

Oracle Fusion Middleware 11g: Build Applications with ADF I

Volume II • Student Guide

D53979GC20

Edition 2.0

July 2010

D68161

ORACLE®

Authors

Kate Heap
Patrice Daux

**Technical Contributor
and Reviewer**

Joe Greenwald
Glenn Maslen

Editors

Aju Kumar
Daniel Milne

Graphic Designer

Satish Bettegowda

Publishers

Pavithran S. Adka
Jobi Varghese

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Disclaimer

This document contains proprietary information and is protected by copyright and other intellectual property laws. You may copy and print this document solely for your own use in an Oracle training course. The document may not be modified or altered in any way. Except where your use constitutes "fair use" under copyright law, you may not use, share, download, upload, copy, print, display, perform, reproduce, publish, license, post, transmit, or distribute this document in whole or in part without the express authorization of Oracle.

The information contained in this document is subject to change without notice. If you find any problems in the document, please report them in writing to: Oracle University, 500 Oracle Parkway, Redwood Shores, California 94065 USA. This document is not warranted to be error-free.

Restricted Rights Notice

If this documentation is delivered to the United States Government or anyone using the documentation on behalf of the United States Government, the following notice is applicable:

U.S. GOVERNMENT RIGHTS

The U.S. Government's rights to use, modify, reproduce, release, perform, display, or disclose these training materials are restricted by the terms of the applicable Oracle license agreement and/or the applicable U.S. Government contract.

Trademark Notice

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Contents

I Introduction

- Course Objectives I-2
- Course Agenda: Day 1 I-3
- Course Agenda: Day 2 I-4
- Course Agenda: Day 3 I-5
- Course Agenda: Day 4 I-6
- Course Agenda: Day 5 I-7

1 Introduction to Oracle Fusion and Oracle ADF

- Objectives 1-2
- Examining Oracle Fusion Architecture 1-3
- Oracle Application Development Framework (ADF) 1-4
- Model-View-Controller Design Pattern 1-5
- How the Oracle ADF Framework Implements MVC 1-6
- Technology Choices for ADF BC Applications 1-7
- Introducing JDeveloper: Oracle's Java and Web Development Tool 1-8
- Obtaining Additional Information 1-9
- Summary 1-10

2 Getting Started with JDeveloper

- Objectives 2-2
- Describing the Benefits of Using JDeveloper 2-3
- Launching JDeveloper 2-5
- Defining Roles 2-6
- Using JDeveloper Features 2-7
- Using JDeveloper's Application Navigator 2-8
- Using JDeveloper's Database Navigator 2-10
- Using JDeveloper's Editors 2-11
- Using JDeveloper's Component Palette 2-13
- Using JDeveloper's Resource Palette 2-14
- Using JDeveloper's Structure Window 2-15
- Using JDeveloper's Property Inspector 2-16
- Using JDeveloper's Log Window 2-17
- Working with JDeveloper Windows 2-18
- Setting IDE Preferences 2-19
- Getting Started in JDeveloper 2-20

| | |
|--|------|
| Creating an Application in JDeveloper | 2-21 |
| Using the Application Overview Page | 2-23 |
| The Application Overview Checklist | 2-24 |
| Creating a Project in JDeveloper | 2-25 |
| Creating Database Connections | 2-27 |
| Creating a Database Connection in JDeveloper | 2-28 |
| Describing the Course Application | 2-30 |
| Presenting the Storefront User Interface | 2-31 |
| Summary | 2-32 |
| Practice 2 Overview: Using JDeveloper | 2-33 |

3 Building a Business Model with ADF Business Components

| | |
|---|------|
| Objectives | 3-2 |
| Describing ADF Business Components (ADF BC) | 3-3 |
| ADF BC Implementation Architecture | 3-4 |
| Types of ADF Business Components | 3-5 |
| Creating ADF Business Components | 3-6 |
| Create Business Components from Tables Wizard: Entity Objects | 3-7 |
| Create Business Components from Tables Wizard: Updatable View Objects | 3-8 |
| Create Business Components from Tables Wizard: Read-Only View Objects | 3-9 |
| Create Business Components from Tables Wizard: Application Module | 3-10 |
| Create Business Components from Tables Wizard: Diagram | 3-11 |
| Examining Created Objects | 3-12 |
| Testing the Data Model | 3-13 |
| Exposing the Application Module to the User Interface | 3-14 |
| Summary | 3-15 |
| Practice 3 Overview: Building a Business Model | 3-16 |

