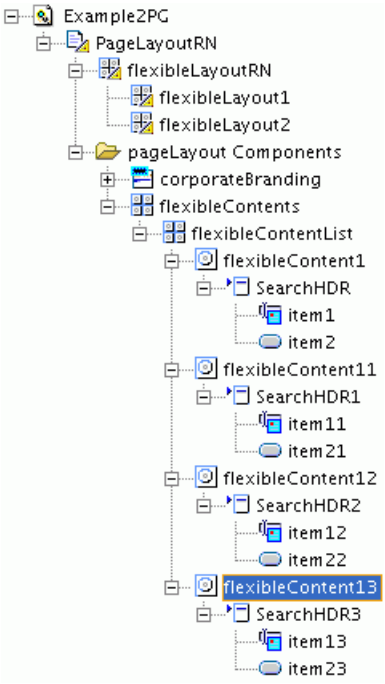
Step 11: Select **SearchHDR** again and select **New** > **Item** from the context menu. Set the `Item Style` property to **button** and set the `Prompt` property to **Go**.



Step 12: Create additional content. In the Structure pane, select **flexibleContent1** and choose **Copy** from the context menu.

Step 13: Select **flexibleContentList1** and choose **Paste** from the context menu.

Step 14: Repeat Step 13 two more times to create content as follows:



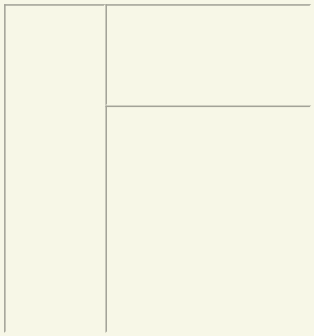
Step 15: Select **flexibleContent11** and in the Property Inspector, set the `flexibleLayoutRef` property to **flexibleLayout2**.

Step 16: Select **flexibleContent12** and in the Property Inspector, set the `flexibleLayoutRef` property to **flexibleLayout2**.

Step 17: In the Structure pane, select the flexibleLayout called **flexibleLayout** and set the `Show Border` property to **True**. This helps demonstrate the example by drawing a border around the content.

Step 18: Run the page.

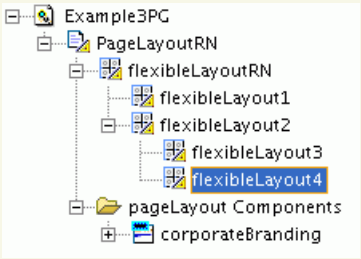
Example 3 - Creating an Asymmetric Layout



A layout like this is obtained by subdividing the cells of the flexibleLayout region. This example is in fact like example 2 , with the right hand column subdivided into two rows.

To create this layout:

1. In the Structure Pane, select the pageLayout region and select **New** > **Region** from the context menu.
2. In the Property Inspector, set the `Region Style` of the new region to **flexibleLayout.**
3. Set the properties of the flexibleLayout region as follows: set `Rows` to **1**, and `Columns` to **2**.
4. OA Extension automatically creates two nested flexibleLayout regions called **flexibleLayout1** and **flexibleLayout2.**
5. Set the `Rows` properties to **2** for **flexibleLayout2**.
6. OA Extension automatically creates two nested flexibleLayout regions called **flexibleLayout3** and **flexibleLayout4**. Refer to the General Instructions for information about other properties you can set on this region.



Example 4 - Creating a Projects Home Page

A Projects Home page has two columns whose content is laid out in a vertical direction. It is similar to the layout in Example 2 above.

Runtime Control

There is no runtime control code necessary to create a configurable page.

Personalization Considerations

For information on how to personalize a configurable page, refer to Chapter 2: Admin-Level Personalizations, in the OA Framework Personalization Guide.

Note: When an end-user changes the state of a flexibleLayout or flexibleContent region (that is, expands (discloses) or collapses (undiscloses) it), OA Framework automatically creates a user-level personalization for that flexibleLayout or flexibleContent region's most recent state.

You should also be aware of the following when you personalize a configurable page:

- To personalize a configurable page, you select the global **Personalize Page** button to navigate to the **Page Layout Personalization** page. This page provides you with the ability to personalize the configurable page and immediately view the personalizations you make. Since the **Page Layout Personalization** page is intended only to provide a preview of your configurable page personalizations, you should not use the buttons or links rendered in the configurable region in this context to navigate to other pages, as they may not work as expected.
- You can also personalize a configurable page in the **Page Layout Personalization** page by launching it from the Functional Administrator Responsibility. However, if your configurable page contains controller code that depends on mandatory user-entered parameters, flow/business-logic, limited access to specific users and/or multi-organization access control, then you should not access the Personalization UI from the Functional Administrator responsibility. When the controller code is executed, the parameters that the controller code is expecting are only available from the URL or the flow or multi-organization access control that results when the page itself is launched from its intended flow. If the **Page Layout Personalization** page is launched from the Functional Administrator responsibility, these parameters may not be available and the page might fail with unexpected errors. To avoid this problem, you should access the Personalization UI for your configurable page using the global **Personalize Page** button on the page itself.

  This restriction applies only to configurable pages whose flexibleLayout and flexibleContent regions are defined in their base metadata. This restriction does not apply to pages that start out as non-configurable, but are later personalized by administrators who add new flexibleLayout and flexibleContent regions using the Create Item page in the Personalization UI. The Functional Administrator Responsibility always displays the **Page Hierarchy Personalization** page when it launches personalizations for these pages that have been made configurable via prior personalizations.
- If the controller of your configurable page depends on URL parameters and a user selects the **Personalize Page** link to personalize the configurable page, the **Page Layout Personalization** page will retain those URL parameters. As a result, when the user selects the **Return to Application** link to return to the configurable page, the URL parameters will be made available again to the configurable page's controller.

Known Issues

- See a summary of key issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
- Javadoc Files
- Lesson(s)
- Sample Code

Creating an End-User Personalizable Page

User Personalizable Pages

A user-personalizable page includes a special search region where users can define saved, named queries that include instructions for results column display and sorting.

- See Chapter 4's Search topic for instructions on creating end-user personalizable content.
- See the Oracle Browser Look-and-Feel (BLAF) UI Guidelines Search and Query Templates for additional information about the feature's design.

Known Issues

- See a summary of key Personalization issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
- Javadoc Files
- Lesson(s)
- Sample Code

Developer Information for Admin-Level Personalizations

Overview

As a developer, the following information may be relevant to you as you develop the pages in your application that can later be personalized by Oracle Administrators as well as end-users:

- Relationship Between Controllers and OA Personalization Framework
- Setting a Localization
- Function-Level Personalizations
- AM Parameter Registry

Relationship Between Controllers and OA Personalization Framework

How do personalizations affect the behavior of your controller code and vice versa? First, any personalizations that you make to a region are stored on top of the base Oracle JDeveloper OA Extension definitions of the region and do not overwrite it. Also remember that a controller determines the layout of a page.

In the case of Admin-level personalizations, when you create the web beans for a page, either in your controller code or setting the Add Indexed Children property to True in OA Extension, the web beans are created with the personalizations applied to them. If you alter any of the web beans' properties in your controllers after creating the web beans, your controller changes will override the personalizations applied to those web beans.

In the case of User-level personalizations, where a user selects and applies a predefined personalized view, the user's personalizations override any changes that you make in your controller.

Setting a Localization

Since product teams handle localization differently, they are responsible for setting the localization that needs to be effective for each page (or each `rootAM`). To initialize the context with a localization code, the product teams need to do the following:

Step 1: Make sure that the page's `rootAM` extends `OAApplicationModuleImpl`.

Step 2: The method `initializeWebValues` in their `rootAMImpl` returns a string that is used as the localization code for this page. Product teams need to override this method with their product specific logic. See: AM Parameter Registry.

Function-Level Personalizations

A function in Oracle E-Business Suite is a piece of application logic or functionality that is registered under a unique name for the purpose of assigning it to, or excluding it from, a responsibility. You can create standard personalizations for a region at the Function level so that the personalizations are effective only for users of a specific function. Once you create a function-level personalization, you can update it or delete it.

Note Oracle may deliver predefined Function-level personalizations. Customers can view but not update or delete "Oracle-seeded" Function-level personalizations.

Function-level personalizations are the highest level of personalizations you can make. Any further personalizations you make to the same region at lower Admin-levels always override the personalizations you make at the Function level.

To maintain function security, the function for which you are personalizing the region must be included in the responsibility from where users launch the page containing the region.

Once you create a function-level personalization, you can pass the function name corresponding to the personalized region in any of the following ways, in order of highest to lowest precedence:

- Specify the function name from the root Application Module using the method `initializeWebValues`. See the AM Parameter Registry section for additional information.
- Specify the function name on the URL using the parameter `OAFunc`. For example:

  ```
  http://<host>:<portID>/OA_HTML/OA.jsp?OAFunc=<custom_function>&...
  ```

  `OAFunc` can be used in two different ways:

  - If `OAFunc` is used without `akRegionCode/akRegionApplId`, then the function corresponding to it is launched.
  - If `OAFunc` is used in addition to `akRegionCode/akRegionApplId` or `page`, then it is used for setting the function context for that page. A function context should be set for the function personalization to take effect.

    For example, suppose you have the following URL that launches an Oracle Applications page that is defined as the **web_html_call** of function XYZ:

```
OA.jsp?OAFunc = XYZ
```

XYZ points to:

```
OA.jsp?page=oracle.apps.fnd.framework.webui.TestRegion
```

If you want a function-level personalization of ABC (defined using OA Personalization Framework) to apply to this page, you should change the **web_html_call** of function XYZ to:

```
OA.jsp?page=oracle.apps.fnd.framework.webui.TestRegion&OAFunc=ABC
```

- In OA Extension, search for the page layout region you just personalized. In the Property Inspector, set the **Function Name** property to the name of the function that you specified for the Function-level personalization.

Disabling Authorization for Function-Level Personalizations

You can disable the authorization required for a function-level personalization if the function-level personalization is set in a secured way. For example, when you embed a region from another product that includes a function-level personalization of your product page, you may find it useful to disable the authorization for that function-level personalization.

To disable the authorization for a function-level personalization, set the constant `oracle.apps.fnd.framework.OAFwkConstants.DISABLE_AUTHORIZING_FUNCTION_PERSONALIZATION` to `"Y"` during the `initializeWebValues` method in the root Application Module as follows:

```
public ArrayMap initializeWebValues (Hashtable paramValues)
{
  ArrayMap map = new ArrayMap();
  map.put(OAFwkConstants.FUNCTION_NAME, myPersonalizationFunction);
  map.put(OAFwkConstants.DISABLE_AUTHORIZING_FUNCTION_PERSONALIZATION, "Y");
  map.put(....);
  return map;
}
```

AM Parameter Registry

You can register parameters associated with any application module by seeding them in the table `AK_AM_PARAMETER_REGISTRY` using the HTML-based **AM Parameter Registry** page. You can later retrieve those parameters from the URL(Request) or Session with the `initializeWebValues` method.

Step 1: The **Application Module Parameter Registry** page is shipped as part of Oracle E-Business Suite Release 12 seed data, but is not associated with any given responsibility. To run this page, a system administrator must first attach the function **AK_AM_PARAMREGSTRY_SEARCH** to a menu and add that menu to a responsibility using the Oracle E-Business Suite **Menus** and **Responsibilities** screens.

Step 2: Use the Search region to search for parameters based on Application Name, Module Definition Name, Parameter Name or Source. You can update existing parameters or select the Create Record button to register new parameters. Choose **Apply** to save your changes.

Step 3: You can later retrieve the parameters for all context setting variables like `functionName` and `localizationCode` using the `initializeWebValues` method. The method `initializeWebValues` returns name-value pairs.

For example, suppose you want to set the `functionName` of the current page. You would need to do the following:

1. Override the method `initializeWebValues` in the application module associated with the current page.
2. Return an ArrayMap entry that has the function name of the page keyed by: `OAFwkConstants.FUNCTION_NAME`.

   Note that if you need to override the localization code as well, then return another entry for the localization code keyed by `OAFwkConstants.LOCALIZATION_CODE`.

   AttentionThe method `initLocalizationCode` has been deprecated, so you should use `initialWebValues` instead.

Known Issues

- See a summary of key Personalization issues with suggested workarounds if available.

Related Information

- BLAF UI Guideline(s)
  - None

- Javadoc Files
  - oracle.apps.fnd.framework.OAFwkConstants
  - oracle.apps.fnd.framework.server.OAApplicationModuleImpl
- Lesson(s)
  - None
- Sample Code
  - None

Chapter 10: Integrating With Other Products

Overview of Portlets

Portlets are reusable components you can insert into web pages created with Oracle WebCenter Portal, Oracle Portal, or any WSRP-compliant portal. They typically display summarized information and provide entry points into applications. Users of a portal can customize the pages they use, adding portlets that are relevant to their job or interests.

Note: Oracle WebCenter Portal 11*g* (11.1.1.9) is not certified with OA Framework Release 12.2.5.

Beginning in Release 12, existing OA Framework portlets, such as Applications Navigator, Favorites, and Worklist, as well as any new portlets that you create, are WSRP/JSR168-compliant. WSRP and JSR168 are two separate industry standard specifications used to render a portlet in a portal server:

- **Java Specification Request 168** (**JSR168**) is a portlet specification that establishes standard APIs for creating portlets. The standards help achieve interoperability between portals and portlets. By adhering to these standards when creating a portlet, you can deploy a portlet on any portal without much change.
- **Web Services for Remote Portlets** (**WSRP**) is a web services protocol for accessing and interacting with web applications from remote sources. WSRP provides standards that enable application providers to provide their services such that they can be easily plugged into any WSRP-compliant portal without any programming change.

As a result of being WSRP/JSR168-compliant, customers of Oracle E-Business Suite can now access Oracle E-Business Suite from any WSRP-compliant third-party portal server, by simply adding their Oracle E-Business Suite portlets into their (third-party) portal server.

As of Release 12.1.3, you can set the profile option FND: Enable Portlet Links in New Window to control whether an OA Framework portlet link selected from an Oracle WebCenter Portal or Oracle Portal Home page launches in a new named browser window or in the existing browser window. By being able to open the link in a new browser window, a user can interact with Oracle E-Business Suite content without altering the state of the Oracle WebCenter or Oracle Portal Home page. Without this support, the Oracle WebCenter Portal or Oracle Portal Home page would refresh with E-Business Suite content whenever a user selects a portlet link. Note that the E-Business Suite page launched in a new window will have a **Close Window** link instead of a **Home** link. This allows the user to explicitly close the E-Business Suite page after completing an E-Business Suite transaction. Alternatively, the user can select the browser window's **Close** icon (**X**) to close the window.

Note: As of Release 12.2, JPDK-type portlets are no longer supported.

Prerequisites

- To execute portlets created with OA Framework you must have access to an installation of any WSRP-compliant portal such as Oracle WebCenter Portal 11*g* or Oracle Portal 11*g*.

  Additional Information: To learn more about integrating Oracle E-Business Suite portlets with Oracle WebCenter Portal spaces, refer to My Oracle Support Knowledge Document 1332645.1, "Using WebCenter 11.1.1 with Oracle E-Business Suite Release 12.2".

  For Oracle E-Business Suite Release 12.2 new customers: You should only integrate Oracle E-Business Suite with Oracle WebCenter Portal. Follow the instructions provided in My Oracle Support Knowledge Document 1332645.1, "Using WebCenter 11.1.1 with Oracle E-Business Suite Release 12.2".

  For Oracle E-Business Suite customers upgrading from Release 12.0 or 12.1 to Release 12.2 and already have Oracle Portal 10*g* / 11*g* or Oracle WebCenter Portal 11*g* configured: Follow the upgrade/migration steps required, as described in the *Oracle E-Business Suite Upgrade Guide, Release 12.0 and 12.1 to 12.2*.

- Regions to be exposed as portlets must be created using OA Framework Release 12 or above.
- The remainder of this document assumes you have created and unit tested a region you want to expose as a portlet. Refer to the Implementing the View - Pages topic, and the OA Framework ToolBox Tutorial for general information about creating regions.

Contents

This document contains the following topics:

- Overview
  - Prerequisites
- Basic Implementation
- Test Your Portlet
- Implement Portlet Caching for Oracle WebCenter Portal
- Implement a Portlet-Specific Customization Page (Optional)
- Troubleshooting Portlet Problems
- Related Information

Basic Implementation

OA Framework lets you expose any standalone regions (**except pageLayout regions**) as portlets. Page layout regions cannot be exposed as portlets because portlets are always rendered inside an HTML table cell and page layout regions contain HTML constructs, such as footers, that cannot be rendered within a table cell.

Note: OA Framework Provider handles the form submission using HTTP GET as per the Java Portlet Specification. (Refer PLT.16.3 The Include Method - "Servlets and JSPs included from portlets must be handled as HTTP GET requests."). Due to this limitation, any regions requiring data entry (like search criteria regions, LOVs, text inputs, etc.) should not be exposed as portlets.

Figure 1: Example of a Portal page including OA Framework portlets.



Exposing an existing region as a portlet is very simple. To expose your region as a portlet:

Step 1: Create a UI Function to register your region as a portlet. See the "Creating Functions" section of the Tabs/Navigation topic for more information regarding the creation of UI Functions using the **Form Functions** form.

Step 2: Set the following properties within your new UI Function:

- **Function Type** = `Web Provider Portlet (internal value is WEBPORTLET)`

  Add a parameter to the portlet function definition.
  `# Parameters = OA_JSR168_PORTLET=Y`

- **HTML call** = `OAP.jsp?region=<your region Reference>`

  For example: `OAP.jsp?region=/oracle/apps/fnd/wf/worklist/webui/WorklistPrtletRG`

  If your portlet includes a drill-down page, append a parameter to the above URI specifying the target of the drill down using a page reference or a function name. For example:

  1. `&detailPage=<target page>`

     For example: `OAP.jsp?region=/oracle/apps/fnd/wf/worklist/webui/WorklistPrtletRG`
     `&detailPage=/oracle/apps/fnd/wf/worklist/webui/AdvancWorklistPG`

  2. `&detailFunction=<Form Function name>`

     For example: `OAP.jsp?region=/oracle/apps/fnd/wf/worklist/webui/WorklistPrtletRG` **`&detailFunction=WF_WORKLIST_DRILL_FUNC`**

- **MDS Reference Path** = `/oracle/apps/fnd/framework/customjrad/webui/CustomJRADViewUpdatePage` (if the portlet may be personalized with the OA Personalization Framework **Create View** or **Update View** page)

  or

  = `<JRAD reference to a personalization page>` (if the portlet may be personalized with a personalization page developed using OA Framework Release 12.0, as described below in Implement a Portlet-Specific Customization Page)

Step 3: Optionally disable portlet personalization.

By default, all portlets rendered using the OA Framework Web Provider include a **Customize** link in the title bar. Clicking on the link displays a customization page that lets you change the portlet title and responsibility. With this page you can customize the portlet title and assign a responsibility to it. Customizing the responsibility is useful when a portlet's content is dependent on the responsibility context, and is associated with multiple responsibilities.

To disable portlet customization, append `&dispRespCustPg=N` to the HTML call URI.

For example:

`OAP.jsp?region=/oracle/apps/fnd/wf/worklist/webui/WorklistPrtletRG`**`&dispRespCustPg=N`**

Step 4: Associate help text defined using Oracle online Help with your portlet.

When you associate help text with your portlet, the OA Framework Web Provider renders a help text icon (**?**) in the portlet title bar. Clicking on the icon displays the help text.

To associate help text with your portlet, append `&oawpHelpTarget=<Oracle online Help reference>` to the HTML call URI.

For example:

`OAP.jsp?region=/oracle/apps/fnd/wf/worklist/webui/WorklistPrtletRG`**`&oawpHelpTarget=worklistHelp`**

Test Your Portlet

After you create a UI Function for your portlet you are ready to test it.

To Test on Oracle WebCenter Portal:

Step 1: Grant access to your new UI Function by adding it to a menu, associating the menu with a responsibility (see Page Security for additional information), and assigning the responsibility to a user.

Step 2: Bounce the Apache listener. This step is necessary because the AOL/J API caches the function definitions and menu structures. Contact your system administrator to bounce the listener.

Step 3: Add an Oracle E-Business Suite Release 12 portlet to an Oracle WebCenter Portal page:

1. Log in to Oracle WebCenter Spaces - For information on how to log in, refer to the section titled, "Logging In to a WebCenter Application" in the *Oracle Fusion Middleware User's Guide for Oracle WebCenter 11g Release 1.*
2. Navigate to the page where you want to add a portlet.
3. Select **Edit Page**, from the **Page Actions** menu to open your page in Oracle Composer.
4. Select **Add Content** for the location where you want to place the portlet.
5. In the Oracle Composer Catalog that appears, select **Portlets** to open the portlets catalog.
6. Select **E-Business Suite WSRP Producer** (which is registered with Oracle WebCenter Portal as described in "Section 8: Register the E-Business Suite Portlet Producer with WebCenter" of the My Oracle Support Knowledge Document 1332645.1, "Using WebCenter 11.1.1 with Oracle E-Business Suite Release 12.2").
7. Select **Add** next to the portlet of interest (that is, to the UI Function that you registered in the Basic Implementation section above.
8. Select **Close** to close the Catalog.
9. In Oracle Composer, select **Edit**, in the header bar on the portlet instance, to edit the portlet's properties.
10. Navigate to the **Display Options** tab and select the **Edit** icon for the portlet property `Render Portlet In IFrame` to set its value to **true**. Select **OK** to close the Edit window.
11. Adjust other portlet properties, such as `Width` and `Height` as desired.
12. Select **OK** to exit the portlet Component Properties window.
13. Select **Save** in Oracle Composer to save your changes.
14. Select **Close** to exit Oracle Composer.

Step 4: Verify that the portlet you added in Step 3 renders properly on the Oracle WebCenter Spaces page.

Implement Portlet Caching for Oracle WebCenter Portal

OA Framework supports portlet caching for Oracle WebCenter Portal and other WSRP-compliant third-party portal servers. However JSR 168 only supports time- or expiry-based caching and not validation-based caching.

Using time- or expiry-based caching, the content of your portlet is cached by Oracle WebCenter Portal for a fixed period of time. Once the portlet is cached, subsequent requests for the same portlet are served directly from Oracle WebCenter Portal's cache until the time interval that you specify for your portlet expires. Once the time limit has expired, the next request for the portlet results in Oracle WebCenter Portal contacting the OA Framework Web Provider to refresh the content. The new content is cached again until the next cache time-out.

Expiration times termed as expiration-cache can be set in any of the following ways:

- During design time, in the deployment descriptor, as a metadata.
- During runtime, programmatically set the property on the response output to the WSRP-compliant portal server.

```
          RenderResponse rRes ....
rRes.setProperty("expiration-cache","0");
//0 - disable caching for the portlet
```

- In the portlet.xml file on the Oracle E-Business Suite middle-tier node, located under:

  $COMMON_TOP/webapps/oacore/html/WEB-INF

  Specify the following entry:

  ```
  expiration-cache/expiration-cache; (time in minutes)
  ```

  The expiration-cache element defines how long the portal/portlet-container should cache content created by the portlet.

Implement a Portlet-Specific Customization Page (Optional)

Note: This section focuses on customizations that end users can make to your portlet. For information about administrator personalization of your portlet content, see Personalizing Your Portlets.

While all portlets support customization of the portlet title and responsibility, you may want users of your portlet to be able to customize other aspects of your portlet's behavior, such as the data that is displayed or the format in which it is displayed. You can do this by defining a portlet-specific customization page and associating it with your portlet.

To create a portlet-specific customization page to capture the additional customization parameters for your portlet:

Step 1: Create one or more view objects for your portlet customization data as needed. Note that you must provide your own table(s) in your application for this purpose.

Step 2: Create an application module and associate your view objects with it. Add any logic that you require for your application module.

Step 3: Create a page with items that bind to the view object(s) you created in Step 1. Set the `AM Definition` on your **pageLayout** region to `oracle.apps.fnd.framework.customjrad.server.CustomJRADTableViewsAM` to ensure the customization data for the default portlet customization page is committed when the user applies any changes. Then, associate your application module with any direct child region of the **pageLayout** region. For example, if your page has the following structure, you would set **Region1** and **Region2's** `AM Definition` property to the fully qualified name of the application module you created in Step 2. You don't have to do anything special with nested regions beneath this level.

```
PageLayoutRN -- has CustomJRADTableViewsAM as the page's root application module
| -- Region1 -- uses your application module
     | -- Region3
     | -- Region4
| -- Region2 -- uses your application module
```

Step 5: Enter the URI of your portlet specific customization page in the **MDS Reference Path** field of your UI Function definition. Once a value is defined for the MDS Reference Path, users will be able to navigate to the portlet specific customization page using the `Next` button in the default portlet customization page.

For example:

```
/oracle/apps/fnd/framework/customjrad/webui/CustomJRADViewUpdatePage
```

Troubleshooting Portlet Problems

You will need to recreate the portlet.xml file if you define a new portlet or your portlet configuration changes. You can do this by performing the following steps. The example

commands are for UNIX platforms.

1. Source the `EBSapps.env` file on the *run* edition file system.
2. Change directory to `$OA_HTML/WEB-INF` by running the following command:

   ```
   $ cd $OA_HTML/WEB-INF
   ```

3. Run the following command to set the Java `CLASSPATH` environment variable:

   ```
   $ export CLASSPATH=$CLASSPATH:$FMW_HOME/oracle_common/webcenter/modules/com.bea.wsrp_10.3.2.0/system/portlet-container.jar
   ```

4. Run the following command to generate the portlet.xml file:

   ```
   $ java -DJTFDBCFILE=<dbc filename> oracle.apps.fnd.framework.jps.OAPortletAppConfigHandler
   ```

5. Source the `EBSapps.env` file on the *patch* edition file system and then perform Steps 2, 3, and 4 there.

Considerations in Multi-Node Environments

In a multi-node environment, a copy of the portlet.xml file will exist in every independent application tier file system.

If you have a multi-node environment, identify the applicable configuration in the following list, and complete the appropriate steps to regenerate the portlet.xml file(s).

- In a multi-node, non-shared file system environment, repeat steps 1-5 above on every application tier node.
- In a multi-node, single shared file system environment, repeat steps 1-5 above on the primary application tier node.
- In a multi-node, hybrid shared file system environment, where there are multiple application tier file systems, repeat steps 1-5 above on any one application tier node per file system.

Related Information

- Javadoc
  - `oracle.apps.fnd.framework.jps.*`

Integrating with Oracle ADF

Overview

You can integrate Oracle E-Business Suite with Oracle Application Development Framework (Oracle ADF). For detailed instructions, see the section titled "Integration with Application Development Framework", in the Developer Tools chapter of the *Oracle E-Business Suite Setup Guide*.

Embedding OBIEE Analytics in an OA Framework Page

Overview

You can embed Oracle Business Intelligence Suite Enterprise Edition (OBIEE) analytics in your OA Framework pages by creating a Rich Content Container item. For prerequisites and additional details on how to integrate the Rich Content Container with OBIEE, see My Oracle Support (formerly Oracle*MetaLink*) Knowledge Document 974422.1, "Embedding Analytics in Oracle E-Business Suite".

JTT/OA Framework Interoperability

Overview

JTT/OA Framework Interoperability is the supported, but restricted interaction between an OA Framework page or region and a JTT page. It provides an interim solution for migrating products written in JTT framework to OA Framework.

Note: Due to fundamental differences between OA Framework and JTT, this solution does not guarantee full functionality of either product. In fact, you should be aware of restrictions that apply before you build a "hybrid" application and test your application to ensure that it works fully in both frameworks. In some cases, you may expect your OA Framework region to work in both frameworks without modification. In most cases, however, you may need to either slightly alter the function or even modify the region/page itself for it to work in both frameworks.

The following table outlines the procedures necessary for page interaction:

	JTT Menu Structure	OA Framework Menu Structure
JTT to JTT	Follow your JTT guidelines. No modifications necessary.	Follow your JTT guidelines. No modifications necessary.
OA Framework to OA Framework	Use the **page** or **OAFunc** parameters with restrictions. The **OAHP** and **OASF** parameters may not be used.	Follow the OA Framework guidelines. No modifications are necessary.
JTT to OA Framework	Use **jtfcrmchrome.jsp** or your own application JSP with the **page** or **AKRegionCode** parameter. For example, `jtfcrmchrome.jsp?page=/x/y/z`. Follow your JTT guidelines to prepare the link or form. You may use `MenuLocation` to locate an OA Framework function that is defined by **OA.jsp**, which is then replace with **jtfcrmchrome.jsp**.	First, follow the OA Framework guidelines to construct the base URI. For example, `OA.jsp?page=/x/y/z`. Next, follow your JTT guidelines to prepare the link or form. You may use `MenuLocation` to locate an OA Framework function.
OA Framework to JTT	Although you may use `ServletSessionManager` or `MenuLocation` to prepare the links in the controllers, your OA Framework regions will not work under the OA Framework menu structure.	Use the **OAFunc** parameter to reach a JTT page by its function name. The JTT page must be contained in a function that is defined as type **INTEROPJSP**.

Attention: The following OA Framework functionality is not supported by JTT/OA Framework interoperability:

- Breadcrumbs
- Printable page
- Save model
- About page
- Processing page or page forwarding upon a PPR request

Contents

- Interoperability with FND Validation Level "Error"
- Launching an OA Framework Region from a JTT Application Menu
  - Declarative Implementation
    - Using the Standard Application JSP (jtfcrmchrome.jsp)
    - Using a Custom Application JSP
  - Run-time Behavior
  - Page Interaction
- Launching a JTT Page from an OA Framework Application
  - Declarative Implementation
  - Run-time Behavior
  - Page Interaction
- Known Issues

Interoperability with FND Validation Level "Error"

In OA Framework, applications are expected to operate with "FND Validation Level" set to "Error." Oracle E-Business Suite products that use JTT/OA Framework interoperability may need to make some adjustments for their applications to work properly with the "Error" validation level:

Important: The Oracle E-Business Suite Applications Technology (TXK) Release 12.2.2 removes the ability to turn the "FND

Validation Level" profile off. Oracle E-Business Suite Release 12.2.2 always runs as if this profile is set to **ERROR**.

1. Always test with the latest OA Framework release if possible. This ensures that you do not encounter a bug that is already fixed in a later release.

2. Always download the latest JTT/OA Framework interoperability patch as fixes in JTT are always rolled into this single JTT/OA Framework interoperability patch. You will not find fixes in mainline JTT patches (eg. JTT.E drops).

If your application has interaction between OA Framework regions and JTT pages, you should check the following:

1. A JTT page that sends a request directly or indirectly to an application JSP (such as, "jtfcrmchrome.jsp") should ensure the request URI is compliant. This includes both POST and GET requests. For example, if you have a HTML form that posts directly or indirectly to an application JSP, the action URI should be processed by ServletSessionManager.

   ```
   <form action="<%=ServletSessionManager.reMACUrl(
     "jtfcrmchrome.jsp", null)%>" method="POST">
   ...
   </form>
   ```

   `ServletSessionManager.getURL(String)` can also be used instead of `ServletSessionManager.reMACUrl(String, Hashtable)`.

   You may notice that even when the action URI is not an application JSP, the URI is still being validated. If the URI is invalid, no further validation is done by JTT or OA. This means even if you try to forward the request to a destination prepared by `ServletSessionManager.getURL(String)`, the validation will not be re-done. For example:

   ```
   <form action="myJTTPage.jsp" method="POST">
   ...
   </form>
   ```

   You will still see compliance mode-related exceptions when "myJTTPage.jsp" forwards the request to `ServletSessionManager.getURL("jtfcrmchrome.jsp")`. As mentioned earlier, the validation will fail as soon as the request is received by "myJTTPage.jsp" and since it fails, no further validation will be done on `ServletSessionManager.getURL("jtfcrmchrome.jsp")`. The OA Framework region rendered by "jtfcrmchrome.jsp" will throw an exception based on the fact that the validation on "myJTTPage.jsp" has failed.

2. If your product submits an OA Framework form ("defaultForm") from an application JSP to another (for example, "jtfcrmchrome.jsp" to "jtfcrmchrome.jsp"), you should be aware that **OAFunc** cannot be used in the form destination. By using **OAFunc**, the `processFormRequest` method of your controller will not be executed. This defeats the purpose of submitting the OA Framework form. This is not particularly related to compliance mode but a case like this will result in a compliance mode exception.

Launching an OA Framework Region from a JTT Application Menu

To allow interoperability between OA Framework and your JTT-based CRM product, you may implement your JTT-based CRM product menu to launch an OA Framework application page.

Note: When you post to an OA Framework page from a different technology such as JTT, OA Framework executes only the `processRequest()` phase in the target page. It does not execute the `processFormData()` and `processFormRequest()` phase. It is important that you do not work around this behavior; please design your application to accommodate this expected behavior.

Declarative Implementation

An OA Framework region must be rendered by an "application" JSP. To include a region in the menu tree, you must create a function whose HTML Call points to the application JSP and whose type is set to "JSP". The minimum standard parameters for an application JSP are "akRegionCode" and "akRegionApplicationId", or "page".

In OA Framework applications, the application JSP is "OA.jsp." In JTT applications, you can use one of the following JSPs:

- **Standard application JSP (jtfcrmchrome.jsp)** - if you want to render a region along with the JTT menu but nothing more. No configuration is required for this JSP.
- **Custom application JSP** - if your page needs extra content in addition to the region, create a custom application JSP so that you have full control over the content of the custom application page.

Using the Standard Application JSP (jtfcrmchrome.jsp)

If you use the standard application JSP, the HTML call for the function may include the same parameters that are typically specified with OA.jsp. For example:

```
jtfcrmchrome.jsp?akRegionCode=XYZ&akRegionApplicationId=690
```

If your URLs still use akRegionCode and akRegionApplicationId, you may optionally replace **jtfcrmchrome.jsp** with **OA.jsp** in the HTML call. This allows existing OA Framework functions to be plugged into the JTT application menu without changes, because at run-time, OA.jsp is replaced by jtfcrmchrome.jsp.

The same applies to URIs returned by `MenuLocation` objects. You may note that the actual HTML calls in the database still reference OA.jsp. When searching for a menu location, note that the actual HTML calls in the database still reference OA.jsp, not jtfcrmchrome.jsp. We recommend, however, that you use a function name as your search key rather than the HTML call, in this case.

Note that you cannot always replace **jtfcrmchrome.jsp** with **OA.jsp** in the HTML call. For example, if your URL uses the "page" or "OAFunc" parameters, you must use jtfcrmchrome.jsp instead of OA.jsp.

Note: If you plan to enable Quick Search or online Help for your region, you must include the extra URL parameter `jttoijappn` in the HTML call. The parameter `jttoijappn` sets the short name of an application. Both Quick Search and online Help rely on `appName` to be set properly on a JTT page. Since **jtfcrmchrome.jsp** is a JTT page that serves all kinds of different applications, you must tell the page in which application it should start the session by specifying this extra parameter. To enable Quick Search or online Help for the example above, you would specify the HTML call as:

```
jtfcrmchrome.jsp?akRegionCode=XYZ&akRegionApplicationId=690&jttoijappn=JTF
```

Generally, the parameters `akRegionApplicationId` and `jttoijappn` point to the same application. However, you may use other parameters rather than `akRegionApplicationId` with jtfcrmchrome.jsp, so you must include `jttoijappn` in your function HTML call if you have an online Help target set in your region or a Quick Search (Quick Find) set on your page.

Using a Custom Application JSP

To create a custom application page, you must follow the JTT framework page template with the use of **jtfframeworkincl.jsp** and **jtffwstylesheet.jsp**.

Note: Do not copy **jtfcrmchrome.jsp** as a template since it has been customized specifically for handling multiple applications and its implementation is considered private.

Step 1: Include **jtfframeworkincl.jsp** in your page.

Step 2: Specify your application page's page URI, which is usually the page name itself. Specify the page URI by setting a page context attribute named `JTF_PAGE_URI`.

Step 3: Include **jtfoaembedincl.jsp** in your page.

Step 4: Finally, at the location where you want the region to be rendered, include **jtfoaembed.jsp**.

Example:

```
...
<%@ include file = "jtfframeworkincl.jsp" %>
<%
  // Setting the application page URI
  pageContext.setAttribute("JTF_PAGE_URI", "myapplication.jsp");
%>
<%@ include file = "jtfoaembedincl.jsp" %>
...
<!-The region is rendered here -->
<%@ include file = "jtfoaembed.jsp" %>
...
```

Note: The region is rendered in a table of "100%" width within the page, therefore, you may not align any additional content to the region side by side.

Step 5: After you create the custom application, create the function for it. For example, the function HTML call may be:

```
myapplication.jsp?akRegionCode=<regioncode>&akRegionApplicationId=<id>
```

Run-time Behavior

Extra URI parameters may be observed when a function containing an application JSP is rendered as part of the JTT menu rendering process. When a region is rendered by the application JSP, all requests originated from the rendered region are directed to the application JSP and all necessary parameters for both JTT and OA Framework are attached.

Page Interaction

From a regular JTT Page to an application JSP

Since the application JSP is actually a JTT page, you can follow JTT coding guidelines to implement links or buttons that launch an application JSP from a JTT page. This includes using the `getURL` method of class `oracle.apps.jtf.base.session.ServletSessionManager` to create links, putting the application JSP URI as a form action, or using `MenuLocation` to search for a function that has an application JSP HTML call. For example, on a JTT page, you may have:

```
<a href="<%=ServletSessionManager.getURL("jtfcrmchrome.jsp?page=MyPage")%>">
 Take me to my page. </a>
```

In the case of `MenuLocation.search`, if your function has an HTML call that starts with **OA.jsp**, it will be replaced with **jtfcrmchrome.jsp**. We recommend that you always use function name as the key rather than the HTML call.

From an OA Framework region on an application JSP to a regular JTT page

Although this is possible, we do not recommended that you implement this type of page interaction. You can use `ServletSessionManager` or `MenuLocation` to prepare the JTT link, but you willl have a JTT dependency in your OA Framework controllers and your OA Framework regions will not work under the OA Framework menu.

Launching a JTT Page from an OA Framework Application

Declarative Implementation

Step 1: To launch a JTT function from an Oracle Application menu, you must change the function's type to **INTEROPJSP** (JSP Interoperable with Oracle E-Business Suite). The change in function type does not affect the validity of the function in JTT applications. If for any reason, you cannot modify the function's type, make a copy of the function and modify the copy's function type.

Note: When a JTT function is launched from an Oracle Application menu, the branding displayed on the page is inherited from OA Framework.

Step 2: Note that the JTT menu structure is not compatible with the Oracle Application menu structure. If you want to plug a JTT sub-menu tree into OA Framework, you must create a new sub-menu following the OA Framework guidelines. The new sub-menu should either have OA functions of type JSP or functions of type **INTEROPJSP**.

Run-time Behavior

Additional URI parameters may be observed when a JTT function is rendered in an OA Framework application. In addition, the query string of a request may sometimes be altered to meet OA Framework requirements, but it is guaranteed that all parameters in the request remain intact.

Page Interaction

From an OA Framework Region to a JTT Page

The JTT page must be contained in a function that has the function type **INTEROPJSP**. You can use the **OAFunc** parameter to navigate to the JTT function just as you navigate to any OA Framework function.

From a JTT Page to an OA Framework Region

You can simply point to **OA.jsp** along with the necessary parameters such as **page**. You must use `ServletSessionManager.getURL` on these OA Framework links just as you would with JTT links. You may also use `MenuLocation` to search for an OA Framework function and construct a URI or form action.

Known Issues

- See a summary of key JTT/OA Framework Interoperability issues with suggested workarounds if available.

Forms / OA Framework Page Integration

Overview

Both OA Framework and Oracle E-Business Suite provide support for the integration of Oracle Forms-based Oracle E-Business Suite forms with OA Framework-based pages. You can launch an Oracle E-Business Suite form from an OA Framework page as well as call an OA Framework HTML page from an Oracle E-Business Suite form.

Contents

- Launching Oracle E-Business Suite Forms from OA Framework Pages
- Launching OA Framework Pages from Oracle E-Business Suite Forms

Launching Oracle E-Business Suite Forms from OA Framework Pages

To launch an Oracle E-Business Suite form from OA Framework, you must first define a button, link or image web bean. The web bean then relies on the FormsLauncher applet provided by Oracle E-Business Suite (AOL/J) to launch the specified form.

Declarative Implementation

Step 1: In the OA Extension Structure pane, select the region in which you want to create the web bean to launch an Oracle E-Business Suite form. Choose **New** > **Item** from the context menu.

Step 2: Set the `ID` property for the item, in accordance with the OA Framework File Standards, and set the `Item Style` property to **button**, **image**, or **link**. You may also launch an Oracle Applications form from a submit button. See the Runtime Control section below for more details.

Step 3: Set the `Destination URI` property of the item with a value following this format (replacing the italicized text as appropriate):

form:*responsibilityApplicationShortName*:*responsibilityKey*:*securityGroupKey*:*functionName*

For example, if you want to launch the FND Menus form, the Destination URI property should be set to:

form:SYSADMIN:SYSTEM_ADMINISTRATOR:STANDARD:FND_FNDMNMNU

Step 4: If you wish to pass parameters to the form, set the `Destination URI` property with a value using the following format (Note that the parameter list is delimited by a space between each "parameter=value" pair):

form:*responsibilityApplicationShortName*:*responsibilityKey*:*securityGroupKey*:*functionName*:*param1=value1 param2=value2 param3=value3*

Note: If you wish to send varchar2 parameter values that contain spaces, use \" to enclose the string value. For example, to pass in something of the form:

```
TXN_NUMBER=LT INVOICE 1
```

Use:

```
TXN_NUMBER=\"LT INVOICE 1\"
```

Step 5: Refer to the following Chapter 4 topics for information about additional properties you may need to set for the specific item: Buttons(Action/Navigation), Buttons (Links), or Images in Your Pages.

Runtime Control

There are no special programmatic steps necessary to launch an Oracle E-Business Suite form from a button, image, or link in an OA Framework page. The OAButtonBean, OALinkBean and OAImageBean support the special form function URL format described above for the `Destination URI` property. When OA Framework encounters this special value, it generates the appropriate URL and also adds a hidden IFrame (inline frame) to the OA Framework page. The hidden IFrame is the target of the FormsLauncher applet provided by Oracle E-Business Suite.

Launching an Oracle E-Business Suite Form From a Submit Button

If you wish to launch an Oracle E-Business Suite form from a submit button in an OA Framework page, you must use the `OAPageContext.forwardImmediatelyToForm(String url)` method from `oracle.apps.fnd.framework.webui.OAPageContext`. An example of how to use this API is shown in the code sample below:

```
public void processFormRequest(OAPageContext pageContext, OAWebBean webBean)
{
 super.processFormRequest(pageContext, webBean);
 if (pageContext.getParameter("Apply")!=null)
 {
  String destination =
    "form:SYSADMIN:SYSTEM_ADMINISTRATOR:STANDARD:FND_FNDMNMNU";
  pageContext.forwardImmediatelyToForm(destination);
 }
}
```

Usage Notes

Microsoft Internet Explorer supports the IFrame element, so when you launch an Oracle E-Business Suite form from OA Framework, only a splash window appears. Any other windows required by the FormsLauncher applet use(s) the hidden IFrame as the target and therefore remain(s) hidden from the user. Firefox, on the other hand, does not support the IFrame element, so in addition to a splash window, the user also sees another window used by the FormsLauncher applet.

Launching OA Framework Pages from Oracle E-Business Suite Forms

Declarative Implementation

To launch an OA Framework page from Oracle Forms-based Oracle E-Business Suite forms, you must define a function to represent the OA Framework page. A function based on an OA Framework page is defined in exactly the same way as any other function in Oracle E-Business Suite. You define the function in the Oracle E-Business Suite Form Functions form and specify the function's `Type`, as **JSP**. Refer to the *Oracle E-Business Suite Developer's Guide* for more information. The function you define is automatically identified by the Oracle E-Business Suite menu and no further action is needed.

Runtime Control

If you wish to call the OA Framework page function directly from an Oracle Applications form, you must use the following Oracle E-Business Suite API:

```
PACKAGE FND_FUNCTION IS
procedure EXECUTE(function_name in varchar2,
                  open_flag     in varchar2 default 'Y',
                  session_flag  in varchar2 default 'SESSION',
                  other_params  in varchar2 default NULL,
                  activate_flag in varchar2 default 'ACTIVATE',
                  browser_target in varchar2 default NULL);
```

You can also pass additional parameters via `other_params` using a URL format, such as `name1=value1&name2=value2...`

For example:

```
fnd_function.execute( function_name => 'OKE_OKEKVCOM'
                          other_params  => 'headerid='||:parameter.k_header_id||
                          '&Ver1='||:compare_version.version1||
                          '&Ver2='||:compare_version.version2);
```

Note: There is no ampersand (&) in front of the first parameter name/value pair, but there is an ampersand in front of subsequent parameter name/value pairs.

Note: The `open_flag` and `session_flag` parameters are not shown in the above example because they are redundant for an HTML function.

Personalization Considerations

- See a summary of Forms / OA Framework Page Integration personalization considerations in the Oracle Application Framework Personalization Guide.

Touch Device Considerations

- See a summary of considerations for the Home page feature on touch devices in the Mobile Applications topic.

Known Issues

- None

Related Information

- BLAF UI Guideline(s)
- Javadoc File(s)
- Lesson(s)
- Sample Code

Appendix A: Summary of OA Component Properties

Overview

Refer to the *OA Component Reference* Help Topic in Oracle JDeveloper 10*g* OA Extension, or My Oracle Support Knowledge Document 1315505.1, *OA Extension Component Reference*.

Appendix B: Oracle Application Framework Profile Options

Overview

Any Oracle E-Business Suite profile options that affect OA Framework application behavior are grouped into the following categories and described below:

- Accessibility
- Access Keys
- Application Module Pooling
- Attachments
- Branding
- Browser
- Datepicker
- Defaulting
- Dialogs - Modal Pop-up
- Enhanced Rendering
- Export
- Frame Busting
- Gestures
- Home Page
- Inline Attachments
- Internationalization
- List Search
- Logging / Diagnostics
- Look-and-Feel
- LOV - Inline Look Ahead
- LOV - Popups
- Page Access Tracking
- Partial Page Rendering (PPR)
- Performance
- Personalization
- Portlets
- Preferences
- Record History
- Release-Specific Behavior
- Rich Interactions
- Session
- Tables
- Tabs/Navigation
- Validation
- Virus Scanning
- Web Server

You can use the Oracle E-Business Suite System Profile Options form to maintain all of the following, with the exception of the user internationalization settings. These are maintained in the Preferences page, which you can access from either the Personal Home Page or any OA Framework application page.