4 Querying and Persisting Data

| | |
|---|------|
| Objectives | 4-2 |
| Using View Objects | 4-3 |
| Characteristics of a View Object (VO) | 4-4 |
| Creating View Objects for Queries | 4-5 |
| Testing View Objects with the Business Components Browser | 4-11 |
| Characteristics of an Entity Object (EO) | 4-12 |
| Using Entity Objects to Persist Data | 4-13 |
| Creating Entity Objects | 4-14 |
| Creating Entity Objects from Tables, Views, or Synonyms | 4-16 |
| Synchronizing an Entity Object with Changes to Its Database Table | 4-17 |
| Generating Database Tables from Entity Objects | 4-18 |
| Characteristics of Associations | 4-19 |

| | |
|---|------|
| Creating Associations | 4-21 |
| Association Types | 4-22 |
| Characteristics of Updatable View Objects | 4-23 |
| Creating Updatable View Objects | 4-24 |
| Creating Updatable View Objects: Attributes and Settings | 4-25 |
| Creating Updatable View Objects: Query | 4-27 |
| Creating Updatable View Objects: Additional Settings | 4-29 |
| Interaction Between Views and Entities: Retrieving Data | 4-30 |
| Interaction Between Views and Entities: Updating Data | 4-31 |
| Creating a Join View Object | 4-32 |
| Including Reference Entities in Join View Objects | 4-33 |
| Creating Master-Detail Relationships with View Objects | 4-34 |
| Linking View Objects | 4-35 |
| Comparing Join View Queries with View Links | 4-36 |
| Refactoring Objects | 4-37 |
| Summary | 4-38 |
| Practice 4 Overview: Creating Entity Objects and View Objects | 4-39 |

5 Exposing Data

| | |
|---|------|
| Objectives | 5-2 |
| Oracle ADF Application Module (AM) | 5-3 |
| Characteristics of an Application Module | 5-4 |
| Creating an Application Module | 5-5 |
| Defining the Data Model for the Application Module | 5-6 |
| Using Master-Detail View Objects in Application Modules | 5-7 |
| Determining the Size of an Application Module | 5-8 |
| Business Components Transactions | 5-9 |
| Using Nested Application Modules | 5-11 |
| Shared Application Modules | 5-12 |
| Application Module Pooling | 5-14 |
| Managing Application State | 5-15 |
| Role of ADF Model | 5-16 |
| Describing the Course Application: Database Objects | 5-17 |
| Describing the Course Application: View Objects | 5-18 |
| Describing the Course Application: Data Controls | 5-20 |
| Summary | 5-21 |
| Practice 5 Overview: Defining Application Modules | 5-22 |