Accessibility

Profile Option / Internal Code	Description	Default Value
Self Service Accessibility Features / ICX_ACCESSIBILITY_FEATURES	Determines the level of accessibility support. • **Standard Accessibility** - pages are accessible to users using assistive technology. • **Screen Reader Optimized** - pages are optimized for screen readers. This may degrade the output for a sighted user. • **None** - pages include behaviors that are not accessible.	**None**

Access Keys

Profile Option / Internal Code	Description	Default Value
FND:DISABLE ACCESS KEYS / FND_DISABLE_ACCESS_KEYS	Determines whether access keys are enabled or disabled regardless of the accessibility mode. Valid values include: • **Yes** - Access keys are disabled in all pages • **No** - Access keys are enabled in all pages **Note**: The value of this profile option may be overridden by a user's setting of the **Disable Access Keys** checkbox in the Preferences page. Refer to the *Oracle E-Business Suite User's Guide* for more information.	No at Site level

Application Module Pooling

See the *OA Framework State Management* and *Application Module and Connection Pooling* topics in the *OA Framework Developer's Guide* for additional information about this feature.

Profile Option / Internal Code	Description	Default Value

FND: Application Module Pool Enabled / AMPOOL_ENABLED	Indicates whether pooling is enabled. If pooling is disabled (value is "No"), application module pool instances are destroyed when no longer needed by a page. In other words, if pooling is disabled, application module instances cannot be recycled. **Note:** Disabling application module pooling can lead to performance degradation. Application module pooling should always be enabled.	Yes
FND: Application Module Pool Monitor Sleep Interval / AMPOOL_MONITOR_SLEEP_INTERVAL	Application module pool monitor thread sleep interval in milliseconds. When the monitor thread wakes up at the specified intervals, it destroys available (inactive) application modules.	300000 (5 minutes)
FND: Application Module Pool Maximum Inactive Age / AMPOOL_MAX_INACTIVE_AGE	The time-out period in milliseconds for available, inactive application modules.	180000 (3 minutes)
FND: Application Module Pool Minimum Available Size / AMPOOL_MIN_AVAIL_SIZE	The minimum number of available application modules allowed per pool (low water mark). See How the Minimum and Maximum Profiles Work Together below.	0
FND: Application Module Pool Maximum Available Size / AMPOOL_MAX_AVAIL_SIZE	The maximum number of available application modules allowed per pool (high water mark). See How the Minimum and Maximum Profiles Work Together below. **Note:** The pool can contain as many *in-use* application module instances as your JDBC connections and memory configuration supports.	10
FND: Application Module Connection Pool Enabled / AMPOOL_CONNECTION_POOL_ENABLED	Indicates whether the connection associated with an Application module should be checked into the connection pool on AM checkin when application modules are pooled. If connection pooling is enabled (the profile option set to "Yes"), when the BC4J framework checks an application module back into the application module pool, OA Framework also checks its associated connection back into the connection pool. A connection is also checked into the connection pool when the application module is checked into the application module pool with its state "managed." In addition, if an application module's `Retention Level` is set to **MANAGE_STATE** or **CONNECTION_AGNOSTIC**, then its connection is checked in at the end of each request; even if the application module has not yet been checked in. When connection pooling is disabled (profile option set to "No"), connections are not checked back into the connection pool when application modules are checked into the application module pool. Additionally, connections are not released from any reserved application modules that have not yet been checked in. In other words, a connection remains dedicated to its owning application module until the connection is reclaimed by the connection harvester, or until the application module is destroyed because it is idle. With this mode, you can avoid reinitializing the connection upon check out so you can reuse any cached JDBC statements. **Note:** Setting this profile option value to "Yes" can cause unpredictable failures in your Oracle E-Business Suite applications if the applications depend on PL/SQL state stored on connections across requests. Since "Yes" mode aggressively releases connections, such state would be lost, leading to failures in products that depend on the state. Set this profile option value to "Yes" if and only if all your application flows run fine in this mode. Otherwise, retain the profile option value as "No", and seek other options, including better middle tier sizing, to handle concurrency.	No

How the Minimum and Maximum Profiles Work Together

This example best describes how these profile options work together. Assume that the "Low Water Mark" is set by the FND: Application Module Pool Minimum Available Size profile and the "High Water Mark" is set by the FND: Application Module Pool Maximum Available Size.

Example:

- Low Water Mark (Minimum Available Size) is 2
- High Water Mark (Maximum Available Size) is 10
- Number of Available Application Modules is 20 (5 of which have timed out)

- BC4J tries to clean up the timed-out application modules until the low water mark is reached, or until it has cleaned out all of the timed out application modules (whichever comes first). In this example, the monitor thread removes all the 5 timed-out application modules, so the pool of available application modules is reduced to 15.

- The monitor thread continues deleting available application modules until the high water mark is reached. In this example, it deletes another 5 application modules, leaving the number of available application modules in the pool at 10.

Attachments

Profile Option / Internal Code	Description	Default Value
Self-Service Oracle Files Enabled / FND_OA_ORACLE_FILES_ENABLED	Determines if enabled attachments within the application can also use Oracle Files Online (OFO) as a document repository. This value can only be set at the application level. The site level is seeded with a fixed value of **N**. Any value you set at the application level takes precedence over the value set at the site level.	N

FND: Attachment AutoVue Server / FND_AUTOVUE_SERVER	Use this profile option to set the end-point URL of the server where the AutoVue Document Print Service is configured. The URL should be of the form: `http://<host>:<port>/AutoVueWS/VueBeanWS`	None

FND: Attachment URL Validation / FND_ATTACHMENT_URL_VALIDATION	Use this profile option to enable the validation of URL-type attachments. Since URL-type attachments may point to external sites, this profile option allows you to enable a check to validate whether the URL is "safe". If the URL is not "safe", a confirmation dialog page appears ("You are being redirected to an external web page <*URL*>. Do you want to continue?"), prompting the user to review the URL before redirecting to the external site with an affirmative action.
	
	Any of the following URLs are considered "Safe":
	
	• A URL that starts with a value defined in the APPS_SERVLET_AGENT profile option. For example, if this profile option value is `http://machine1.example.com:8000/OA_HTML`, then all URLs starting with this server and port are considered "safe".
	• A URL that starts with a value specified in the `oracle.apps.fnd.sso.WebEntries` property. You can define this property in one of two ways.
	  ◦ In JDeveloper, navigate to the **Tools** menu > **Project Properties...** > **Run/Debug** > **Edit ...** and specify the `oracle.apps.fnd.sso.WebEntries` property and value in the **Java Options** text field (for example, `Doracle.apps.fnd.sso.WebEntries=http://www.example.com`).
	  ◦ In the deployed server, set `oracle.apps.fnd.sso.WebEntries` as a server start argument.
	
	This profile option can only be set at the Site level. Valid values include:
	
	• **None** - does not validate the selected URL-type attachment and redirects immediately to the URL site.
	• **Warn** - validates whether the selected URL-type attachment is a "safe" URL. If a URL is not "safe", a confirmation dialog page appears ("You are being redirected to an external web page <URL>. Do you want to continue?"), prompting the user to review the URL before redirecting to the external site with an affirmative action. This is the default value.
	• **Block** - validates whether the selected URL-type attachment is a "safe" URL. If the URL is not "safe", a message appears indicating the URL is blocked ("You are attempting to access an external web page <URL> from a site that is not allowed. Please contact your System Administrator.")
		Warn at Site level

Branding

See the Oracle Browser Look-and-Feel (BLAF) UI Guideline: Branding for additional information about this feature.

Profile Option / Internal Code	Description	Default Value
Corporate Branding Image for Oracle Applications / FND_CORPORATE_BRANDING_IMAGE	Controls the corporate branding image used in all OA Framework pages. If no value is set for this profile, OA Framework renders the corporate branding image by using /OA_MEDIA/FNDSSCORP.gif or by using the image specified for the corporateBranding element in a particular page's admin personalization. **Note:** This profile option also controls the corporate branding image for JTF-based applications. Refer to the "Implementing the Oracle CRM Technology Foundation" chapter of the *Oracle E-Business Suite CRM System Administrator's Guide* for more information. A valid value for this profile option is the name of a GIF file that contains the corporate branding image you wish to use. For example, CustomerImage.gif. The image file CustomerImage.gif should be placed under the $OA_MEDIA directory. As of Release 12.2.5, if you set a new corporate branding image with this profile, you can specify alternative text (alt-text) to describe your image file by updating the **Description** field of the **Corporate Branding Image for Oracle Applications** profile itself. This profile option should be set at the Site level.	None
FND: Branding Size / FND_BRANDING_SIZE	As of Release 12.2.4, this profile controls whether global buttons are displayed as links, icons or both in the Universal Global Header. Valid values include: • **Icons Only** - displays global buttons as icons only. • **Links Only** - displays global buttons as text links only • **Both Links and Icons** - displays global buttons as icons with text below the icons.	Icons Only

Browser

Profile Option / Internal Code	Description	Default Value
Force Page Refresh/ FND_FORCE_PAGE_REFRESH	Forces an Oracle E-Business Suite application page to refresh, by expiring the page from the browser cache, when a user presses the browser Back button. Force Page Refresh is enabled only if this profile's value is set to **Yes**. **Warning:** This profile option is exclusively used to resolve shared desktop security issues as stated in the General section of My Oracle Support Knowledge Document 395446.1, *Oracle Application Framework Troubleshooting Guide*, and My Oracle Support Knowledge Document 389422.1, *Recommended Browsers for Oracle Applications Release 12*. You should not not employ this profile option beyond its intended usage.	No

FND: Enable HTML5 Doctype / APPS_ENABLE_HTML5_DOCTYPE	Enables the uptake of the HTML5 standards doctype into generated HTML source by:
	• Adding the following HTML5 standards mode directive into the first line of every OA Framework-based page:
	`<!DOCTYPE html>`
	• Updating the `<head>` tag to include the following meta information for Internet Explorer:
	`<meta http-equiv="X-UA-Compatible" content="ie=edge">`
	By uptaking the HTML5 standard doctype, all OA Framework-based pages can run in a browser's preferred standards mode rather than in quirks mode or compatibility mode.Â Valid values include:
	• **No** - HTML5 standards doctype uptake is not enabled.
	• **Yes** - HTML5 standards doctype uptake is enabled.
	Note: This profile applies only to the Skyros Look-and-Feel in Release 12.2.4 and the Alta Look-and-Feel in Release 12.2.5, that is, the Oracle Applications Look and Feel profile option is null or set to **Skyros Look and Feel** or **Alta Look and Feel**, respectively.
	Caveat: With Internet Explorer 11 (IE11), all OA Framework-based pages revert to IE10 Standards mode due to multiple incompatibilities between OA Framework and UIX for IE11 Standards mode.
		Yes

Datepicker

Profile Option / Internal Code	Description	Default Value
FND: Enable Open Date Picker In Popup / FND_ENABLE_OPEN_DATEPICKER_IN_POPUP	Determines whether the Date Picker opens in a popup or a window. This profile option can be set at Site, Application, Responsibility and User levels.	True at Site level

Defaulting

For additional information about this feature, see the Defaulting section in the "Implementing the View" topic of the "Building an OA Framework Application (the Basics)" chapter in the *OA Framework Developer's Guide*.

Profile Option / Internal Code	Description	Default Value
FND: OA:Enable Defaults / FND_OA_ENABLE_DEFAULTS	Determines whether default values specified in personalizations and base metadata are applied to your pages. The specific rules of how default values are applied are discussed in the Defaulting section of the "Implementing the View" topic and in the Setting Default Values section of the *OA Framework Personalization Guide*. Valid values are **Y** (enable defaulting) and **N** (disable defaulting). This profile option may be updated at Site and Responsibility levels.	N

Dialogs - Modal Pop-up

Profile Option / Internal Code	Description	Default Value
FND: Enable Dialog Pages as Modal Popups / FND_ENABLE_DIALOG_PAGES_AS_MODAL_POPUPS	Indicates whether dialog pages (invoked using the `OAPageContext.redirectToDialogPage` API) are rendered as modal pop-ups. This profile can be set at the Site, Application, Responsibility and/or User levels. Valid values include: • **Yes** - Render dialog pages as modal pop-ups • **No** - Render dialog pages as standard OA Framework pages	Yes at Site level

Enhanced Rendering

Profile Option / Internal Code	Description	Default Value
FND: Enable JET Graphs and Charts / FND_ENABLE_JET_GRAPH_CHART	Enable JET Graph and Charts	JET Graph Only
FND: Enable JET Colorpicker / FND_ENABLE_JET_COLORPICKER	Enable JET Colorpicker	True at Site level

Export

Profile Option / Internal Code	Description	Default Value

Export MIME type / FND_EXPORT_MIME_TYPE	For data that is exported via the Export button, this profile defines the export file format (value separator), the content-type header of the response and the export file extension. Valid values include:
	• **Text/comma-separated-values** - Sets the following:
	  • content-type header = text/comma-separated-values
	  • export file extension = .csv
	• **Text/tab-separated-values** - Sets the following:
	  • content-type header = text/tab-separated-values
	  • export file extension = .txt
	• **None** - sets the following:
	  • content-type header = text/comma-separated-values
	  • export file extension = .csv
		None

FND: Unicode in Export / FND_UNICODE_IN_EXPORT	Switches the exporting unicode encoding as indicated if the profile FND_NATIVE_CLIENT_ENCODING is set to UTF-8. Valid values include: • **0 / UnicodeLittle If Excel Exists** - exports the file in UnicodeLittle if Microsoft Excel is detected on the client machine. This is the behavior of Release 12.1.2 and prior. • **1 / Always UnicodeLittle** - exports the file in UnicodeLittle regardless of whether Microsoft Excel is detected on the client machine. • **2 / Always UTF-8** - exports the file in UTF-8 regardless of whether Microsoft Excel is detected on the client machine.	0 / UnicodeLittle If Excel Exists

Frame Busting

Profile Option / Internal Code	Description	Default Value
FND: Disable Frame Busting / FND_DISABLE_FRAME_BUSTING	Determines whether Frame Busting mode is enabled or disabled at the Site, Application or Responsibility level. Valid values include: • **False** - Frame Busting mode is enabled • **True** - Frame Busting mode is disabled Enabling Frame Busting mode provides improved clickjacking protection of OA Framework-based application pages.Â Therefore, Oracle recommends you enable Frame Busting mode for all usages. However, if you have customizations that do not work in this mode, such as OA Framework-based pages embedded in third-party application frames that run in a separate domain, you may disable Frame Busting mode.Â In such cases, you should do so for as small a footprint as possible.Â For example, consider setting the FND: Disable Frame Busting profile to True only for responsibilities that use custom embedding. **Note:** The X-Frame-Options HTTP response header also provides clickjacking protection by ensuring a site's content is not embedded into other sites. For more information on how to enable or disable X-Frame-Options for all pages, refer to Chapter 9 in Part 2, Secure Configuration, of *Oracle E-Business Suite Security Guide*.	False at Site level

Gestures

Profile Option / Internal Code	Description	Default Value
FND: Enable Touch Gestures / FND_TOUCH_GESTURES_ENABLED	Indicates whether gestures support is enabled for touch devices. This profile can be set at the Site, Application, Responsibility and/or User levels. Valid values include: • **True** - Enable gesture support • **False** - Disable gesture support **Note:** The following profiles options must also be set accordingly to enable gestures support: • **FND: Enable Rich Table Interactions** must be set to **True** at the Site, Application and/or User level to allow users to perform rich interactions using gestures on touch devices. • **Oracle Applications Look and Feel / APPS_LOOK_AND_FEEL** must be set to the default value to display the rich table features.	True at Site level
FND: Enable Touch Gesture Diagnostics / FND_TOUCH_GESTURE_DIAGNOSTICS_ENABLED	Indicates whether a diagnostic pop-up for gestures is enabled. This profile can be set at the Site, Application, Responsibility and/or User levels. Valid values include: • **True** - Enable a gestures diagnostic pop-up • **False** - Disable a gestures diagnostic pop-up	None at Site level (treated as False)

Home Page

Profile Option / Internal Code	Description	Default Value

Self Service Personal Home Page Mode / APPLICATIONS_HOME_PAGE	**Note**: Prior to Release 12.2.9, the profile option APPLICATIONS_HOME_PAGE determines the look-and-feel of the Oracle Self-Service Applications Personal Home Page. With the change to use Masonry, the user preference set in the Settings > Preferences page takes precedence over the profile option value. The values set in this page for both home page and icon style are stored in the FND_USER_PREFERENCES table. If no value is set here, the value from the APPLICATIONS_HOME_PAGE profile option is used. The APPLICATIONS_HOME_PAGE profile option determines the look-and-feel of the Oracle Self-Service Applications Personal Home Page. As of Release 12.2.4, this profile now supports two new values: **Framework Tree** and **Framework Simplified**. Valid values include: • **Framework only** - when set to this value, OA Framework honors the value of the FND: Disable Configurable Home Page profile option to display the Home Page according to the behavior prior to Release 12.2.4. • **Framework Tree** - when set to this value, OA Framework honors the value of the FND: Disable Configurable Home Page according to the behavior prior to Release 12.2.4. • **Framework Simplified** - displays the 12.2.4 Simple Home Page. The following values are deprecated and no longer supported: • **Personal Home Page** - displays the earlier blue/gray personal Home Page displayed prior to Release 12.1. • **Personal Home Page with Framework** - displays a combination Home Page. Users log in to the old-style personal Home Page, but when they select a responsibility, the new OA Framework navigation page displays instead of the old blue/gray menu.	Framework only at Site level
FND: Disable Configurable Home Page / FND_CONFIG_HOMEPAGE_DISABLED	Determines whether the Configurable Home Page, with the tree-based Navigator, is enabled or disabled. Valid values include: • **False** - when set to this value, OA Framework displays the Configurable Home Page, with the tree-based Navigator available prior to Release 12.2.4. • **True** - when set to this value, OA Framework displays the Home page, with the flat list Navigator available in Release 12.1. Refer to the *Oracle E-Business Suite Setup Guide* and the *Oracle E-Business Suite User's Guide* for additional information about the Configurable Home page and this profile. **Note:** This feature was formerly controlled by the profile Enable Configurable Homepage / FND_CONFIG_HOMEPAGE_ENABLED, which is no longer used, but is retained for Release 12.1.2 backwards compatibility.	False at Site level
FND: Disable Navigator and Favorites Rich Menu / FND_DISABLE_NAVIGATOR_AND_FAVORITES_RICH_MENU	Determines whether the Navigator and Favorites Pull-down Menus feature is enabled or disabled. Refer to the *Oracle E-Business Suite Setup Guide* and the *Oracle E-Business Suite User's Guide* for additional information about the feature and this profile. **Note:** This feature was formerly controlled by the profile FND Slideout menu / FND_SLIDEOUT_MENU, which is no longer used, but is retained for Release 12.1.2 backwards compatibility.	False at Site level
Sign-On:Notification / SIGNONAUDIT:NOTIFY	• As of Release 12.2, this profile determines whether a tip message describing the number of open workflow notifications appears on a user's Home Page. Refer to the *Oracle E-Business Suite Setup Guide* and the *Oracle E-Business Suite User's Guide* for additional information about the feature and this profile. Valid values include: • **No** - Tip message listing open notifications is not displayed • **Yes** - Tip message listing open notifications is displayed	No at Site level
FND: Show Instance Name / FND_SHOW_INSTANCE_NAME	As of Release 12.2.6, the instance name can be displayed on the Home Page by setting the profile option FND_SHOW_INSTANCE_NAME to "True".	False at Site level

Inline Attachments

Profile Option / Internal Code	Description	Default Value
FND: Disable Inline Attachments / FND_DISABLE_INLINE_ATTACHMENTS	Determines, in conjunction with the `Inline Attachment Enabled` property of the attachment region item, whether the Inline Attachments feature is enabled or disabled at the Site or Application level. If the value of the `Inline Attachment Enabled` property is set to **default**, then the value of this profile option controls the display of the Inline Attachment pop-up UI at the site or application level with the following valid values: • **False** - Inline Attachments pop-up UI appears with an OnClick event • **True** - Attachment page UI appears with an OnClick event **Note:** The value of the `Inline Attachment Enabled` property set on the attachment region item overrides this profile option value.	False at Site level

Internationalization

Profile Option / Internal Code	Description	Default Value

Server Timezone / SERVER_TIMEZONE_ID	The time zone for the database server. It is assumed that all dates stored in the database will be interpreted relative to this time zone. This profile is only updatable at the Site level.	ïN/A
Client Timezone / CLIENT_TIMEZONE_ID	The time zone for the client (user). This profile is updatable only at the Site or User level. Fields that specify a date and time are queried and displayed to the user after automatically applying a time zone conversion as indicated by the Server Timezone and Client Timezone profiles. Conversely, a date and time entered by the user will undergo the opposite conversion before being stored into the database. **Note:** This value should be exactly the same as the timezone setting of the operating system of the machine from which the user is accessing the E-Business Suite instance.	N/A
ICX: Date format mask / ICX_DATE_FORMAT_MASK	The format used when displaying date fields. When a field displays both date and time, the date component is displayed in the format specified here, and the time component is displayed in a 24 hour format including hours, minutes and seconds (for example 14:45:30 for 45 1/2 minutes past 2:00 pm)	31-DEC-1999
ICX: Language / ICX_LANGUAGE	User session language.	American English
ICX: Territory / ICX_TERRITORY	User session territory or country.	United States
ICX: Numeric Characters / ICX_NUMERIC_CHARACTERS	The decimal character and group separator to be used for formatting numbers.	
FND: NATIVE CLIENT ENCODING / FND_NATIVE_CLIENT_ENCODING	The native encoding of the user's client operating system. The OA Framework file upload/download and data export features use this value as the file encoding.	WE8MSWIN1252

List Search

Profile Option / Internal Code	Description	Default Value
FND: Enable List Search / FND_ENABLE_LIST_SEARCH	Enable List Search	False at Site Level
FND: Enable Quick Search for Query Beans / FND_ENABLE_QUICK_SEARCH_QUERYBEANS	Enable Quick Search for Query Beans	True at Site Level
FND: Enable Infotile View for Query Beans / FND_ENABLE_INFOTILE_VIEW_QUERYBEANS	Enable Infotile View for Query Beans	True at Site Level

Logging / Diagnostics

See the Logging topic in the *OA Framework Developer's Guide* for additional information about this feature.

Note: A *Recommended Setting* value is listed in the Default Value column only if the two values differ.

Usage Guidelines

In normal operations, only UNEXPECTED (Level 6) errors requiring administrator attention should be logged. Also, users should have access to the Diagnostics page. These settings should be used at the Site level:

- FND_DIAGNOSTICS : No
- AFLOG_ENABLED : Yes
- AFLOG_MODULE : %
- AFLOG_FILENAME: null
- AFLOG_LEVEL : Unexpected

Profile Option / Internal Code	Description	Default Value
FND: Debug Log Enabled / AFLOG_ENABLED	Setting this to Yes enables the logging feature.	No **Recommended Setting:** Yes
FND: Debug Log Filename for Middle-Tier / AFLOG_FILENAME	Setting this to a complete filename causes the log messages to be written to a file rather than the database.	Not set (null)
FND: Debug Log Module / AFLOG_MODULE	Setting this limits logging to the specified module (fully qualified class name).	%
FND: Debug Log Level / AFLOG_LEVEL	The log level for messages that you would like to write to the database. OA Framework will log all messages with log levels greater than or equal to this profile option, so that the higher the value, the fewer messages that will be logged.	6 (UNEXPECTED)
FND: Diagnostics / FND_DIAGNOSTICS	Setting this to Yes causes a Diagnostics global button to render on every page. Select this button to view the log messages for the page. Enabling this profile also automatically renders the "About this page" link at the bottom of every OA Framework page. **Note:** Setting this to "Yes" also ensures that a detailed error stack can be viewed to help diagnose unhandled exceptions. When this is set to "No," a simple error message asking the user to contact the System Administrator is shown; the detailed error stack is not accessible.	No
FND: Developer Mode / FND_DEVELOPER_MODE	Enables the Edit Region global button. Also enables Developer Test Mode diagnostics.	Not set (null) **Recommended Setting:** No

FND: OA Framework REST Service Audit Enabled / FND_OAF_REST_LOG_ENABLED	Setting this to Yes enables REST logging by capturing the REST payload for input and output operations. Set this profile along with the profile FND: Debug Log Enabled to "Yes" to log the payload to the tables FND_LOG_MESSAGES and FND_LOG_ATTACHMENTS. **Note:** Irrespective of the log level set by the profile FND: Debug Log Level, REST logs always display at level 3.	No

Look and Feel

For additional information about this feature, see the Controlling UIX Rendering Output (Look and Feel / Facets) topic in the *OA Framework Developer's Guide*.

Profile Option / Internal Code	Description	Default Value
Oracle Applications Look and Feel / APPS_LOOK_AND_FEEL	Specifies the Look-and-Feel for all OA Framework-based pages. This value can be set at any of the Site, Application, Responsibility or User levels. Valid **Look and Feel** values for different releases include: • 12.2.5 and higher - **Alta** • 12.2.4 - **Skyros** • 12.2.3 - **Skyros** or **Swan Desktop** • 12.2.2 - **Swan Desktop** • 12.1.1 to 12.1.3 - **Swan Desktop** • 12.0 to 12.0.6 - **Swan Desktop** **Note:** You may also specify any Look-and-Feel (LAF) created in the Customizing Look-and-Feel (CLAF) UI. All new LAFs created in the CLAF UI get registered in a lookup table that this profile option reads. Examples include sample skins such as **Minimal Look and Feel** and **Simple Look and Feel** that are provided as basic references for custom skin development. **Note**: As indicated in the list of valid values above, each release of Oracle E Business Suite is associated with a standard LAF / skin. Oracle E-Business Suite patches are only provided for the standard LAF for that release. Oracle does not provide Oracle E-Business Suite patches for any other LAF, including custom or sample skins such as **Minimal Look and Feel** or **Simple Look and Feel** that are provided as basic references for custom skin development.	Not set (null), which implies the **Alta Look and Feel** - Oracle's corporate browser Look-and-Feel for Release 12.2.5.

LOV - Inline Look Ahead

Profile Option / Internal Code	Description	Default Value
FND: Disable Look Ahead LOV / FND_DISABLE_LOOK_AHEAD_LOV	Determines whether the Look Ahead LOV feature is enabled or disabled at the Site or Application level. Valid values include: • **False** - Look Ahead LOV feature is enabled • **True** - Look Ahead LOV is disabled **Note:** The value of the `Look Ahead Enabled` property set on the messageLOVInput item overrides this profile option value.	False at Site level
FND: Minimum Characters for Look Ahead / LOOK_AHEAD_MIN_CHARS	If the Look Ahead LOV feature is enabled, then this profile determines the minimum number of characters a user must enter to activate the Look Ahead LOV window. This profile option can be set to a value of **1** or higher at the Site or Application level. **Note:** The value of the `Minimum Characters For Look Ahead` property set on the messageLOVInput item overrides this profile option value. **Note:** For smaller width **messageLovInput** items (whose width is less than or equal to 5), the minimum number of characters required to activate the Look Ahead LOV has a default of 1.	3 at Site level

LOV - Popups

Profile Option / Internal Code	Description	Default Value
FND: Enable Open LOV In Popup / FND_ENABLE_OPEN_LOV_IN_POPUP	Determines whether a LOV (any type) is opened in a popup or window. **Note:** To open a LOV in a window (as in previous releases), set the profile option to False. Can be set at the Site, Application, Responsibility and User levels.	True at Site level

Page Access Tracking

For additional information about this feature, see the Page Access Tracking topic in the *OA Framework Developer's Guide*.

Profile Option / Internal Code	Description	Default Value
JTF_PF_MASTER_ENABLED / JTF_PF_MASTER_ENABLED	Master On/Off switch for Page Access Tracking. LOV displays list of valid values which include, among others, **true** or **false.**	If this profile is not set, the value is interpreted as **false**.

JTF_PF_ENABLED / JTF_PF_ENABLED	Determines if Site/Application/Responsibility/User has Page Access Tracking turned on/off. LOV displays list of valid values which include, among others, **true** or **false.**	Not Set (null) If this profile is not set, the value is interpreted as **false**.
JTF_PF_LEVEL / JTF_PF_LEVEL	As a bitmask, this profile determines what information to log when Page Access Tracking is on for a particular access. LOV displays list of valid values which include, among others: • **22** - Log session information (Client Browser, Language, and HTTP Header). • **118** - Log session Information and cookies. • **254** - Log session information, cookies, and URL parameters. • **126** - Log session information, cookies, and all parameters.	22
JTF_PF_FLUSH_INTERVAL / JTF_PF_FLUSH_INTERVAL	Determines how often Page Access data is flushed to the database (in seconds).	If this profile is not set, the default is **120** seconds.
JTF_PF_BUFFER_SIZE / JTF_PF_BUFFER_SIZE	Determines how often Page Access data is flushed to the database (in number of page accesses).	50

Partial Page Rendering (PPR)

See the Dynamic User Interface topic in the *OA Framework Developer's Guide* for additional information about this feature.

Profile Option / Internal Code	Description	Default Value
FND: Disable Partial Page Rendering / FND_PPR_DISABLED	Controls whether partial page rendering is disabled. If this profile option is *not* set, the "No" value is assumed. Valid values include: • **No** - Partial page rendering is enabled. • **Yes** - Partial page rendering is disabled. In this case, UIX automatically renders a **Go** button next to each item with a partialAction enabled.	The profile option is not set by default.
Enable PPR Debugging / FND_ENABLE_PPR_DEBUGGING	Controls the partial page rendering debugging feature. When this profile option is enabled, the partial targets are displayed at the top of the screen as the page renders. This profile option can be set at all Levels. The site value is **N** by default. Valid values include: • **N** - (Default) partial page rendering debugging is disabled. • **Y** - Partial page rendering debugging is enabled.	This profile option is disabled by default.
FND: Disable PPR Event Queuing/ FND_PPR_EVENT_QUEUE_DISABLED	This profile option enables/disables PPR Event Queuing (by mouse clicks and hotkeys). PPR Event Queuing is enabled by default, value **No** ('N'). You can set the value to **Yes** ('Y') to disable PPR Event Queuing. **Note:** See the PPR Event Queuing section of the Dynamic User Interface document for information about how to enable event queuing for a specific page. This option can be set at all Levels. The site value is **No** ('N') by default. Valid values include: • **No** - (Default) PPR event queuing is enabled • **Yes** - PPR event queuing is disabled	PPR event queuing is *enabled* by default.
FND: Disable PPR Auto Tabbing / FND_PPR_DISABLE_AUTOTABBING	Controls the partial page rendering auto-tabbing feature. When this profile option is set to **N** and the auto-tabbing feature is enabled, tabbing out of a text input field that triggers a PPR event takes the focus to the next field on the page. This profile option can be set at all Levels. The site value is **N** by default. Valid values include: • **N** - (Default) partial page rendering auto-tabbing is enabled • **Y** - partial page rendering auto-tabbing is disabled	This profile option is enabled by default.

Performance

Profile Option / Internal Code	Description	Default Value
Upload File Size Limit / UPLOAD_FILE_SIZE_LIMIT	Specifies the maximum allowable file size in KB for uploaded attachments. For example, if the limit is 2MB, this value should be set to 2000 or 2000K.	

ICX: MatchCase View / ICX_MATCHCASE_VIEW	This is available for ICX Web Inquiries pages only. This profile option controls the "Match Case" checkbox in the "Advanced Search" region of Web Inquiries. Setting this profile option value to "Checked" or "Hidden" helps avoid running poor performing queries which would normally disable indexes using an upper() clause. This profile option can be set at all levels. Valid values include: • **Unchecked** - the Match Case checkbox will be rendered in an unchecked state • **Checked** - the Match Case checkbox will be rendered in a checked state • **Hidden** - the Match Case checkbox will NOT be rendered, and it will behave as if checked. Instead of the checkbox, the UI displays a message that says "Match Case has been selected for you."	Unchecked
FND: View Object Max Fetch Size / VO_MAX_FETCH_SIZE	Limits the number of rows that any user can fetch in a query, thereby limiting the amount of memory he/she consumes. You can reduce this number to ensure that the middle-tier resources are shared evenly across all of your users. **Note:** This profile option can be set at the application level. If you are running applications for which the default 200 rows limit is too low, you can set a higher value for this particular application.	200

Personalization

See the OA Framework Personalization Guide for additional information about this feature.

Note: A *Recommended Setting* value is listed in the Default Value column only if the two values differ.

Profile Option / Internal Code	Description	Default Value
Personalize Self-Service Defn / FND_CUSTOM_OA_DEFINITION	This is intended for system administrators who wish to personalize regions at the localization, site, verticalization, org and responsibility levels. On enabling this profile option for the administrator, every OA Framework page will contain a global Personalize button. By clicking on this global button, the administrator can personalize the regions available on that page. **Note:** As of Release 12.1.3, the "About this page" link automatically renders at the bottom of every OA Framework page if you enable this profile option.	No **Recommended Setting**: Yes for System Administrators, No for other users
Disable Self-service Personal / FND_DISABLE_OA_CUSTOMIZATIONS	This is a system profile option specifically created for use by Oracle Support. You can set this profile option to "Yes" or "No" at the site or application level. If this system profile option is set to Yes, any personalizations made by the customer, regardless of the level at which the personalizations were made, will not be applied. All pages using OA Framework will now display the regions based on their original definitions. **Note:** When this profile is set to "Yes", a warning message that all personalizations are disabled is displayed on every page to which a user navigates.	No
FND: Personalization Seeding Mode / FND_PERSONALIZATION_SEEDING_MODE	*For Oracle E-Business Suite development use only.* Replaces the deprecated profile, FND_CREATE_SEEDED_PERSONALIZATIONS. When this profile is set to "Yes", it sets the `developerMode` flag to "Yes" for any function and user personalization created in Oracle E-Business Suite development. These personalizations are then protected against update/delete at the customer's site.	Not set (null)

FND: Personalization Region Link Enabled / FND_PERSONALIZATION_REGION_LINK_ENABLED	Enables the "Personalize Region" links on a page if the Personalize Self-Service Defn / FND_CUSTOM_OA_DEFINTION profile is set to Yes. Valid values: • **Yes** - renders the "Personalize Region" links above each region in a page. Each link takes you first to the Choose Personalization Context page, then to the Page Hierarchy Personalization page with focus on the region node from which you selected the "Personalize Region" link. *The scope is always set to the region itself. You can not navigate up to the region's parent.* For example, the "Personalize Region" link for /oracle/apps/abc/xyz.region1 focuses the HGrid on region1. No locator breadcrumbs are provided for you to navigate up to region1's parent. If a region on the page extends another region, the personalization context is the region being extended. For example, if /oracle/apps/abc/xyz.region1 extends /oracle/apps/abc/SharedRegionX, the scope is /oracle/apps/abc/ShareRegionX. This means the personalization occurs on ShareRegionX if you personalize region1. • **No** - "Personalize Region" links are not rendered. You must select the global Personalize button to personalize the page. • **Minimal** - renders the "Personalize Region" links, but minimizes the number of links displayed. The rule for minimization is that if a region's direct children all display a "Personalize Region" link when the profile value is set to "Yes", then the region itself will not display a link when the profile value is set to "Minimal". The scope is always set to the document where the region node resides. For example, if the region is /oracle/apps/abc/xyz.region1, the scope of the personalization context will be /oracle/apps/abc/xyz and the Page Hierarchy Personalization page will focus on region1. When the "Personalize Region" links are minimized, you can always navigate up to a region's parent, grandparent, and so on, using the locator breadcrumbs shown on the HGrid. Each link takes you to the Choose Personalization Context page, then to the Page Hierarchy Personalization page with focus on the region node from which you selected the "Personalize Region" link. If a region on the page extends another region, the personalization context still remains the current page. For example, if /oracle/apps/abc/xyz.region1 actually extends /oracle/apps/abc/SharedRegionX, the scope remains /oracle/apps/abc/xyz. This means a per-instance personalization occurs if you personalize region1. If you wish to personalize the shared region, you can navigate to the Choose Context page and select the shared region as the scope. **Note:** Enabling the "Personalize Region" links allows users to also personalize regions that are dynamically added to the page from custom code in the controller (that is, regions added using `createWebBean(OAPageContext pageContext, String reference, String name, boolean isMDS)` in the `oracle.apps.fnd.framework.webui.OAWebBeanFactory` class). These dynamically-added regions always display "Personalize Region" links, even if the profile value is set to Minimal.	No (As of Release 12.2.5)
Fnd Xliff Export Root Path / FND_XLIFF_EXPORT_ROOT_PATH	Use this profile option to set the root path used to generate the full path where the Xliff files are exported to when users extract their translated personalizations using the Extract Translation Files page in OA Personalization Framework. The permissions for the root path directory that you specify must be set to read, write, create for all users, using `chmod 777 [dir_path].`	Not set (null)
Xliff Import Root Path / FND_XLIFF_IMPORT_ROOT_PATH	Use this profile option to set the root path used to derive the full path from where the Xliff files are uploaded when users use the Upload Translations page in OA Personalization Framework to upload translated personalizations.	Not set (null)
FND: Personalization Document Root Path / FND_PERZ_DOC_ROOT_PATH	Use this profile option to define the root path where personalizations documents are **exported to** or **imported from** when users use the Database page or the File System page of the Functional Administrator responsibility's Document Manager, respectively. We recommend you set this profile to the **$APPL_TOP** staging area (`$APP_TOP/<CompanyIdentifier>/<CustomProductShortName>/<product-version>/mds/webui`) of the current deployed environment, where personalization documents are to be imported from or exported to. This profile option should be set at the Site level.	/tmp/custdocs

FND: Enable Personalization Workbench / FND_ENABLE_PERSONALIZATION_WORKBENCH	As of Release 12.2.7, use this profile option to enable the Personalization Workbench UI. Valid values: • **Yes** - the Personalize Page link under the global **Settings** icon navigates to the Personalization Workbench UI. • **No** - the Personalize Page link under the global **Settings** icon navigates to the classic Page Hierarchy Personalization page. **Note:** If this profile option is set to **Yes**, the value of the **FND: Personalization Region Link Enabled** profile is ignored as **Personalize Region** links are not supported when the Personalization Workbench is enabled.	No
FND: User Theme (FND_USER_THEME	The value of this profile option determines the theme used in Oracle Application Framework pages. This profile option can be set at the site, application, responsibility, or user level.	ï
FND: Message Component Layout Preferred Width Percentage / FND_MESSAGECOMPONENTLAYOUT_PREFERRED_WIDTH	Sets user-preferred width for message component layout. For example, 50% of default.	N/A

Portlets

Profile Option / Internal Code	Description	Default Value
FND: Enable Portlet Links in New Window / FND_PORTLET_LINK_NEWWINDOW_ENABLED	Controls whether an OA Framework page launched from a link in a portlet opens in a new browser window or refreshes the base portal page. This profile option can only be set at the Site level. Valid values include: • **Yes** - OA Framework page corresponding to a link in a portlet launches in a new browser window • **No** - OA Framework page corresponding to a link in a portlet launches in the same window as the base portal page	No at Site level

Preferences

See the Buttons (Global topic in the *OA Framework Developer's Guide* for additional information about this feature.

Profile Option / Internal Code	Description	Default Value
FND: Show General Preferences Flag / FND_SHOW_GEN_PREF	Controls whether the "General Preferences" menu entry is displayed when the user selects the "Preferences" global button. Valid values include: • **Yes** - "General Preferences" are displayed • **No** - "General Preferences" are hidden	Yes

Record History

Profile Option / Internal Code	Description	Default Value
FND: Record History Enabled / FND_RECORD_HISTORY_ENABLED	Enables the Record History icon for a Header, Table or Advanced Table region that has the Record History feature implemented.	No

Release-Specific Behavior

Profile Option / Internal Code	Description	Default Value
FND: Migrated to JRAD / FND_MIGRATED_TO_JRAD	Specifies if the application has been migrated to run against the MDS repository (or if it was created using the Oracle9*i* JDeveloper or Oracle JDeveloper 10*g* OA Extension). Valid values are "Y" and "N". If the value is "Y", then the page will be run from the MDS repository. Otherwise, page definitions are retrieved from the AK repository. **Note:** You do not have to set the this profile provided you are using the OA.jsp?page=/oracle/apps/... or OA.jsp?OAFunc=<FunctionName> syntax for function Web HTML Call values. **Note:** This value can be set only at the application level. The site level value has a fixed, seeded value of "N." The application level value will take precedence over the site level value. For the application level value, OA Framework uses the application associated with the page definition (not the responsibility) to read this profile option.	N

Rich Interactions

Profile Option / Internal Code	Description	Default Value

FND: Enable Rich Table Interactions / FND_ENABLE_RICH_TABLE_INTERACTIONS	Indicates if rich UI features for classic tables, advanced tables, and HGrids are enabled. Rich interactions include column resizing, column reordering, horizontal scrolling, and detaching table. This profile can be set at the Site, Application, and/or User levels. Valid values include: • **True** - Rich interactions are enabled • **False** - Rich interactions are disabled **Note**: As a prerequisite for export to Excel or PDF, the FND: Enable Rich Table Interactions profile option must be enabled.	True at Site level

Session

Profile Option / Internal Code	Description	Default Value
ICX: Limit time / ICX_LIMIT_TIME	Indicates the amount of time after which the user, regardless of their level of activity, will be asked to validate their credentials in order to continue working.	4 (There is no recommended value, this only needs to be set if you want to implement the behavior described.)
ICX:Session Timeout / ICX_SESSION_TIMEOUT	Maximum idle time for the Oracle E-Business Suite user session (specified in minutes).	

Tables

Profile Option / Internal Code	Description	Default Value
FND: Enable BNE Table Integration / FND_ENABLE_BNE_TABLE_INTEGRATION	Introduced in Release 12.2.6, this profile allows you to enable the export of table data from OA Framework classic or advanced tables without having to implement an Export button. When enabled, an export icon automatically renders in the table control bar, indicating that the user can download the table data as a Microsoft Excel spreadsheet. Valid values include: • **True** - Enables export of table data as an Excel spreadsheet • **False** - Disables export of table data as an Excel spreadsheet	False at Site level
FND_ENABLE_SIMPLIFIED_TABLE_CONTROL_BAR	Introduced in Release 12.2.9, this profile option can be set to **False** by system adminstrators to force display of the menu choices in the toolbar itself. The profile option can be set at the Site, Responsibility, or User level.	True at Site level

Tabs / Navigation

Profile Option / Internal Code	Description	Default Value
FND: Top-Level Menu Display Mode / FND_TOP_LEVEL_MENU_DISPLAY_MODE	Introduced in Release 12.2.5, this profile indicates the display mode of a "Level 1" menu for navigation between pages. Valid values include: • **Links Only** - renders the menu and page layout in the classic mode, where the top-level ("Level 1") menu appears as tabs with links. • **Icons and LInks** - renders the Release 12.2.5 enhanced menu UI, where the top-level ("Level 1") menu appears as icons and links. • **Icons and Links on Tablets Only** - renders the Release 12.2.5 enhanced menu UI, where the top-level ("Level 1") menu appears as icons and links for tablets only. On desktops, the top-level ("Level 1") menu appears as the classic tabs with links.	Links Only at Site level

Validation

See the Security topic in the *OA Framework Developer's Guide* for more information about the validation features inOA Framework.

Profile Option / Internal Code	Description	Default Value
FND Validation Level / FND_VALIDATION_LEVEL	Controls whether accelerated validation is enforced for the Oracle E-Business Suite. Valid values include: • **None** - accelerated validation is disabled. • **Error** - accelerated validation is enabled. Errors are displayed as runtime exceptions. This value can be set on any release that supports setting the value. • **Log** - additional validation is enabled. Errors are logged without a runtime interruption. Note that Logging must be enabled. This profile option can be set only at the site level, and the Java Virtual Machine (JVM) must be bounced to see any changes. In addition, the database should also be bounced as some caching occurs there. **Important:** Oracle E-Business Suite Applications Technology (TXK) Release 12.2.2 removes the ability to turn this profile off.Â Oracle E-Business Suite Release 12.2.2 and later alway runs as if this profile is set to **ERROR**.	Error **Note:** As per Chapter 10 in Part 2, Secure Configuration, of *Oracle E-Business Suite Security Guide*. we strongly recommend that this profile option remains set to **Error**. For more information about this profile, refer to My Oracle Support Knowledge Document 946372.1, *Secure Configuration of E-Business Suite Profiles*.