6 Declaratively Customizing Data Services

| | |
|--------------|-----|
| Objectives | 6-2 |
| Using Groovy | 6-3 |

| | |
|--|------|
| Using Groovy Syntax in ADF | 6-4 |
| Using Groovy Expressions for Validation | 6-6 |
| Internationalizing the Data Model | 6-8 |
| Editing Business Components | 6-9 |
| Modifying the Default Behavior of Entity Objects | 6-10 |
| Defining Attribute Control Hints | 6-12 |
| Modifying the Default Behavior of Entity Objects | 6-14 |
| Synchronizing with Trigger-Assigned Values | 6-15 |
| Modifying the Default Behavior of Entity Objects | 6-17 |
| Using Alternate Key Entity Constraints | 6-18 |
| Creating Alternate Key Entity Constraints | 6-19 |
| Editing View Objects | 6-21 |
| Modifying the Default Behavior of View Objects | 6-22 |
| Defining View Object Control Hints | 6-24 |
| Modifying the Default Behavior of View Objects | 6-26 |
| Performing Calculations | 6-27 |
| Modifying the Default Behavior of View Objects | 6-28 |
| Restricting and Reordering the Columns Retrieved by a Query | 6-29 |
| Modifying the Default Behavior of View Objects | 6-30 |
| Changing the Order of Queried Rows | 6-31 |
| Modifying the Default Behavior of View Objects | 6-32 |
| Restricting the Rows Retrieved by a Query | 6-33 |
| Using View Criteria (Structured WHERE Clauses) | 6-34 |
| Role of View Criteria in Search Forms | 6-36 |
| Using Parameterized WHERE Clauses | 6-37 |
| Using Named Bind Variables | 6-38 |
| Modifying the Default Behavior of View Objects | 6-39 |
| Retaining and Reusing a View Link Accessor Row Set | 6-40 |
| Modifying the Default Behavior of View Objects | 6-41 |
| Creating View Accessors | 6-42 |
| Using a List of Values (LOV) | 6-43 |
| Defining the View Accessor for the List of Values | 6-44 |
| Defining the List of Values | 6-45 |
| Modifying Application Modules | 6-46 |
| Changing the Locking Behavior of an Application Module | 6-47 |
| Summary | 6-48 |
| Practice 6 Overview: Declaratively Modifying Business Components | 6-49 |

7 Programmatically Customizing Data Services

| | |
|---|-----|
| Objectives | 7-2 |
| Generating Java Classes for Adding Code | 7-3 |

| | |
|---|------|
| Programmatically Modifying the Default Behavior of Entity Objects | 7-4 |
| Supporting Entity Java Classes | 7-5 |
| Traversing Associations | 7-7 |
| Overriding Base Class Methods | 7-8 |
| Overriding Base Class Methods Example: Updating a Deleted Flag Instead of Deleting Rows | 7-9 |
| Overriding Base Class Methods Example: Eagerly Assigning Values from a Database Sequence | 7-10 |
| Programmatically Modifying the Default Behavior of View Objects | 7-11 |
| Supporting View Object Java Classes | 7-12 |
| Examining View Object Methods | 7-13 |
| Changing View Object WHERE or ORDER BY Clause at Run Time | 7-16 |
| Using Named Bind Variables at Run Time | 7-18 |
| Programmatically Retaining and Reusing a View Link Accessor Row Set | 7-19 |
| Traversing Links | 7-20 |
| Application Module Files | 7-22 |
| Centralizing Implementation Details | 7-23 |
| Adding Service Methods to an Application Module | 7-25 |
| Coding the Service Method | 7-27 |
| Publishing the Service Method | 7-28 |
| Testing Service Methods in the Business Components Browser | 7-30 |
| Accessing a Transaction | 7-31 |
| Committing Transactions | 7-33 |
| Customizing the Post Phase | 7-34 |
| Customizing the Commit Phase | 7-35 |
| Using Entity Objects and Associations Programmatically | 7-36 |
| Finding an Entity Object by Primary Key | 7-37 |
| Updating or Removing an Existing Entity Row | 7-38 |
| Creating a New Entity Row | 7-39 |
| Using Client APIs | 7-41 |
| Creating a Test Client | 7-42 |
| Using View Objects in Client Code | 7-43 |
| Using Query Results Programmatically | 7-44 |
| Using View Criteria Programmatically | 7-45 |
| Iterating Master-Detail Hierarchy | 7-47 |
| Finding a Row and Updating a Foreign Key Value | 7-48 |
| Creating a New Row | 7-49 |
| Summary | 7-48 |
| Practice 7 Overview: Programmatically Modifying Business Components | 7-51 |