FND Function Validation Level / FND_FUNCTION_VALIDATION_LEVEL	Controls whether additional accelerated validation above the existing function validation is enforced for Oracle E-Business Suite. Valid values include: • **None** - additional validation is disabled. • **Error** - additional validation is enabled. Errors are displayed as runtime exceptions. This value can be set on any release that supports setting the value. • **Log** - additional validation is enabled. Errors are logged without a runtime interruption. Note that Logging must be enabled. This profile option can be set only at the site level, and the Java Virtual Machine (JVM) must be bounced to see any changes. In addition, the database should also be bounced as some caching occurs there. **Note:** This profile may be overridden with a corresponding JVM setting. **Important:** The Oracle E-Business Suite Applications Technology (TXK) Release 12.2.2 removes the ability to turn this profile off.Â Oracle E-Business Suite Release 12.2.2 always runs as if this profile is set to **ERROR**.	Error **Note:** As per Chapter 10 in Part 2, Secure Configuration, of Oracle E-Business Suite Security Guide. we strongly recommend that this profile option remains set to **Error**. For more information about this profile, refer to My Oracle Support Knowledge Document 946372.1, *Secure Configuration of E-Business Suite Profiles*.
Restrict text input / FND_RESTRICT_INPUT	Controls whether cross-site scripting scanning is turned on or off at the Site level. Valid values include: • **Yes** - scanning is enabled • **No** - scanning is disabled **Note:** When this profile option is not set (null), scanning is enabled.	Yes
Framework Validation Level / FRAMEWORK_VALIDATION_LEVEL	Controls whether page function validation features are enabled for OA Framework pages. Valid values include: • **Error** - page function validation is enabled. Errors are displayed as runtime exceptions. • **Log** - page function validation is enabled. Errors are logged without a runtime interruption. Note that Logging must be enabled. • **None** - page function validation is disabled. This profile option can be set only at the site level, and the Java Virtual Machine (JVM) must be bounced to see any changes. In addition, the database should also be bounced as some caching occurs there. **Important:** The Oracle E-Business Suite Applications Technology (TXK) Release 12.2.2 removes the ability to turn this profile off.Â Oracle E-Business Suite Release 12.2.2 always runs as if this profile is set to **ERROR**.	Error **Note:**As per Chapter 10 in Part 2, Secure Configuration, of Oracle E-Business Suite Security Guide. we strongly recommend that this profile option remains set to **Error**. For more information about this profile, refer to My Oracle Support Knowledge Document 946372.1, *Secure Configuration of E-Business Suite Profiles*,
FND: Fixed Key Enabled / Â FND_FIXED_KEY_ENABLED	Controls whether a fixed encryption key is used for accelerated validation. This profile should be set at the user level. Valid values include: • **Y** - enables a fixed encryption key, as defined by the FND: Fixed Key profile, to be used by accelerated validation for all of a user's sessions. • **N** - a unique encryption key for accelerated validation is used for each new user session.	Not Set (null) Only if this profile is set to **Y** will a fixed encryption key be used.
FND: Fixed Key / FND_FIXED_SEC_KEY	The fixed security key to use for accelerated validation if FND: Fixed Key Enabled is set to **Y** for the user. This profile should be set at the user level. Define the key as a hexadecimal string of size 64. If the key is not a hexadecimal string, session creation will fail.	None

Virus Scanning

Profile Option / Internal Code	Description	Default Value

FND: Disable Virus Scan / FND_DISABLE_VIRUS_SCAN	Determines whether virus scanning occurs for file type documents that are uploaded. This profile can be set at the Site, Responsibility, Application, or User level. Valid values include: • **Yes** - Virus scanning is disabled • **No** - Virus scanning is enabled **Note**: The value of the `Virus Scan Enabled` design-time property for the File Upload features and Attachment feature may override the value of this profile option. **Note**: The profiles "FND Attachment Antivirus Software Type" and "FND Attachment Antivirus Server" must be set appropriately for virus scanning to occur.	No at Site level
FND Attachment Antivirus Software Type / FND_ANTIVIRUS_SOFTWARE	Specifies the name of the anti-virus software. Currently, only Symantec anti-virus software is supported. Valid values include: • **NONE** - No anti-virus software specified • **SYMANTEC** - Semantec anti-virus software **Note:** If this profile is not set then normal uploading of the file takes place without any virus scanning.	None at Site level
FND Attachment Antivirus Server / FND_ANTIVIRUS_SERVER	Specifies the server address where the anti-virus software is running. **Note:** If the profile "FND Attachment Antivirus Software Type" is set properly but "FND Attachment Antivirus Server" is not set or has some invalid value, an error message will result and the file will not upload.	www.symantec.com at Site level

Web Server

Profile Option / Internal Code	Description	Default Value
Application Framework Agent / APPS_FRAMEWORK_AGENT	Specifies the Java listener for your HTTP server (the host and port for the web server that will be used by OA Framework applications). It can be set at the Site and the User level. Value (provide your own hostname and portname): `http://<hostname>:<portname>` **Warning:** Both the HTTP server and the Java listener should be properly configured and started before OA Framework applications can be launched.	
Apps Servlet Agent / APPS_SERVLET_AGENT	Determines the location where servlets are executed. Value (provide your own hostname and portname): `http://<hostname>:<portname>/OA_HTML` **Note:** This is a preexisting Oracle E-Business Suite profile option that must be set for graphs.	

Appendix C: OA Framework ToolBox Technical Reference Manual (TRM)

Overview

This document describes the tables and views defined for the OA Framework ToolBox Tutorial and Sample Library.

Contents

- Database Diagram
- Table Descriptions
  - FWK_TBX_PO_HEADERS
  - FWK_TBX_PO_LINES
  - FWK_TBX_PO_SHIPMENTS
  - FWK_TBX_SUPPLIERS
  - FWK_TBX_SUPPLIER_SITES
  - FWK_TBX_EMPLOYEES
  - FWK_TBX_ITEMS
  - FWK_TBX_ITEM_CCIDS
  - FWK_TBX_ADDRESSES
  - FWK_TBX_PROJECT_HEADERS
  - FWK_TBX_PROJECT_DETAILS
  - FWK_TBX_LOOKUP_TYPES_TL
  - FWK_TBX_LOOKUP_CODES_B
  - FWK_TBX_LOOKUP_CODES_TL
  - FWK_TBX_LOOKUP_CODES_VL
  - FWK_TBX_LOOKUP_TYPES_VL

Database Diagram

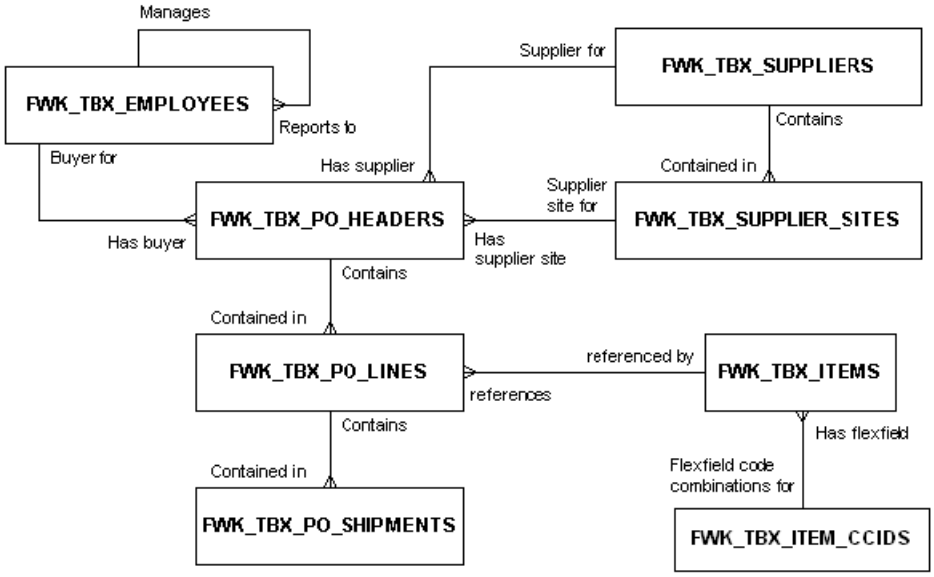


Table Descriptions

Tip: All tutorial database objects begin with FWK_TBX% if you want to describe them in SQL*Plus.

FWK_TBX_PO_HEADERS

A purchase order header Includes descriptive information about the purchase order (the buyer, the supplier, terms and conditions, and so on). A purchase order header has one or more purchase order lines.

Column	Type	Null	Description
HEADER_ID (PK)	NUMBER	NOT NULL	Header unique identifier
DESCRIPTION	VARCHAR2(240)		Order description

STATUS_CODE	VARCHAR2(30)	Order status
SUPPLIER_ID	NUMBER	Supplier
SUPPLIER_SITE_ID	NUMBER	Supplier site
CURRENCY_CODE	VARCHAR2(30)	Currency
BUYER_ID	NUMBER	Buyer
PAYMENT_TERMS_CODE	VARCHAR2(30)	Payment terms
CARRIER_CODE	VARCHAR2(30)	Carrier
SHIP_TO_ADDRESS_ID	NUMBER	Shipping address
BILL_TO_ADDRESS_ID	NUMBER	Billing address
RATE	NUMBER	Conversion rate if foreign currency PO
CONFIRM_FLAG	VARCHAR2(1)	Indicates whether the PO has been accepted by a supplier.
[DESC FLEX COLUMNS]		Standard descriptive flexfield columns
[WHO COLUMNS]		Does not include concurrent program WHO columns

FWK_TBX_PO_LINES

A purchase order line includes information about the goods and/or services being purchased (the item, unit of measure, order quantity, price, and so on). A purchase order line has one or more purchase order shipments.

Column	Type	Null	Description
LINE_ID (PK)	NUMBER	NOT NULL	Line unique identifier
HEADER_ID	NUMBER	NOT NULL	Header unique identifier
LINE_NUMBER	NUMBER	NOT NULL	Line unique identifier (user's perspective)
ITEM_ID	NUMBER		Predefined item identifier
ITEM_DESCRIPTION	VARCHAR2(240)		Item description
UNIT_OF_MEASURE	VARCHAR2(30)		Unit of measure
QUANTITY	NUMBER		Line order quantity
UNIT_PRICE	NUMBER		Unit price
[DESC FLEX COLUMNS]			Standard descriptive flexfield columns
[WHO COLUMNS]			Does not include concurrent program WHO columns

FWK_TBX_PO_SHIPMENTS

A purchase order shipment ncludes information about where and when to ship the goods/services described on the line (need-by date, shipment quantity, ship-to address, and so on).

Column	Type	Null	Description
SHIPMENT_ID (PK)	NUMBER	NOT NULL	Shipment unique identifier
LINE_ID	NUMBER	NOT NULL	Line unique identifier
SHIPMENT_NUMBER	NUMBER	NOT NULL	Shipment user unique identifier
SHIP_TO_ADDRESS_ID	NUMBER		Shipping address
NEED_BY_DATE	DATE		Date item(s) are required
ORDER_QUANTITY	NUMBER		Shipment order quantity
RECEIPT_QUANTITY	NUMBER		Shipment quantity received
RECEIPT_DATE	DATE		Shipment receipt date
PROMISE_DATE	DATE		For acknowledged purchase orders, date supplier promises to deliver.

[DESC FLEX COLUMNS]	Standard descriptive flexfield columns
[WHO COLUMNS]	Does not include concurrent program WHO columns

FWK_TBX_SUPPLIERS

Column	Type	Null	Description
SUPPLIER_ID (PK)	NUMBER	NOT NULL	Supplier unique identifier
NAME	VARCHAR2(80)	NOT NULL	Supplier name
ON_HOLD_FLAG	VARCHAR2(1)		Indicates if supplier is currently on hold
START_DATE	DATE		Supplier effective start
END_DATE	DATE		Supplier effecitve end
[DESC FLEX COLUMNS]			Standard descriptive flexfield columns
[WHO COLUMNS]			Does not include concurrent program WHO columns

FWK_TBX_SUPPLIER_SITES

Column	Type	Null	Description
SUPPLIER_ID	NUMBER	NOT NULL	Supplier unique identifier
SUPPLIER_SITE_ID (PK)	NUMBER	NOT NULL	Supplier site unique identifier
SITE_NAME	VARCHAR2(20)		Supplier site name
PAYMENT_TERMS_CODE	VARCHAR2(30)		Payment terms
CARRIER_CODE	VARCHAR2(30)		Carrier
PURCHASING_SITE_FLAG	VARCHAR2(1)		Indicates if this is a Purchasing site
ADDRESS_ID	NUMBER		Site address
START_DATE	DATE		Site start date effective
END_DATE	DATE		Site end date effective
[DESC FLEX COLUMNS]			Standard descriptive flexfield columns
[WHO COLUMNS]			Does not include concurrent program WHO columns

FWK_TBX_EMPLOYEES

Column	Type	Null	Description
EMPLOYEE_ID (PK)	NUMBER	NOT NULL	Employee unique identifier
TITLE	VARCHAR2(30)		Title (for example: Mr., Mrs.)
FIRST_NAME	VARCHAR2(20)		First name
MIDDLE_NAMES	VARCHAR2(60)		Middle names
LAST_NAME	VARCHAR2(40)		Last name
FULL_NAME	VARCHAR2(240)		Full name
EMAIL_ADDRESS	VARCHAR2(240)		Email address
MANAGER_ID	NUMBER		Manager
POSITION_CODE	VARCHAR2(30)		Position (for example: Director, Buyer)
START_DATE	DATE		Employee hire date
END_DATE	DATE		Employee termination date
SALARY	NUMBER		Salary
[DESC FLEX COLUMNS]			Standard descriptive flexfield columns
[WHO COLUMNS]			Does not include concurrent program WHO columns

FWK_TBX_ITEMS

Column	Type	Null	Description
ITEM_ID (PK)	NUMBER	NOT NULL	Item unique identifier
FWKITEM_ID	NUMBER		Key flexfield code combination
FWKITEM_STRUCTURE_ID	NUMBER		Key flexfield structure
START_DATE_ACTIVE	DATE		Obsolete column (see FWK_TBX_ITEM_CCIDS table)
END_DATE_ACTIVE	DATE		Obsolete column (see FWK_TBX_ITEM_CCIDS table)
ENABLED_FLAG	VARCHAR2(1)	NOT NULL	Obsolete column (see FWK_TBX_ITEM_CCIDS table)
SUMMARY_FLAG	VARCHAR2(1)	NOT NULL	Obsolete column (see FWK_TBX_ITEM_CCIDS table)
SEGMENT1	VARCHAR2(20)		Obsolete column (see FWK_TBX_ITEM_CCIDS table)
SEGMENT2	VARCHAR2(20)		Obsolete column (see FWK_TBX_ITEM_CCIDS table)
SEGMENT3	VARCHAR2(20)		Obsolete column (see FWK_TBX_ITEM_CCIDS table)
SEGMENT4	VARCHAR2(20)		Obsolete column (see FWK_TBX_ITEM_CCIDS table)
ITEM_DESCRIPTION	VARCHAR2(240)	NOT NULL	Item description
[DESC FLEX COLUMNS]			Standard descriptive flexfield columns
[WHO COLUMNS]			Does not include concurrent program WHO columns

FWK_TBX_ITEM_CCIDS

FWKITEM key flexfield code combinations.

Column	Type	Null	Description
FWKITEM_ID (PK)	NUMBER		Key flexfield code combination unique identifier
FWKITEM_STRUCTURE_ID	NUMBER		Key flexfield structure
START_DATE_ACTIVE	DATE		Start date
END_DATE_ACTIVE	DATE		End date
ENABLED_FLAG	VARCHAR2(1)	NOT NULL	
SUMMARY_FLAG	VARCHAR2(1)	NOT NULL	
SEGMENT1	VARCHAR2(60)		
SEGMENT2	VARCHAR2(60)		
SEGMENT3	VARCHAR2(60)		
SEGMENT4	VARCHAR2(60)		
SEGMENT5	VARCHAR2(60)		
SEGMENT6	VARCHAR2(60)		
SEGMENT7	VARCHAR2(60)		
SEGMENT8	VARCHAR2(60)		
SEGMENT9	VARCHAR2(60)		
SEGMENT10	VARCHAR2(60)		

[WHO COLUMNS]			Does not include concurrent program WHO columns

FWK_TBX_ADDRESSES

Column	Type	Null	Description
ADDRESS_ID (PK)	NUMBER	NOT NULL	Address unique identifier
ADDRESS_NAME	VARCHAR2(20)	NOT NULL	
ADDRESS_LINE_1	VARCHAR2(60)		
ADDRESS_LINE_2	VARCHAR2(60)		
ADDRESS_LINE_3	VARCHAR2(60)		
DESCRIPTION	VARCHAR2(240)		
EMAIL_ADDRESS	VARCHAR2(240)		
COUNTRY	VARCHAR2(2)		
TOWN_OR_CITY	VARCHAR2(30)		
POSTAL_CODE	VARCHAR2(30)		
START_DATE_ACTIVE	DATE		
END_DATE_ACTIVE	DATE		
TELEPHONE_NUMBER_1	VARCHAR2(60)		
TELEPHONE_NUMBER_2	VARCHAR2(60)		
TELEPHONE_NUMBER_3	VARCHAR2(60)		
[DESC FLEX COLUMNS]			Standard descriptive flexfield columns
[WHO COLUMNS]			Does not include concurrent program WHO columns

FWK_TBX_PROJECT_HEADERS

Column	Type	Null	Description
PROJECT_ID (PK)	NUMBER	NOT NULL	Project unique identifier
NAME	VARCHAR2(240)	NOT NULL	Project name
START_FROM	DATE		
END_TO	DATE		
START_DATE	DATE		
COMPLETION_DATE	DATE		
TASK_TYPE	VARCHAR2(240)		
TEXT_RIGHT	VARCHAR2(240)		
[WHO COLUMNS]			Does not include concurrent program WHO columns

FWK_TBX_PROJECT_DETAILS

A project can have many tasks.

Column	Type	Null	Description
TASK_ID (PK)	NUMBER	NOT NULL	Task unique identifier
PROJECT_ID	NUMBER	NOT NULL	Project unique identifier
TOP_TASK_ID	NUMBER	NOT NULL	Indicates if this is a subtask of another task (if TOP_TASK_ID != TASK_ID, this is a subtask).

TASK_NUMBER	VARCHAR2(30)	
TASK_NAME	VARCHAR2(240)	
START_FROM	DATE	
END_TO	DATE	
TASK_TYPE	VARCHAR2(240	
TEXT_RIGHT	VARCHAR2(240)	
[WHO COLUMNS]		Does not include concurrent program WHO columns

FWK_TBX_LOOKUP_TYPES_TL

Multilanguage table of lookup code types.

Column	Type	Null	Description
LOOKUP_TYPE	VARCHAR2(30)	NOT NULL	Lookup type unique identifier
DISPLAY_NAME	VARCHAR2(80)	NOT NULL	Displayed lookup type name
DESCRIPTION	VARCHAR2(240)		Description
LANGUAGE	VARCHAR2(4)	NOT NULL	Translation language
SOURCE_LANG	VARCHAR2(4)	NOT NULL	Creation language

FWK_TBX_LOOKUP_CODES_B

Base table for untranslated lookup code columns.

Column	Type	Null	Description
LOOKUP_TYPE	VARCHAR2(30)	NOT NULL	Lookup type
LOOKUP_CODE	VARCHAR2(30)	NOT NULL	Lookup code unique identifier
START_DATE_ACTIVE	DATE		Lookup code effective from date
END_DATE_ACTIVE	DATE		Lookup code effective to date

FWK_TBX_LOOKUP_CODES_TL

Multilanguage table for translatable lookup code columns.

Column	Type	Null	Description
LOOKUP_TYPE	VARCHAR2(30)	NOT NULL	Lookup type
LOOKUP_CODE	VARCHAR2(30)	NOT NULL	Lookup code unique identifier
MEANING	VARCHAR2(80)	NOT NULL	Displayed lookup code value
DESCRIPTION	VARCHAR2(240)		Description
LANGUAGE	VARCHAR2(4)	NOT NULL	Translation language
SOURCE_LANG	VARCHAR2(4)	NOT NULL	Creation language

FWK_TBX_LOOKUP_CODES_VL

View which presents lookup codes in the database session language.

FWK_TBX_LOOKUP_TYPES_VL

View which presents lookup types in the database session language.

Appendix D: Oracle Application Framework URL and Request Parameters

Overview

Any URL parameters that affect OA Framework application behavior are are grouped into the following categories and described below:

- AM Pool
- Events
- Look-and-Feel
- Menu
- Personalization
- Small Product Branding
- UI Features
  - Breadcrumbs
  - Tree (3-Frame JSP)

General

URL Parameter	Description	Additional Information
akRegionCode	The ID of a region, as defined in Oracle JDeveloper 10*g* OA Extension. Previously, this parameter was used along with **akRegionApplicationId** to launch a specific application page. Although these parameters have not been deprecated, their usage is being phased out and replaced by OAHP and OASF.	None available.
akRegionApplicationId	The application ID of a region. Previously, this parameter was used along with **akRegionCode** to launch a specific application page. Although these parameters have not been deprecated, their usage is being phased out and replaced by OAHP and OASF.	None available.
page	Full package name of the page to render. Previously, this parameter was used to launch a specific page. Although this parameter has not been deprecated, its usage is being phased out and replaced by OAHP and OASF.	None available.
region	Full package name of the region to render. Previously, this parameter was used to launch a specific region. Although this parameter has not been deprecated, its usage is being phased out and replaced by OAHP and OASF.	None available.

AM Pool

URL Parameter	Description	Additional Information

retainAM	Used by the Application Module (AM) pool to control application module state.	See the section titled Retaining the Application Module Across Pages, in the *OA Framework State Management* topic of *Chapter 2: OA Framework Essentials*.

Events

URL Parameter	Description	Additional Information
source	Indicates the ID or name of the web bean that raised an event. For example, events generated by a table contain the table name as the "source" parameter.	None available.
event	Indicates the type of event raised by a web bean. For example, selecting on a sortable header of a table generates an event with value "sort".	None available.
value	This is a parameter passed by certain events like table sort, where the index of the column selected for sorting is passed as the "value" parameter.	None available.
size	Indicates the number of rows currently displayed in a table (relevant only to the navigation event).	None available.
state	Indicates the current sort state (ascending or descending) of the column on which sorting is invoked (relevant only for the sort event).	None available.

Look-and-Feel

URL Parameter	Description	Additional Information

OALAF	Used to set the Look-and-Feel rendering of a single page.	See the Controlling the Look-and-Feel section, in the *Controlling UIX Rendering Output* topic of *Chapter 6: Advanced OA Framework Application Topics*.
OARF	Enables one of four HTML Look-and-Feel facets.	See the Controlling the Facet section, in the *Controlling UIX Rendering Output* topic of *Chapter 6: Advanced OA Framework Application Topics* and the Printable Page topic of *Chapter 4: Implementing Specific UI Features*.

Menu

URL Parameter	Description	Additional Information
OAHP	Only used with the OASF parameter, to establish the current menu context. This parameter points to a "Home Page" menu. Also used by the 3-Frame JSP Tree to display the menu context in the Top frame.	See the Menu Context section in the *Tabs / Navigation* topic of *Chapter 4: Implementing Specific UI Features*.
OASF	Specifies a function and tells OA Framework to select this function in the "Home Page" menu context specified by the OAHP parameter. Also used by the 3-Frame JSP Tree to display the menu context in the Top frame.	See the Menu Context section in the *Tabs / Navigation* topic of *Chapter 4: Implementing Specific UI Features*.
OAMC	Specifies whether to keep or remove the current menu context.	See the Setting the Menu Context section in the *Tabs / Navigation* topic of *Chapter 4: Implementing Specific UI Features*.

Personalization

URL Parameter	Description	Additional Information
OAFunc	Used to pass a function name to the URL. This parameter may be used in: • Personalizations - to pass the function name of a personalized region to the URL. • JTT/OA Framework Interoperability - To reach a JTT page by its function name from OA Framework. • 3-Frame JSP Tree - to pass the names of functions as the source for each frame.	See the Function-Level Personalizations section in the *Administrator-Level Personalizations* topic of the *Oracle Application Framework Personalization Guide*. See the JTT/OA Framework Interoperability topic in *Chapter 6: Advanced OA Framework Development Topics*. See the Defining a 3-Frame JSP Page for Use with the Tree Component section in the *Tree* topic of *Chapter 4: Implementing Specific UI Features*.

Small Product Branding

URL Parameter	Description	Additional Information
OAPB	Specifies the function name that represents your product branding text.	See the Basic (Non-Contextual) Branding section in the *Branding* topic of *Chapter 4: Implementing Specific UI Features* and the Branding section in the *Personalizing Your System* topic of *Chapter 2: Personalizing OA Framework Applications* in the *OA Framework Personalization Guide*.

Breadcrumbs

URL Parameter	Description	Additional Information

addBreadCrumb	Used by Breadcrumbs to control the breadcrumb behavior.	See the Locator Element: Breadcrumbs topic in *Chapter 4: Implementing Specific UI Features*.

Tree (3-Frame JSP)

URL Parameter	Description	Additional Information
OAFRDIM	Used by the 3-Frame JSP Tree to override the default Start frame and Top frame width and height.	See the Declarative Implementation of the *Defining a 3-Frame JSP Page for Use with the Tree Component* section in the *Tree* topic of *Chapter 4: Implementing Specific UI Features*.
OAHP	Only used with the OASF parameter, to establish the current menu context. This parameter points to a "Home Page" menu. Also used by the 3-Frame JSP Tree to display the menu context in the Top frame.	See the Menu Context section in the *Tabs / Navigation* topic of *Chapter 4: Implementing Specific UI Features.*
OASF	Specifies a function and tells OA Framework to select this function in the "Home Page" menu context specified by the OAHP parameter. Also used by the 3-Frame JSP Tree to display the menu context in the Top frame.	See the Menu Context section in the *Tabs / Navigation* topic of *Chapter 4: Implementing Specific UI Features*.
OAPLRS	Used by the 3-Frame JSP Tree to suppress the page header information in the Center frame.	See the Declarative Implementation of the *Defining a 3-Frame JSP Page for Use with the Tree Component* section in the *Tree* topic of *Chapter 4: Implementing Specific UI Features*.

Appendix E: OA Framework Development Frequently Asked Questions (FAQ)

Overview

For the most up-to-date list of common questions and solutions related to development using OA Framework, refer to My Oracle Support Knowledge Document 1455931.1, *Oracle Application Framework Development FAQ*.

For common problems and solutions about installation and configuration of OA Framework, and deployment of OA Framework-based applications, refer to My Oracle Support Knowledge Document 1512113.1, *Oracle Application Framework Installation Troubleshooting*.

Appendix F: OA Framework Known Issues

Contents

This section is organized into the following categories:

- UI Features
  - Attachments
  - Data Export/Import
  - Declarative Page Flow
  - Flexfields
  - File Upload/Download
  - HGrid
  - LOV
  - Locater Element: Breadcrumbs
  - Locator Element: Train
  - Mobile Applications
  - Panel Splitter
  - Page Layout
  - Partial Page Rendering (Dynamic User Interface)
  - Printable Page
  - Processing Page
  - Rating Bar
  - Record History
  - Rich Table Interactions
  - Rich Text Editor
  - Save Model ('Warn About Changes')
  - Search
  - Tables (Classic and Advanced)
  - Tree
- Other/General

Warning: This document does not list all known bugs and enhancements for the OA Framework. Instead, it lists selected, key issues that we hope will facilitate your design and development efforts.

Additional Information: Refer to the *Recommended Browsers for Oracle E-Business Suite Release 12*, My Oracle Support (formerly Oracle MetaLink) Document 389422.1, for any browser-specific known issues.

UI Features

Attachments

- Any attachment added successfully without a specified title behaves as follows:
  - Displays "Undefined" as its default title in the base Attachments page because the link to view the attachment is based on the title text.
  - Displays no title in the inline View Attachments pop-up as there this no link to view the attachment..
- Mouse users can drag and drop an Inline Attachment pop-up, but keyboard-only and mobile device users can not do so.

Data Export/Import

- If you want to export data from a switcher in a table, advanced table or HGrid, the case name and ID of the case's child item must be identical. Generally, this is not an issue if the switcher region is defined declaratively using OA Extension, as the case name automatically defaults to the ID of a case's child item.

Declarative Page Flow

- Parallel flows are not supported when using Workflow page flows.

Flexfields

Descriptive Flexfields

- Descriptive flexfield context can be lost after a user selects a context value. The issue has been reported in various bugs, including 25180539 and 22841421. Basically, the behavior is that after a full page refresh, the context value the user has selected is lost. The issue is the result of product code doing an unconditional VO query in the controller processRequest phase.
  For example, here is some sample code from a product controller:

```
OAViewObject vo =
(OAViewObject)pageContext.getApplicationModule(webBean).findViewObject("BankAM
.BankInfoPLSQLVO");
vo.invokeMethod("initQuery", params);
```

  This code needs to be amended by the addition of one of the following checks before calling initQuery:

```
if(!vo.isPreparedForExecution())
```

  or

```
if(!vo.isExecuted())
```

- If more than one descriptive flexfield is in the same table, OA Framework only renders one context switcher on the table action area. UI team needs to finalize how to render this situation.
- In the Oracle E-Business Suite Forms-based environment, a descriptive flexfield can default its context value from a reference field. To accomplish this in Oracle Application Framework-based pages, you can implement a Reference Field or you can do the following:

  1. Using Personalization, create a new region prior to the region containing the descriptive flexfield.
  2. Attach a controller to this region.
  3. In the `processRequest()` method of the controller, add code similar to the examples below depending on your scenario.

     Note: If the reference field source is another field in the page (which is not under a table or HGrid), then you can get its value as follows:

```
// Get the root bean and the DFF bean.
OAWebBean rootBean = pageContext.getRootWebBean();
OADescriptiveFlexBean dff = (OADescriptiveFlexBean)rootBean.findChildRecursive("DFF bean name");


// Find the source bean that defaults the context field.
OAWebBean sourceBean = rootBean.findIndexedChildRecursive("legal_structure");
String value = null;
if (bean != null && bean instanceof OAWebBeanFormElement){
// Get the value from the bean
Object objValue = ((OAWebBeanFormElement)bean).getValue(pageContext);
if ( objValue != null) value = objValue.toString();
} dff.setFlexContext(pageContext, profileValue); }
```

Note: Here, the assumption is that the source bean is a Form Bean. If it is otherwise, say, a MessageStyledText web bean, use the appropriate API to get the value.

Note: If you want to use the above code in processFormRequest, you may have to refer to the processRequest Method section for Setting Up a Descriptive Flexfield in the Flexfields topic. If the reference field source is a profile option, then you can get the value directly using `pageContext.getProfile(PROFILE_CODE)`.

```
// Get the root bean and the DFF bean
OAWebBean rootBean = pageContext.getRootWebBean();
OADescriptiveFlexBean dff = (OADescriptiveFlexBean)rootBean.findChildRecursive("DFF bean name");
// Get the Profile value.
String profileValue = (String) pageContext.getProfile("Profile Name");
dff.setFlexContext(pageContext,profileValue);
```

For more information see the Flexfields chapter.

Key Flexfields

- For a table that contains a key flexfield, where the code combination LOV UI is turned off, that is the Java API `OAKeyFlexBean.usecodeCombinationLOV` is set to false, the individual columns belonging to the key flexfield do not get exported when selecting the Export button.
- When a segment is hidden for a key flexfield, and you perform a search in the **Search and Select: Key Flexfield** window to select pre-existing codes, the results of the search will include all matching codes, but the codes will only show displayed segments. Any hidden segment values in those codes are not shown. As a result, if the database contains key flexfield code values where only a hidden segment differs between multiple codes, then the key flexfield search results will display all those matching codes and they will appear as duplicate values because the hidden segment value that differs between those codes is not visible.

Reference Fields

- The specified reference field should exist in the current OA Framework page.
- A standard flex event with the name "FlexReferenceUpdate" event should be associated with the reference bean specified. If the reference bean is already associated with an action, an error is thrown indicating the bean specified cannot act as a reference field.
- When a flex item is in update mode and the value derived from the view object row associated with this flexfield is different from the value supplied in the reference web bean, the following message displays as a Developer Mode exception:

  "The value in flexfield context reference web bean does not match with the value in the context of the Descriptive flexfield web bean PermitFlex. If this is not intended, please go back to correct the data or contact your Systems Administrator for assistance."

- If there are multiple reference fields for the same property of a field, the order in which the field is taken into consideration is uncertain.
- A reference field cannot be under a Table or Advanced Table or H-Grid.

File Upload/Download

- When the upload bean is under a PPR region, the first time enctype is not added to form bean. To resolve this, add the following code to the controller of the page process request:
  `((OAFormBean)pageContext.findFormBean(pageContext.getFormName(webBean))).setUsesUpload(true);`
- The Export operation is disabled when Oracle E-Business Suite is run on the Apple iPad.

HGrid

- HGrid automatically fetches all the records of a tree node, when the tree node is first displayed. The intent is to fetch only the number of records specified by the Record Set Size property on the hGrid or nodeDefinition and use record navigation to eventually fetch all records.

LOV

- Bug 5033415 - LOVs will not function properly in Microsoft Internet Explorer in HTTPS mode if a Proxy server being used is not compatible with HTTP 1.1. To work around this issue, modify Internet Explorer's HTTP 1.1 settings as follows. Navigate to **Internet Options** > **Advanced** > **HTTP 1.1 settings.** Uncheck the options **Use HTTP 1.1** and **Use HTTP 1.1 through proxy connections**.
- Browser plugins such as Yahoo! Companion can interfere with the LOV modal window.
- Results fields for LOVs should not be added inside a hide/show region. Doing so can lead to results fields not being populated when the hide/show region is in a closed state.
- If a user enters a value in a LOV field and selects the Submit button on the page without tabbing out of the LOV field first, validate-on-submit occurs, along with the following, depending on the value entered in the LOV field:
  - If there is exactly one match to that entered value, the LOV field is populated with that value.
  - If there is more than one match to that value, the error "**Error - Select a valid value"** results.

  In this scenario, the `isLovEvent ()` event is not triggered hence the code that should be executed as a result of this event does not run. To work around this issue, identify all other cases when a piece of logic needs to be executed and include their required conditions in the same condition statement that also checks for `pageContext.isLovEvent ()`.
- For the Look Ahead LOV feature, in Internet Explorer 6 (or less), the `Look Ahead Records Displayed` property is not honored if it is set greater than 10, set either via the OA Extension property inspector or the `setLookAheadSize(String maxSize)` method on `OAMessageLovInputBean`. It will always be 10 as the maximum.
- If a Look Ahead LOV appears over a **messageChoice** item (also known as a poplist), on the page, the messageChoice data will appear over the Look Ahead LOV data. This is a known issue specific to Internet Explorer version 6.
- The Look Ahead LOV is Disabled in Screen Reader Accessibility mode.
- A Look Ahead LOV on number fields leads to full table scans.
- In Internet Explorer version 6, there is a problem with scroll bar event handling in Look Ahead LOVs.

- In Internet Explorer, using the [Space bar] to select a highlighted record does not work.
- In Microsoft Internet Explorer (IE) 7.0 and above, if you enable the option "Always open pop-ups in a new tab" when a pop-up is encountered (under "Tools" > "Internet Options" > "General" > "Tabs" > "Settings"), undesirable behavior results when you select an LOV icon on a base page. Selecting the LOV icon opens a new tab in the same browser window, but the tab is disabled. This issue can not be fixed as it is inherent to the browser and not to OA Framework. To work around this issue in IE, you can do one of the following:
  - Right-click or use Ctrl+Tab to switch to and activate the LOV screen. This will also disable the base page.
  - In IE, rather than select the option "Always open pop-ups in a new tab" when a pop-up is encountered (under "Tools" > "Internet Options" > "General" > "Tabs" > "Settings"), select the option "Let Internet Explorer decide how pop-ups should open".

  If you encounter this issue in Firefox, then enable the option "When I open a link in a new tab, switch to it immediately" (under "Tools" > "Options" > "Tabs").
- For the Look Ahead LOV feature, if a search returns multiple rows, a LOV window launches only when the LOV region item's attribute does not match the LOV result item's attribute.
- Within the same application module (AM) and transaction, every LOV bean should have a unique node name. If two or more LOVs have the same node name, the LOV mappings of them should also be the same, or there may be issues such as no validation on submit.

Locater Element: Breadcrumbs

- When the user navigates by selecting a breadcrumb link, the page state may be lost (Bug 2431147). The reason is that the root application module retention behavior is determined by the `retainAM` URL parameter value of the breadcrumb URL.

  Note: Do *not* change all the breadcrumb URL values to have `retainAM=Y` to work around the problem. This could cause serious scalability issues by tying up application module resources and increasing the memory load. If this is a serious problem for you, consider using the `oracle.apps.fnd.framework.webui.OAReleaseListener` to ensure that individual application modules are retained or released as needed.

- As tracked in Bug 1728163, breadcrumbs exhibit the following problem when using the browser Back button:

  User navigates from Page 1 (Tab 1 is highlighted) to Page 2, and then to Page 3. The breadcrumbs show "Tab 1 > Page 2" in Page 3. The user presses the browser Back button to return to Page 2, selects a link to go to Page 4, and then navigates to Page 5. The desired breadcrumb list in Page 5 is "Tab 1 > Page 4", but the displayed breadcrumb list is "Tab 1 > Page 2 > Page 4" because the web server is not aware of the client-side navigation event to Page 2.

- The current OA Framework breadcrumb implementation and the latest breadcrumbs UI standards have some discrepancies. The new UI standards move in the direction of a predefined, fixed page hierarchy for the breadcrumbs while OA Framework breadcrumb implementation tends to model the dynamic navigation path. The menu breadcrumbs also behave differently from the UI standards.

  Note: Due to backward-compatibility concerns, OA Framework does not plan to add or revise breadcrumbs behavior unless the change has a localized impact.

Locator Element: Train

- The interactive train does not work with page flows that are implemented with Oracle Workflow.
- If a Train is associated with a Navigation Bar and the RENDERED property of a step in the Train is set to false, it is the responsibility of the developer to explicitly ensure the RENDERED property for that step in the associated Navigation Bar is also set to false. A mismatch between the Train and the Navigation Bar is created if RENDERED properties are not synchronized. For example, if the second step of a three-step Train is not rendered, the Train will display only two steps (Step 1 and Step 3), and the Navigation Bar will display all three steps.

Mobile Applications

- **Scrollbar** - The Apple Safari browser does not render a visible scrollbar for iFrames and Text Areas even though you can scroll these areas using your fingers. The result is that there is no visible indication that more data exists and can be viewed by scrolling down or up. Affected OA Framework components include:
  - Home page
  - Rich Text Area
  - Pop-up
  - Look Ahead LOV
  - Table

  Note: Starting in Release 12.2.5, the Table and Popup components show a visible scroll bar once you tap within the component and start to scroll.

- **Mouse Hover** - Since there is no equivalent gesture for a mouse hover on touch devices, OA Framework components that rely on the mouse hover event do not work. This affects the OA Framework Graph component.
- **File Dialog** - Whenever the File Upload or file-type Attachment components are used, the **Choose File** button rendered by the Safari browser on iOS allows access only to whatever attachments the platform supports for Safari. For example, iOS currently allows access to Photo gallery and camera, but not PDF files via the Browse button dialog.
- **iFrame** - The Safari browser does not honor the `Height` and `Width` properties of the iframe tag and as a result, the iFrame occupies the complete width of the parent region causing misalignment of the page content. Affected OA Framework components include:
  - Parameterized Pop-up
  - Rich Content Container
- **Export** - Data export to Microsoft Excel is not supported on tablets.

Panel Splitter

- In Microsoft Internet Explorer (IE) 11, the panel splitter does not render a vertical scroll bar even when the panel content exceeds the panel height.
- Any page with a panel splitter component always displays a browser-level vertical scroll bar even if the content fits within the browser height.
- The panel splitter state does not persist if a panel contains a message rich text editor (MRTE) component and the user selects **Change to Plain Text Mode**.

Page Layout

Message Component Layout

- Per the UI team, tabbing order is more important than row/column span capabilities in a **messageComponentLayout** region. Therefore, the **messageComponentLayout** does not support row/column spanning.
- If your regions include buttons below all the fields, and they are personalizable, there is currently no way to ensure that the buttons continue to render last (in the first column of the last row). Use a defaultSingleColumn region even though it is deprecated.

Partial Page Rendering (Dynamic User Interface)

- Bug 7491681- Tabbing out of a PPR enabled field in Mozilla takes the focus to next focusable item in spite of setting profile 'FND: Disable PPR Auto Tabbing' to Y.
- If a page has a subtab layout where any one of the subtabs contains a defaultSingleColumn or defaultDoubleColumn region whose Rendered property gets changed during PPR, the page hangs, preventing the user from navigating to other subtabs. To avoid this problem, use a header region rather than a defaultSingleColumn or defaultDoubleColumn region in the subtab.

Printable Page

- Printable pages do not support the Rich Text Editor.

Processing Page

- For Release 12 Back button support, none of the Global buttons or links, except Home/Return to Portal, Logout and Close Window display in the "Processing" page.

Rating Bar

- The rating image must currently be the default star image.

Record History

- Record history is not supported for Classic or Advanced tables with a nested AM.

Rich Table Interactions

- **Browser Certification** - Refer to the *Recommended Browsers for Oracle E-Business Suite Release 12*, My Oracle Support (formerly Oracle MetaLink) Document 389422.1, for the list of browsers that support the Rich Table features.
- **Column Reordering** - If you enable Totaling on a column in a classic or advanced table, you may only reorder the columns to the left of the Totaled column amongst the columns on the left and reorder the columns to the right of the Totaled column amongst the columns on the right. You may not reorder the columns on the left of the Totaled column to the right of the Totaled column or reorder the columns on the right of the Totaled column to the left of the Totaled column.
- **Column Resizing** - If a classic or advanced table is present inside a cellFormat region and the cellFormat is right-aligned, then when you resize the columns of the table, the table will fill the remaining space in the browser towards the left and expand beyond the browser's horizontal space with a scroll bar. If you minimize the browser window using the minimize icon at the top right corner of the browser window, the table will appear towards the end of the page with a scroll bar. If you refresh the page again, this behavior no longer appears. A workaround for this issue is to align the cellFormat web bean towards the left. If this is not possible, then you may wish to disable all the rich interactions on these tables by programmatically setting the following:

```
tableBean.setColumnResizeEnabled(false);
tableBean.setColumnReorderEnabled(false);
tableBean.setHorizontalScrollEnabled(false);
tableBean.setDetachEnabled(false);
```

- **See known issues for Mobile Applications.**

Rich Text Editor

- Rich Text Editor is not supported on a Printable page.
- Bug 3982588 - When displaying a page containing a Rich Text Editor region, Microsoft Internet Explorer may omit some HTML closing tags (such as </LI>) on the page when you leave the page, even though these tags are restored and saved in the database. As a result of this mismatch, the next time you reopen the page, a Save warning window may appear when you leave the page by choosing a submit button.
- The font toolbar setting does not always reflect the correct format of the selected text.

  Example: A user invokes a page that contains a MRTE. The user selects the value Times New Roman from the font poplist and enters some text. When they begin a new paragraph they change the font value to Ariel. If the user then returns to the first paragraph and adds some additional text, the correct Times New Roman font is used but the toolbar font setting doesn't change. It still displays Ariel.

- In creating a number order list within the rich text content, introducing a line space between two list items causes the numbering to reset. For example, suppose you have the following numbered list:

```
1. Line A
2. Line B
```

  Introducing a line space between Line A and Line B will reset the numbering as follows:

```
1. Line A
1. Line B
```

- On Safari and Mozilla Firefox 3.5 and above, the Cut, Copy and Paste icons do not render due to security restrictions imposed by Safari and Firefox 3.5 and above.

Save Model ('Warn About Changes')

- In order to provide save model support for Javascript links, the OA Framework adds some special Javascript on the `onClick` event. The OA Framework does not currently support chaining Javascript functions on the `onClick` event.

Search

- When implementing the autoCustomizationCriteria mode for a query region, the custom simple and advanced search regions must reside in the same XML as the query and table regions. This restriction is required because the search regions mappings don't use a fully qualified ID; the IDs are assumed to be valid only within the current document.
- If you use `setQuery()` on a view object associated with the query bean results region, then you should also call `vo.setFullSqlMode(FULLSQL_MODE_AUGMENTATION)` on the view object. This ensures that the order by or the WHERE clause, that is generated by OA Framework, can be correctly appended to your VO. The behavior with `FULLSQL_MODE_AUGMENTATION` is as follows:
  1. The new query that you have programmatically set takes effect when you call `setQuery` and execute the query.
  2. If you call `setWhereClause` or if a customer personalizes the criteria for your region, BC4J augments the `whereClause` on the programmatic query that you set. For example:

     ```
     select * from (your programmatic query set through setQuery)
     where (your programmatic where clause set through setWhereClause)
     order by (your programmatic order set through setOrderBy)
     ```

     The same query is changed as follows if a customer adds a criteria using personalization:

     ```
     select * from (your programmatic query set through setQuery)
     where (your programmatic where clause set through setWhereClause) AND (additional personalization where clause)
     order by (your programmatic order set through setOrderBy)
     ```

Warning: If you do not set `FULLSQL_MODE_AUGMENTATION`, the `whereClause` and/or the `orderBy`, which was set programmatically, will not augment on the new query that you set using `setQuery`. It will instead augment on your design time VO query.

- Due to timing issues, always use the controller on the query region for any `OAQueryBean` specific methods.
- When using query regions in a multi-page flow, the query regions *must* have unique IDs. For example, query region IDs for the EmpSearchPG and DeptSearchPG can be EmpQueryRN and DeptQueryRN respectively.
- If a Query Bean configuration contains multiple tables/advanced tables as its children, rich interactions are supported only in the first table. As a result we recommend you disable rich interactions altogether in this scenario.

Tables (Classic and Advanced)

- Bug 4543106 - (BLAF : NUMERIC DATA ALIGNMENT ISSUE) Currently, when the data alignment of a messageLovInput item in a table column is set to TEXT_FORMAT, the client-side numeric validations may fail after a form submit.
- Inner tables (Table-in-Table) do not support List of Values (LOV) controls.
- To support accessibility for screen readers, you must also specify column headers for your classic or advanced table. if you wish to implement a table without column headers, design it as a tableLayout region. A table without column headers is not accessible and as a result, will lead to an error that mistakenly identifies the table as not a data table.

Tree

Used in 3-Frame JSP Page

- Window title - there is no way to specify the window title for a 3-Frame JSP page.
- 'Return to' link - there is no way to specify a 'Return to' link in a 3-Frame JSP page.
- Sync-up between top and center frame - There is no easy way to synchronize the top frame and the center frame. For example, you can drill down to other pages from the center/content frame, but there is no easy way to handle this. The developer needs to render the entire page (just not the frame) to do this.
- Unspecified size for the frames - Currently, OA Framework hard codes the size of each frame. This is subject to change based on the feed back from the developers.

Other/General

JTT/OA Framework Interoperability

Launching an OA Framework Region from a JTT Application Menu

- If your region has a pageButtonBar, the content of this region will be rendered twice. As a result, if you have a controller associated with the pageButtonBar region, the `processFormRequest` method in the controller will be executed twice on the same request. As a work around, you may try the following:
  1. Instead of assigning the controller to the pageButtonBar region, merge the controller into the controller of the pageLayout region, or call the pageButtonBar controller's method from the controller of the pageLayout region, as if the two controllers have been merged.
  2. Use a parameter to check if the `processFormRequest` method has been executed. For example:

```
public void processFormRequest(OAPageContext pageContext,
        OAWebBean webBean)
  {
   // If this method is executed once already for the same
   // page context, return now.
   if (pageContext.getParameterObject("_x/y/z/pfr_executed") != null)
        return;
   try
     {
        // your processFormRequest code
     }
   finally
     {
      // Set it in the finally clause to make sure
      // it's set even when an exception is thrown
      pageContext.putParameter("_x/y/z/pfr_executed", "yes");
     }
  }
```

  The `getParameterObject` method returns only the parameter object set by the `putParameter` method. You can use the method `getParameter` only if you are sure the parameter name you choose does not match any other existing parameter names from the request.