8 Validating User Input

Objectives 8-2

Validation Options for ADF BC Applications 8-3

Triggering Validation Execution 8-5

Handling Validation Errors 8-6

Specifying the Severity of an Error Message 8-7

Using Groovy Variables in Error Messages 8-8

Storing Error Messages as Translatable Strings 8-9

Defining Validation in the Business Services 8-10

Using Declarative Validation: Built-in Rules 8-11

Defining Declarative Validation 8-12

Using Declarative Validation: Built-in Rules 8-13

Using Declarative Built-in Rules: Collection Validator 8-14

Using Declarative Built-in Rules: Unique Key Validator 8-15

Using Declarative Validation: Built-in Rules 8-16

Using Declarative Built-in Rules: Compare Validator 8-17

Using Declarative Built-in Rules: Key Exists Validator 8-19

Using Declarative Built-in Rules: Length Validator 8-20

Using Declarative Built-in Rules: List Validator 8-21

Using Declarative Built-in Rules: Range Validator 8-22

Using Declarative Built-in Rules: Regular Expression Validator 8-23

Using Declarative Validation: Built-in Rules 8-24

Using Declarative Built-in Rules: Script Expression Validator 8-25

Using Declarative Custom Rules: Entity-Specific Rules (Method Validators) 8-26

Using Declarative Custom Rules: Creating an Entity-Specific Method Validator 8-27

Using Declarative Global Validation Rules 8-28

Creating the Java Class for a Global Rule 8-29

Examining the Generated Skeleton Code 8-30

Modifying the Code for the Global Rule 8-31

Assigning the Global Rule to an Object 8-33

Testing the Global Validation Rule 8-34

Using Programmatic Validation 8-35

Debugging Custom Validation Code with the JDeveloper Debugger 8-36

Using a Domain to Create Custom-Validated Data Types 8-37

Creating and Using a Domain 8-38

Coding Validation in a Domain 8-39

Specifying Validation Order 8-40

Summary 8-41

Practice 8 Overview: Implementing Validation 8-42

| | |
|---|------|
| 9 Troubleshooting ADF BC Applications | |
| Objectives | 9-2 |
| Troubleshooting the Business Service | 9-3 |
| Troubleshooting the UI | 9-4 |
| Using Logging and Diagnostics | 9-5 |
| Displaying Debug Messages to the Console | 9-6 |
| Java Logging | 9-7 |
| Core Java Logging | 9-9 |
| Using ADF Logging | 9-10 |
| Configuring ADF Logging | 9-11 |
| ODL Configuration Overview Editor | 9-13 |
| Creating Logging Configurations in JDeveloper | 9-14 |
| Viewing ODL Logs | 9-15 |
| Using Design-Time Code Validation | 9-16 |
| Auditing Java Code | 9-17 |
| Design-Time XML Validation | 9-18 |
| Design-Time JSPX Validation | 9-19 |
| Using Tools and Utilities | 9-20 |
| Testing Java Code with JUnit | 9-21 |
| Unit Testing with JUnit | 9-22 |
| Using JDeveloper's Profiler | 9-23 |
| Running the Profiler | 9-24 |
| Using Audit Profiles | 9-25 |
| Identifying Search Paths on Windows with FileMon | 9-26 |
| Using the JDeveloper Debugger | 9-27 |
| Understanding Breakpoint Types | 9-28 |
| Using Breakpoints | 9-30 |
| ADF Framework Debugging | 9-31 |
| Using the EL Evaluator | 9-32 |
| ADF Structure Window and ADF Data Window | 9-33 |
| Object Preferences | 9-34 |
| Using Oracle ADF Source Code for Debugging | 9-35 |
| Setting Up Oracle ADF Source Code for Debugging | 9-36 |
| Using Quick Javadoc | 9-37 |
| Setting Breakpoints in Source Code | 9-38 |
| Using Common Oracle ADF Breakpoints | 9-39 |
| Debugging Interactions with the Model Layer | 9-40 |
| Correcting Failures to Display Data | 9-41 |
| Correcting Failures to Invoke Actions and Methods | 9-44 |
| Debugging Life Cycle Events: Task Flows | 9-46 |
| Debugging Life Cycle Events: Parameters and Methods | 9-47 |

| | |
|--|------|
| Debugging Life Cycle Events: Switching Between the Main Page and Regions | 9-48 |
| Obtaining Help | 9-49 |
| Requesting Help | 9-50 |
| Summary | 9-51 |
| Practice 9 Overview: Troubleshooting | 9-52 |