- If you want to use **OAFunc** or the `pageContext.forwardImmediately` method, you have one of two options:
  - Use your application JSP name, for example, **jtfcrmchrome.jsp** in the HTML call of the destination function. OR
  - Change the function type of the destination function to **INTEROPJSP**. This function type informs the Oracle E-Business Suite to replace **OA.jsp** with **jtfcrmchrome.jsp** in the destination function. This replacement takes place only when the region is launched from the JTT menu.

Launching a JTT Page from an OA Framework Application

- Pages that use **jtfsrfp.jsp** are not supported. This is normally not a problem because almost all JTT framework pages use **jtfsrnfp.jsp**.
- Pages that use **jtfdnbar.jsp** are not supported because **jtfdnbar.jsp** is obsolete due to lack of BLAF support. Executing **jtfdnbar.jsp** in OA Framework applications throws a runtime exception. Please use **jtfdnbartop*.jsp** and **jtfdnbarbtm.jsp** instead.
- Note that omitting the "ACTION" attribute of the HTML "FORM" tag may cause problems. If problems occur, they will occur in JTT applications as well.
- Navigation callback settings have no effect on OA Framework pages. For example, a JTT page may include a navigation callback JavaScript function that disallows a user from leaving the page until a certain text field is filled. When this JTT page runs in an OA Framework application, users will be able to leave this page freely when the destination is an OA Framework page because the callback JavaScript function will not be invoked when the destination is an OA Framework page.

Hiding Add to Favorites Icon

A new API was introduced in OA Framework Release 12.2.7 to hide the "Add to Favorites" icon on an OA Framework page. The "Add to Favorites" icon may appear in some product pages that are not truly bookmarkable pages.

Developers may use the following example code in a page controller to hide the "Add to Favorites" icon in such pages:

```
OAPageLayoutBean  plb = pageContext.getPageLayoutBean();
```

```
plb.setBookmarkEnabled(false);
```

Appendix G: Oracle Application Framework Extensible Regions

Overview

Some parts of your page can be inherited by extending other regions. Any extensible regions that affect OA Framework application behavior are described below:

Extensible Regions

Extensible Region	Description	Additional Information
OAReqFieldDescRG	Required Field Indicator. For example, extend `/oracle/apps/fnd/framework/webui/OAReqFieldDescRG` region to get a Required Key region. Refer to the Extends Exercise in the Oracle Application Framework Toolbox Tutorial for more information.	Use this region whenever there is at least one required field on the page. Add this region under the `pageStatus` named child of the root `pageLayout` web bean.

Appendix H: Oracle JDeveloper Technical Reference Paper

Oracle Application Framework: JDeveloper 9.0.3 IDE vs. JDeveloper 10.1.3 IDE

Oracle JDeveloper, together with OA Extension (an Oracle E-Business Suite design-time extension) makes up the integrated development environment (IDE) for OA Framework. In OA Framework Release 11.5.10, the IDE is based on Oracle 9*i* JDeveloper (9.0.3). In OA Framework Release 12, the IDE is based on Oracle JDeveloper 10*g* (10.1.3).

Oracle JDeveloper 10g provides many new features for developer productivity, user interface and standards support. These features are outlined in What's New in Oracle JDeveloper 10g Release 3 (10.1.3). Features that are of interest to Oracle E-Business Suite developers are listed under the Core IDE section. As an Oracle E-Business Suite developer, you should continue to follow the development guidelines and OA Extension implementation details provided in the OA Framework Developer's Guide when using the 10.1.3 IDE.

While there may be many subtle differences between the 9.0.3 and 10.1.3 design-time environments, this technical reference paper focuses only on those major differences that are relevant to an Oracle E-Business Suite developer.

Contents

This technical reference paper contains the following sections:

1. Maintaining Projects in 9.0.3 vs. 10.1.3
    a. Categories
    b. Dynamic Projects
    c. Project Properties
    d. Working Sets
2. Deploying Files

Maintaining Projects in 9.0.3 vs. 10.1.3

The paradigm for maintaining a project in Oracle 9*i* JDeveloper differs from that used in Oracle JDeveloper 10g. In 9.0.3, the **System - Navigator** is the default panel that you use to view and maintain your project files. In 10.1.3, although the **System - Navigator** is still available, a new **Applications Navigator** is the now the default panel. The System and Applications navigators offer different views of your projects and have different features associated with them, as described in the sections below.

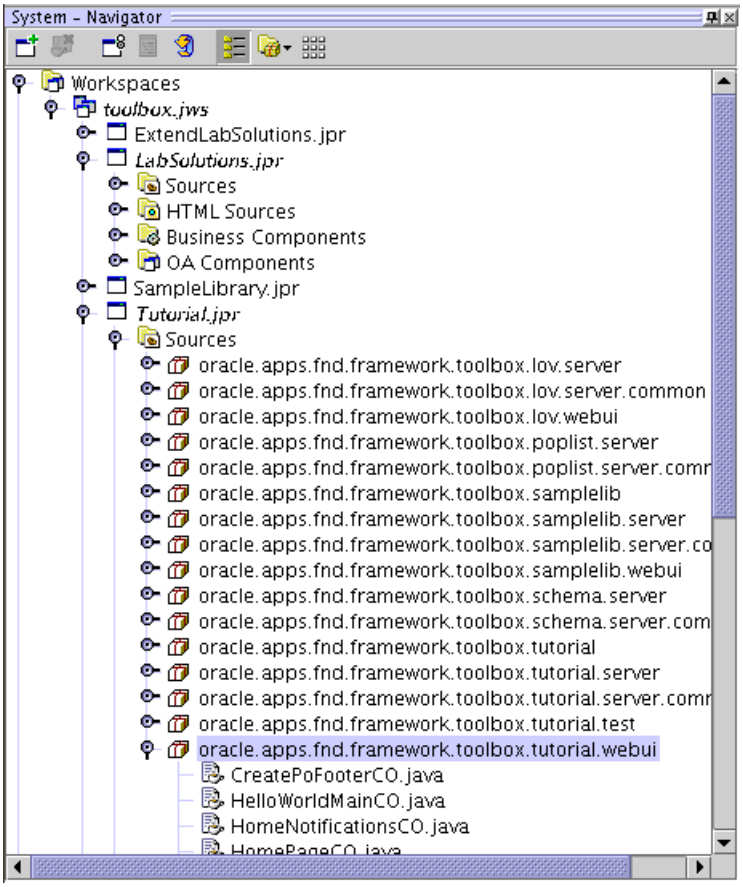
Categories

One notable difference between Oracle 9*i* JDeveloper and Oracle JDeveloper 10g is that Oracle JDeveloper 10g does not let you display a project's files by categories.

9.0.3 Behavior

When you first open a project in Oracle 9*i* JDeveloper, the project's components are displayed alphabetically as a flat list in the **System - Navigator** panel. Most Oracle E-Business Suite developers then select the project and choose the **Project** > **Show Categories** menu option to display the project files by categories. These categories (**Sources**, **HTML Sources**, **Business Components** and **OA Components**) organize the files in a logical context for developers to use and locate, as shown in Figure 1.

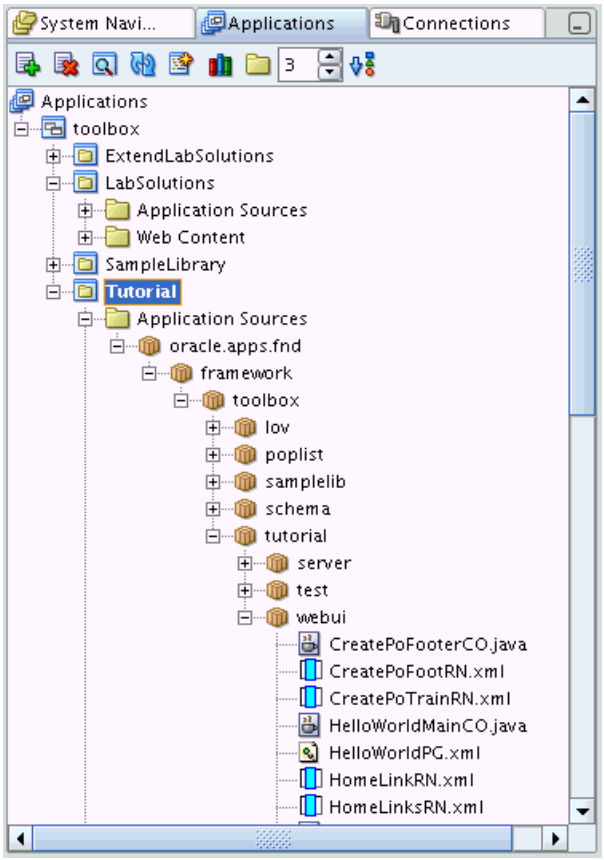
Figure 1: Oracle JDeveloper 9*i* System Navigator, displaying theToolbox Tutorial project contents by category



10.1.3 Behavior

In Oracle JDeveloper 10g, the **Applications Navigator** is the default panel rather than the **System Navigator**, as shown in Figure 2.

Figure 2: Oracle JDeveloper 10g Applications Navigator, displaying theToolbox Tutorial project contents



In 10.1.3, the **System Navigator** represents your projects in terms of files and directories, whereas the **Applications Navigator** represents your projects in terms of higher-level logical components. When you open your workspace and project in the **Applications Navigator**, the project components are organized within two folders:

- **Application Sources** - contains all your application source files in a flattened list. This includes your BC4J, Java and XML files.
- **Web Content** - contains your JSP files along with anything in that belongs in OA_HTML.

Select the **Sort by Type** icon in the navigator toolbar to sort the list of files in these folders alphabetically or by type.

Note:

As an Oracle E-Business Suite developer using 10.1.3, we recommend you use the **Applications Navigator** as it provides better project abstractions than the **System Navigator**. Although 10.1.3 does not display your project files by categories, you can configure your view of the project in the **Applications Navigator** by including or excluding content from specific subfolders. See the Project Properties discussion below for more information.

Dynamic Projects

Oracle JDeveloper 10g supports a new concept called dynamic projects.

9.0.3 Behavior

The projects supported in Oracle 9*i* JDeveloper are *static* projects in that you select the files you want to include in your project.

10.1.3 Behavior

Oracle JDeveloper 10g supports *dynamic* projects, that is, once you specify the source paths (for your Java and HTML files), *everything* under those paths are included in your project. This means that by default, all the CSS and JSP files in your HTML source are now exposed in the **Web Content** folder and all the Java files in your Java source are now exposed in the **Application Sources** folder of the **Applications Navigator**.

For an Oracle E-Business Suite developer, this exposes more artifacts than you may want to see. However, 10.1.3 does allow you to configure how the project is displayed in the Applications Navigator. See the Project Properties discussion for more information.

To help you work more efficiently with dynamic projects, 10.1.3 also provides the following navigator toolbar icons:

-  **Level** - this spinbox in the navigator toolbar lets you specify the package hierarchy node level used to display the source content. For example, if you set **Level** to **3** and then open the **Tutorial** project from **toolbox.jws**, you would see **oracle.apps.fnd** as the initial level under the **Application Sources** folder . If you set **Level** to **4**, you would see **oracle.apps.fnd.framework** as the beginning level.
-  **Toggle Libraries** - this icon lets you show or hide all the libraries attached to a particular project.
-  **New View** - this icon lets you open a new **Applications Navigator** panel that displays only the selected project as its main node.
-  **Toggle Directories** - this icon lets you view Web Content files and directories as they are displayed in their physical location under OA_HTML or as a virtual mapping.

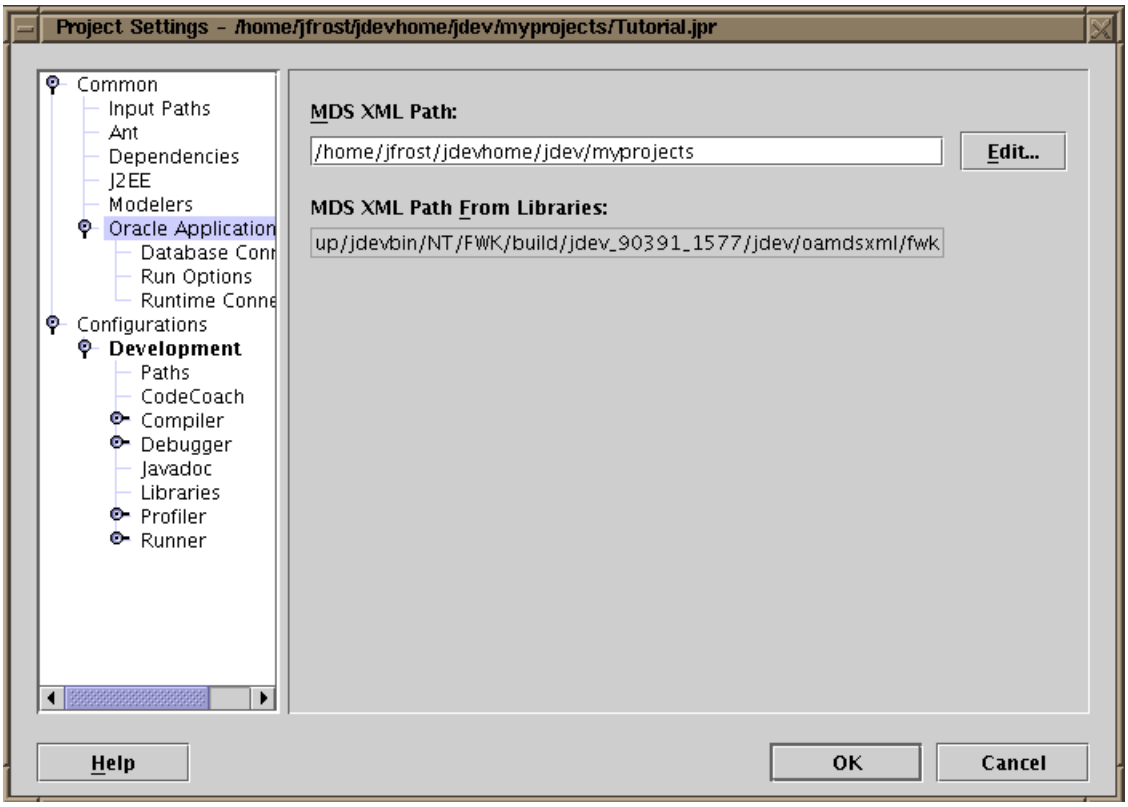
For more details about these features, refer to the Oracle JDeveloper 10g online help.

Project Properties

9.0.3 Behavior

In 9.0.3, to view or modify a project's settings, you select a project, then choose the **Project** > **Project Settings...** menu option.

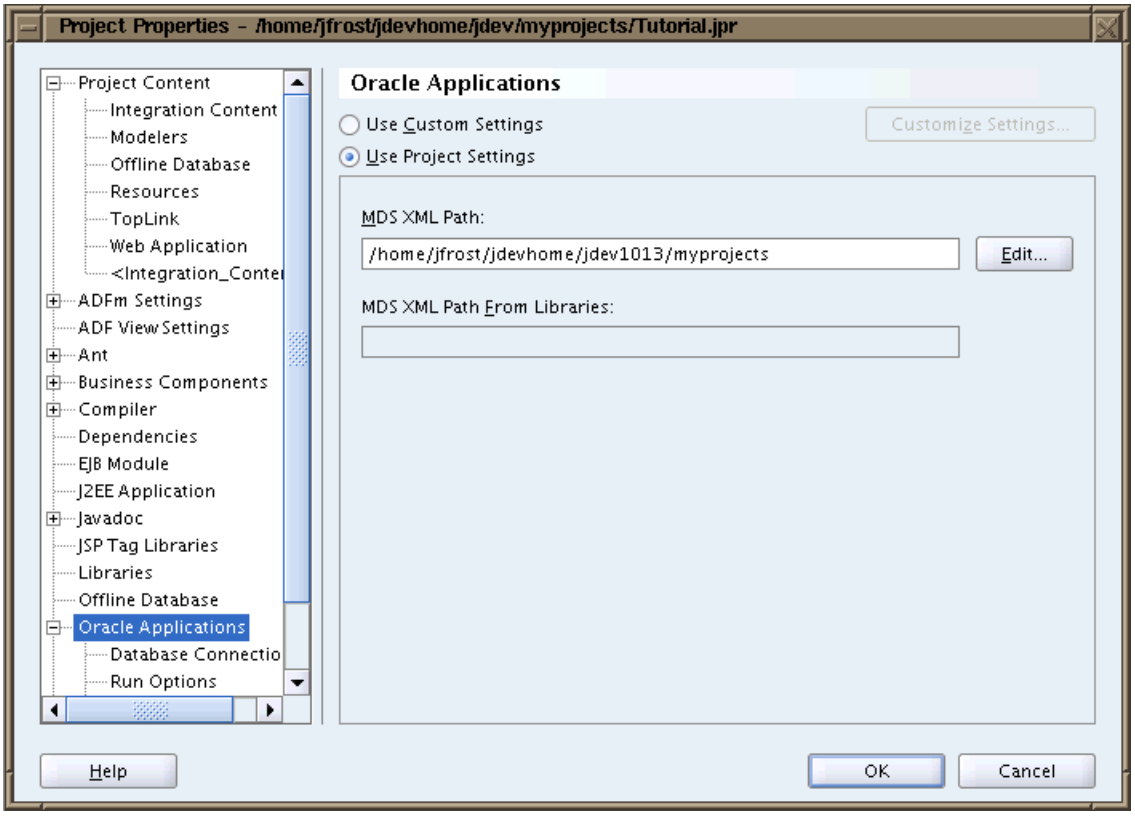
Figure 3: Project Settings for JDeveloper 9*i*



10.1.3 Behavior

In 10.1.3, Project Settings are now called Project Properties, and can be accessed by selecting the project, then selecting the **Project Properties...** icon on the navigator toolbar or by double-clicking on the project in the navigator. While there are differences between the 9.0.3 settings and the 10.1.3 properties, the properties/settings under the **Oracle E-Business Suite** node (**Database Connection**, **Run Options**, and **Runtime Connection**) are still the same, as shown in Figure 3 and Figure 4, respectively.

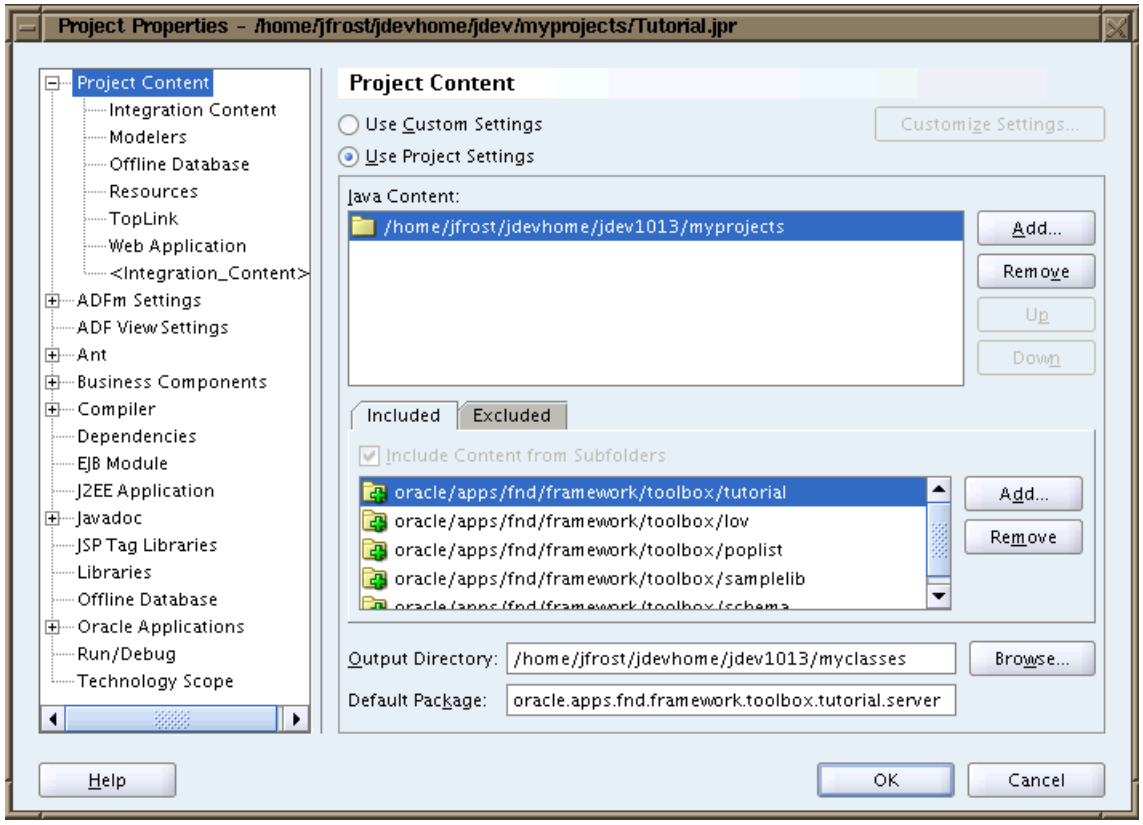
Figure 4: Project Properties for JDeveloper 10g



10.1.3 also allows users to control which subdirectories or files to show or hide in a project by setting up inclusion and exclusion rules. These rules apply to both the **Applications Navigator** and **System Navigator** views of the project. For example:

- **To control the default application source content that appears in a project** (content that appears in the **Application Sources** folder of the **Applications Navigator** view) - select the **Project Content** node in the **Project Properties** dialog. In the **Project Content** page that appears, you can add or remove files or subfolders from the **Include** or **Exclude** lists to show or hide specific content, as shown in Figure 5.

Figure 5: Project Content node of Project Properties



- **To control the default web content that appears in a project** (content that appears in the **Web Content** folder of the **Applications Navigator** view) - select the **Project Content** > **Web Application** node in the **Project Properties** dialog. In the **Project Content: Web Application** page that appears, you can add or remove files or subfolders from the **Include** or **Exclude** lists. As an Oracle E-Business Suite developer, you may want to add the **cabo** and **jsp** subfolders to the **Exclude** list, as you generally do not need to work with these files.

Note:

When you create a new OA project, by default, all files are included in and no files are excluded from the project. To keep the customization of your project view simple, we recommend that you not modify the **Include** list, but simply add files or subfolders that you wish to hide to the **Exclude** list.

Working Sets

Oracle JDeveloper 10g allows you to create a "working set" of your project. A working set is a subset of files from your project source path contents that you want to work with. By creating a working set, you can build just the working set without building the entire project. You can specify directories, as well as specific file types or files to hide/show in a working set. The working set, however, is only available in the **System Navigator** view.

To manage working sets, select the **Working Set** icon from the **System Navigator** toolbar and choose **Manage Working Sets...** from the dropdown menu, or choose **View** > **Options** > **Manage Working Sets**.

On the **Files** tab of the **Working Sets** dialog, as shown in Figure 6, you can select **Add File Filter...** or **Add Directory Filter...** to display dialogs that allow you to create file-level or directory-level filters, respectively.

Figure 6: Working Set dialog

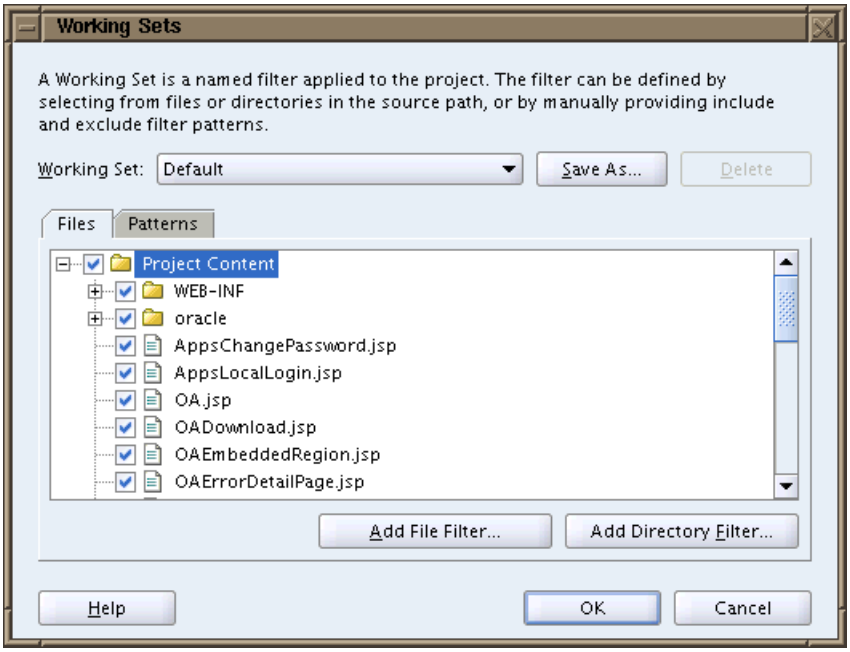


Figure 7: Add File Filter dialog

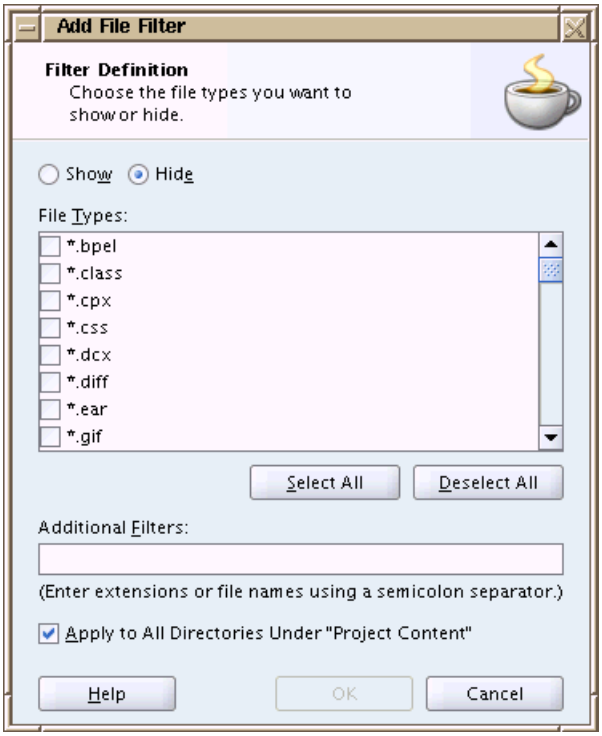


Figure 8: Add Directory Filter dialog



For additional information about working sets, refer to the topic "Managing Working Sets" in *Getting Started with JDeveloper*. (**Tip:** To view this topic, select **Help** > **Full Text Search** from the JDeveloper 10g menu and enter **Managing Work Sets** as the text string to search.)

Note:

Unlike the project property include/exclude rules that you can display in the Applications or System navigators, the rules for a working set apply only to the directories and files displayed in the **System Navigator**. A working set is also saved on a user home and not on a project, thereby making it difficult for other users or projects to share.

Deploying Files

In 9.0.3, files are not deployed until you run a JSP file. In 10.1.3, files are now deployed when you first open a project.

Appendix I: Recommended Browsers

Overview

Refer to My Oracle Support Knowledge Document 389422.1, *Recommended Browsers for Oracle E-Business Suite Release 12*.

Appendix J: OA Framework Support Guidelines for Customers

Overview

This document supplements the *Oracle Application Framework Developer's Guide*, which describes the extensive set of features and capabilities made available by Oracle Application Framework. The information included in that book is intended to provide you with a complete understanding of the technology, tools, and standards upon which the numerous OA Framework-based applications in Oracle E-Business Suite are built.

Oracle E-Business Suite Release 12 introduced many new features, including tools to perform types of customization to OA Framework-based applications that could not be performed in earlier releases.

This document describes the prerequisites necessary to support certain features, and the support options that are available for performing customizations. When reviewing the capabilities and methods presented below, it is important to understand that the resources available to support your work depend upon the extent and type of customizations you plan to perform.

The most current version of this document is published in My Oracle Support Knowledge <Document 395441.1>. Before starting any customization work, you should review the latest version in case more information has been added or other updates made.

Contents

- Browser Prerequisites
- Available Support Options
- Limitations and Guidelines

Browser Prerequisites

Oracle E-Business Suite Release 12.1.2 introduced the following rich user interface features:

- Inline Attachments
- Look Ahead LOV
- Pop-ups
- Navigator/Favorites Pull-down Menus
- RESTful Service Interface

These features are supported with a subset of the certified browsers described in My Oracle Support Knowledge <Document 389422.1>, *Recommended Browsers for Oracle E-Business Suite Release 12*.

Available Support Options

Release 12 of Oracle Application Framework provides significant new capabilities to perform personalizations, and extend OA Framework based web applications. For a full description of available options, refer to the "Customization Primer" in the *Oracle Applications Framework Personalization Guide*. In this document, the term "customizations" collectively refers to those capabilities.

The objective of this note is to assist Oracle E-Business Suite customers with understanding the level of support provided for the different types of customizations possible, including alternative resources that are available for certain types of work but fall beyond the scope of standard support processes.

Personalizations

Personalizations performed within the scope of the OA Personalization Framework are a fully supported means of customizing OA Framework based applications.

Due to its declarative nature and durable architecture, the OA Personalization Framework continues to be recommended as the primary means for customizing OA Framework based applications. The supported capabilities, methods and tools for performing personalizations are documented in the *Oracle Application Framework Personalization Guide*. The most current version of this Personalization guide is published in the Oracle E-Business Suite Documentation Library, which is supplied on a physical CD in the Oracle E-Business Suite Release 12 software bundle.

Access the latest content from the Oracle E-Business Suite Online Documentation Library.

Methods or capabilities that not detailed in the *Oracle Application Framework Personalization Guide*, fall beyond the scope of the OA Personalization Framework, and are not supported for Oracle E-Business Suite installations.

Customers utilizing the capabilities of the OA Personalization Framework must ensure their Release 12 instance is kept current with the latest OA Framework patchset applied. When reporting issues against Personalization, Oracle Support will as a first step, require you to check and confirm you have applied the most current patchset to ensure that the latest fixes for known issues have been applied to your instance. You can find more information about the current patchset, if available, including known issues addressed in

that patchset by referring to the latest Oracle Application Framework Release Notes on My Oracle Support.

Extensions

Release 12 of the OA Framework and the accompanying Oracle 10*g* JDeveloper release provide features for developing a new class of Oracle applications extensions not available to customers in prior releases.

Assistance with customer-developed extensions is available via the following resources:

- **Oracle Application Framework Developer's Guide**
  The Developer's Guide fully documents the capabilities of the Framework including instructions, examples and essential standards for implementing business-tier objects, UI components and server-side features. Specifically, Chapter 9 of the Developer's Guide under the section "Extending OA Framework Applications", provides instructions on how to extend OA Framework based applications with custom business logic, including detailed guidelines for BC4J code extensions.

- **Oracle Application Framework ToolBox Tutorial Application**
  The ToolBox Tutorial application is a sample application accompanied by extensive examples with step-by-step instructions that demonstrate the usage of business objects and UI components to build OA Framework based application pages, against a simple Purchase Order type application schema, installed on your Release 12 instance.
  The ToolBox includes a specific tutorial lab on "Extending OA Framework Applications".

- **OA Framework JavaDoc**
  Documents all core Oracle Application Framework packages and classes, including UIX and BC4J objects extended by the Framework.

- **OA Framework Discussion Forum on the Oracle Technology Network**
  OTN hosts a discussion forum for Technology - OA Framework . Navigate to OTN Forums under the E-Business Suite (`http://forums.oracle.com/forums/category.jspa?categoryID=3`). You can use the forum to post questions and exchange information with other customers on the OTN community working on extensions. The OA Framework Development team and Oracle Support will monitor and participate in some of the discussion threads on this forum. Additionally, you may also consider participating in the OTN JDeveloper forum for usage questions concerning Oracle JDeveloper 10*g*.

- **Oracle E-Business Suite Product Documentation**
  Some products may provide additional information on extending application specific business objects and functionality. Consult My Oracle Support (`https://support.oracle.com`) under the respective product for more information.

For issues logged with Oracle Support to address questions concerning OA Framework based extensions or usage of the OA Extension tool, Oracle Support will evaluate the nature of the question, and in most cases refer the customer to one or more of the resources outlined above.

Limitations and Guidelines

Before starting work on any customizations, it is essential that customers be aware of the following limitations and guidelines:

- Customers who intend to work with Oracle JDeveloper 10*g* OA Extension, and develop extensions to their installed OA Framework-based self-service applications must use the specific build of the Oracle JDeveloper 10*g* that corresponds to the OA Framework Release 12 installed in their runtime environment. Consult the corresponding "About Oracle E-Business Suite Technology Update" document for the JDeveloper ARU that corresponds to the Release 12 ATG patchset you install.

- Oracle does not provide access to Java source code for OA Framework or products. You should consider the Developer's guide and available JavaDoc for the classes you are working with as the only documented sources of information available to determine the characteristics of the object you are extending.

- Design-time options and expected run-time behavior of OA Framework components are fully documented in the Developer's Guide and JavaDoc mentioned above.

- In order to log issues with Oracle Support concerning components, such as unexpected run-time behavior of a component, customers will be required to provide a simple reproducible test case *written against the OA Framework ToolBox Tutorial schema* or an *E-Business Suite product schema*. The test case must not rely on any custom schema elements or custom class libraries, and must be runnable by Oracle Support without any custom dependencies.

- Oracle does not recommend that customers extend controller objects associated with regions or web beans in shipped E-Business Suite product pages. Controller class (`oracle.apps.fnd.framework.webui.OAControllerImpl`) methods should effectively be considered private, since their implementation is subject to change. *Controller extensions are therefore not considered to be durable between upgrades*. If it is absolutely essential to handle custom form submit events on a shipped product page, `processFormRequest()` is the only method that should be overriden in a controller class, although the risks outlined above still apply.

- Customers are fully responsible for all custom code written to support customer-developed extensions. This includes any custom Look-and-Feel (or custom skin) that you develop and deploy. Oracle Support and Oracle E-Business Suite development will not review custom code or custom skins. Questions such as those relating to design, and usage of components to develop extensions, will generally be redirected to the OTN forums mentioned above.
- Each release of Oracle E-Business Suite is associated with a standard Look-and-Feel / skin. Oracle only supports Look-and-Feel issues on the standard Look-and-Feel for that release, and not for any other Look-and-Feel, including sample skins uch as "Minimal Look and Feel" and "Simple Look and Feel" that are provided as basic references for custom skin development.
- To facilitate transparent upgrades and new feature uptake, custom code must comply with the Oracle E-Business Suite OA Framework coding standards described in Chapter 8 of the OA Framework Developer's Guide
- .
- Customers planning to undertake advanced or complex extension projects may consider engaging services available from Oracle Consulting or Oracle Partner resources. Oracle Consulting and Partner organizations offer an alternative means of support through consulting resources who have been specially trained or certified on OA Framework and Oracle E-Business Suite technology. For more information on what options are available, refer to the information under Oracle Consulting Services (`http://www.oracle.com/us/products/consulting/index.html`) and the Oracle Partner Network (`http://www.oracle.com/partners/index.html`).

Glossary

A B C D E F G-L M-N O P-Q R S-T U -Z

A

- **Accelerator Key (also known as a "Hot Key")** - Predefined numers or letters that let users quickly execute actions using the keyboard instead of the mouse. In Windows, users exercise the accelerator keys by selecting **Alt + <char>** from the keyboard.
- **Accessible** - The product meets the standards of United States Federal law section 508, Part 1194: Electronic and Information Technology Accessibility Standards and other guidelines that provide for a comfortable, productive experience for all users:
    - the product must be usable without a mouse (keyboard only)
    - the product must be usable by a blind user (with a screen reader or braille reader)
    - there must be no reliance on sound
    - there must be no reliance on color
    - there must be no reliance on animation or timing
- **AK Repository** - The declarative metadata repository used with pre-11.5.10 versions of the OA Framework. In release 11.5.10, this is replaced with MDS.
- **Application Module** - A BC4J container for related objects (meaning they all participate in the same task). An application module usually contains code that serves as an intermediary between the controller code and the model code (helping to separate the client side code from the server side code). Application modules are attached to page definitions and sometimes to region definitions.
- **Application Service** - A service bean created specifically for user interfaces. See Service Bean.
- **ARCS** - Oracle E-Business Suite internal source control system.
- **ARU** - The Automated Release Update system automates the process of defining, building, packaging and distributing bug fixes and new functionality to Oracle customers. All Oracle E-Business Suite applications (and the OA Framework itself) are shipped via an ARU which customers can download from My Oracle Support.
- **Association Object** - BC4J association objects implement the relationships between entity objects. For example, a purchase order header can reference a supplier, or it can own its order lines.
- **Attribute Set** - Bundles of region or item properties that can be reused either as is or with modifications. For example, all buttons sharing the same attribute set would have the same label and Alt text.

B

- **BC4J** - Oracle Business Components for Java framework leveraged by the OA Framework to implement model behavior in the MVC architecture.
- **Bookmarkable Page** - An OA Framework-based page that can be accessed across Oracle E-Business Suite user sessions.
- **Bound Value** - A web bean value that is dynamically obtained from an underlying data source and resolved to the component at rendering time.
- **Browser Tier** - Refers to a browser for client access in a typical three-tier JSP application. Also referred to as a client tier.
- **Business Object** - A business object is a self-contained representation of a real-world object in the business domain: a document, a place, a person or a thing. From an implementation perspective, a business object involves one or more entities related through composition (meaning that child entities cannot exist without their parent).
- **Business Object Service** - A service bean designed for use by programmatic clients (as opposed to a user interface).

C

- **Client Action** - A UIX construct for associating client-side behavior with a web bean. Specifically, in the OAF, Client Actions can be configured to submit the form (see Fire Action) and execute a partial page render (see Partial Action).
- **Client Side** - Within the middle tier (web application server) of a JSP application, the UI web bean hierarchy is constructed and the application business logic executes. OA Framework refers to the client side as the client classes (view and controller code) in this middle tier that drive the HTML user interface.
- **Controller (1)** - In the context of a Model-View-Controller application (MVC), the controller responds to user actions and directs

application flow.

- **Controller (2)** - In an OA Framework application, this term is often used to specifically refer to the Java UI Controller associated with one or more regions in a page.
- **CSS** - HTML cascading style sheet used to control fonts, colors, indentation and other formatting.

D

- **Database Tier** - Refers to the database for enterprise data in a typical three-tier JSP application.

E

- **Encoding (Parameter)** - The process of converting a URL parameter value to comply with the HTTP syntax rules. For example, blank spaces are illegal, so the parameter value buyerName=John Doe would be encoded as buyerName=John%20Doe.
- **Encryption (Parameter)** - The process of obfuscating a sensitive URL parameter value to make it unintelligable to a user.
- **Enterprise Java Bean (EJB)** - Standalone, distributed, server-side software components designed to run in a special environment called an EJB container. This container manages the life cycles of its beans and provides transaction services. It also abstracts the location of the bean itself.
- **Entity Expert** - A special singleton class registered with an entity object (EO) that performs operations on behalf of the EO.
- **Entity Object (EO)** - BC4J entity objects encapsulate the business rules (validations, actions and so on) associated with a row in a database table, view, synonym or snapshot.

F

- **Fire Action** - A type of client action. When associated with a bean, this submits the enclosing form when the bean is selected (or, in some cases depending on the component type, when the cursor leaves the bean).
- **Form** - A form is an HTML construct that groups data entry controls like fields (both hidden and visible), poplists and so on with action controls (like buttons) that are capable of "submitting the form." When the user selects a submit button, for example, the browser issues a POST request which sends the form's data to the server.

  Tip: People often use the terms "POST" and "submit form" interchangeably when talking about the OA Framework.

G - L

- **Item** - Leaf node UI components like buttons, links, text fields, poplists and so forth which have no children.
- **JavaBean (or "bean" for short)** - A reusable component that implements specific design patterns to make it easy for programmers and development tools to discover the object's properties and behavior.
- **JSP** - A file with some HTML and Java code that executes top to bottom. At runtime, it is compiled into a Java class which is actually a servlet.
- **LOV** - A user interface control that lets users choose a value from a predefined list of values for the purpose of populating one or more fields on a page.

M-N

- **MDS** - The declarative metadata repository used with 11.5.10+ OA Framework applications. This replaced the older AK Repository.
- **Middle Tier** - Refers to the web application server in a typical three-tier JSP application. The middle tier is where the application objects lives and is also referred to as the business logic tier.
- **Multiple Language Support (MLS)** - Implies that an entity's values can be translated into multiple languages.
- **Model** - In the context of a Model-View-Controller application (MVC), the model encapsulates the underlying data and business logic of the application. It also provides an abstraction of the real-world business object(s) and application service(s) that it provides.
- **Model-View-Controller (MVC**) - A design pattern that involves dividing a module (which can vary in scope from a single component to a complete application) into three areas of functionality: the model, the view and the controller.

O

- **Oracle E-Business Suite User Session** - When the user logs in to an OA Framework application, the OA Framework creates an AOL/J `oracle.apps.fnd.comon.WebAppsContext` object and a browser session-based cookie that together keep track of key Oracle E-Business Suite context information like the current responsibility, organization id and various user attributes (user name, user id, employee id and so on). The Oracle E-Business Suite user session is associated with a servlet session, however, it has its own life cycle and time-out characteristics.
- **Object Version Number (OVN)** - A standard column used for locking rows in the database. When a new row is inserted into a table, the table's OVN column value for that row is set to a nominal value, typically one. The OVN value is then incremented whenever the row is updated. This value is preserved until the next update or delete, and is never decrimented or reset to a previous value.

P-Q

- **Package (1)** - For Java files and individual UI component definition XML files, the package corresponds to a physical directory storing related files.
- **Package (2)** - For OA components, a package refers to a file that contains attributeSets. AttributeSets are a collection of properties that can be re-used in OA components.
- **Page** - A hierarchy of regions that is accessible using a URL. Pages generally represent a cohesive unit of functionality that may stand on its own, or work in concert with other pages.
- **PageLayout** - A region that is a high-level layout element that is a template for the entire page. It supports several navigation and content areas for the creation of pages.
- **Passive [Query] Criteria** - Any LOV query criteria that you specify programmatically in an associated LOV controller.
- **Partial Action** - A type of client action. When associated with a bean, this triggers a PPR event when the bean is selected (or, in some cases depending on the component type, when the cursor leaves the bean).
- **Partial Page Rendering (PPR)** - A mechanism for selectively refreshing parts of a page (as opposed to the whole page) in response to user actions. For example, if a user sorts a table, only the table's contents changes; the rest of the page remains the same.
- **Personalization** - The ability for customers to declaratively alter the UI to suit user or business needs.
- **Quik Apache** - A web-based, centralized tool that used in the Oracle E-Business Suite development group to quickly configure dedicated listeners for application testing.

R

- **Region** - Rectangular area that determines layout and UI component presentation in the user interface. Regions are containers which can contain both regions and items. Common region examples include tables, flow layouts, headers and so on.
- **Region Item** - An older OA Framework term for an item.
- **Render** - When web beans are "rendered," UIX includes them in the web bean hierarchy. Furthermore, HTML is generated for the component and sent to the browser. Note that "rendered" doesn't necessarily mean "displayed" (although it usually does for most web beans). Some components are always "hidden" (never displayed to the user), but must be rendered to be used. `oracle.apps.fnd.framework.webui.beans.form.OAFormValueBean` and `oracle.apps.fnd.framework.webui.beans.form.OAFormParameterBean` are examples of this case.
- **Rendering Context** - A class that encapsulates all the information that the UIX framework needs to resolve bound values during web bean rendering.
- **Request** - The unit of work for a web application is a request, which is actually a request/response pair. The browser communicates with the middle tier using HTTP (Hyper Text Transfer Protocol) which involves sending a *request* message to which the middle tier replies with a *response* message.
- **Root Application Module** - Each pageLayout region in an OA Framework application is associated with a "root" application module which groups related services and establishes the transaction context. This transaction context can be shared by multiple pages if they all reference the same root application module, and instruct the framework to retain this application module (not return it to the pool) when navigating from page to page within the transaction task.

S-T

- **Server Side** - Within the middle tier (web application server) of a JSP application, the UI web bean hierarchy is constructed

and the application business logic executes. OA Framework refers to the server side as the server classes (model code) in this middle tier that can support any client (not just OA Framework user interfaces).

- **Service Bean** - A self-describing, standalone component that can ultimately be deployed as a web service, an EJB session bean or as a co-located Java API (meaning it is deployed in the same Java Virtual Machine [JVM] as any clients that use it). There are two types of service beans, largely differentiated by their intended usage: Business Object Service and Application Service.
- **Servlet** - A Java-based web server extension program that implements a standard API.
- **Servlet Session** - A mechanism for maintaining state between HTTP requests during a period of continuous interaction between a browser and a web application. A session may be initiated at any time by the application and terminated by the application, by the user closing the browser, or by a period of user inactivity. A session usually corresponds to an application login/logout cycle, although this isn't strictly true for an OA Framework application.
- **Simplest Possible Expression Language (SPEL)** - An industry-standard expression language included in the JSP Standard Tag Library (JSTL).
- **Simple Object Access Protocol (SOAP)** - An XML-based protocol for interacting with objects (sending and receiving messages) on remote systems. All web services use SOAP. Note that SOAP does not specify the underlying transport protocol for message delivery: HTTP, SMTP or any other Internet protocol can be used, although HTTP is the most common.
- **Three-Tier Architecture** - An application architecture that consists of a client tier (or browser tier), a middle tier (or business logic tier) and a database tier.
- **Transaction** - An interface to manage a database operation. All operations to modify data in a transaction must succeed before the server will accept the changes.

U-Z

- **UIX** - Oracle XML user interface framework leveraged by the OA Framework for rendering and interacting with HTML web beans
- **Validation Application Module** - An application module created exclusively for the purpose of grouping and providing transaction context to related validation view objects. Typically, a standalone entity object or the top-level entity object in a composition would have an associated validation application module.
- **Validation View Object** - A view object created exclusively for the purpose of performing light-weight SQL validation on behalf of entity objects or their experts.
- **View (1)** - In the context of a Model-View-Controller application (MVC), the view formats and presents model data to the user.
- **View (2)** - A database construct providing a view of selected data.
- **View Link** - Establishes a master/detail relationship between two view objects.
- **View Object** - In the simplest terms, a BC4J view object encapsulates a database query and provides iteration over and access to the view rows in its result set.
- **View Row** - Represents a single row in a view object.
- **Web Service** - A standalone software program that can be run on the Internet. While a web service can be very small (an application that lets you get the current time, for example), or large and complex, all web services standard technologies that together enable dynamic discovery and interaction.
- **Web Services Definition Language (WSDL)** - An XML-based language that describes the services offered by remote systems. It describes the location of the service, and the operations (methods) that it supports.

Copyright © 2000, 2020, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

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.

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