10 Understanding UI Technologies

| | |
|---|-------|
| Objectives | 10-2 |
| Enabling the World Wide Web with HTML and HTTP | 10-3 |
| Describing the Java Programming Language | 10-4 |
| Using Java as a Language for Web Development | 10-6 |
| What Are Servlets? | 10-8 |
| JavaServer Pages (JSP) | 10-9 |
| What Are JavaBeans? | 10-10 |
| What Is JavaServer Faces (JSF)? | 10-11 |
| JSF Key Concepts | 10-13 |
| JSF Component Model | 10-14 |
| JSF Multiple Renderers | 10-15 |
| Traditional Navigation | 10-16 |
| Defining Navigation by Using the JSF Controller | 10-17 |
| JSF Navigation: Example | 10-18 |
| Using JSF Components | 10-19 |
| Using JSF Managed Beans | 10-20 |
| Overview of JSF Page Life Cycle | 10-21 |
| Formal Phases of the JSF Life Cycle | 10-22 |
| Key Characteristics of Rich User Interfaces | 10-24 |
| Adding to JSF with ADF Faces | 10-25 |
| Using the ADF Controller | 10-26 |
| ADF Life Cycle Phases | 10-27 |
| Summary | 10-28 |

11 Binding UI Components to Data

| | |
|----------------------------------|-------|
| Objectives | 11-2 |
| Creating a JSF Page | 11-3 |
| Adding UI Components to the Page | 11-5 |
| Using the Component Palette | 11-6 |
| Using the Context Menu | 11-7 |
| Using the Data Controls Panel | 11-8 |
| Oracle ADF Data Controls | 11-9 |
| Describing the ADF Model Layer | 11-10 |
| Types of Data Bindings | 11-11 |

| | |
|---|-------|
| Using Expression Language (EL) | 11-13 |
| Expression Language and Bindings | 11-14 |
| Creating and Editing Data Bindings | 11-16 |
| Rebinding: Example | 11-17 |
| Opening a Page Definition File | 11-18 |
| Editing Bindings in a Page Definition File | 11-19 |
| Editing Bindings from a Page | 11-20 |
| Tracing Data Binding: From Database to Databound Components | 11-21 |
| Tracing Data Binding: From AM to Data Control | 11-22 |
| Tracing Data Binding: Creating Databound Components | 11-23 |
| Tracing Data Binding: From Data Control to Databound Components | 11-24 |
| Examining Data Binding Objects and Metadata Files | 11-27 |
| Binding Existing Components to Data | 11-29 |
| Accessing Data Controls and Bindings Programmatically | 11-30 |
| Running and Testing the Page | 11-31 |
| Summary | 11-32 |
| Practice 11 Overview: Creating Databound Pages | 11-33 |

12 Planning the User Interface

| | |
|---|-------|
| Objectives | 12-2 |
| Describing the Model-View-Controller (MVC) Design Pattern | 12-3 |
| Implementing MVC with the ADF Framework | 12-4 |
| Characteristics of ADF Task Flows | 12-5 |
| Characteristics of Unbounded ADF Task Flows | 12-6 |
| Working with Unbounded Task Flows | 12-7 |
| Characteristics of Bounded Task Flows | 12-8 |
| Comparing Unbounded and Bounded Task Flows | 12-10 |
| Bounded and Unbounded ADF Task Flows: Example | 12-11 |
| Creating an Unbounded Task Flow | 12-12 |
| Creating a Bounded Task Flow | 12-13 |
| Converting Task Flows | 12-14 |
| Using a Bounded Task Flow | 12-15 |
| Using ADF Task Flow Components | 12-16 |
| Defining ADF Control Flow Rules | 12-19 |
| Example of ADF Control Flow Rules | 12-20 |
| Using the Navigation Modeler to Define Control Flow | 12-21 |
| Using the Configuration Editor to Define Control Flow | 12-22 |
| Editing the.xml File to Define Control Flow | 12-23 |
| Using the Structure Window to Modify a Task Flow | 12-24 |
| Using Wildcards to Define Global Navigation | 12-25 |
| Using Routers for Conditional Navigation | 12-27 |

| | |
|--|-------|
| Defining Router Activities | 12-28 |
| Calling Methods and Other Task Flows | 12-29 |
| Defining a Task Flow Return Activity | 12-30 |
| Making View Activities Bookmarkable (or Redirecting) | 12-31 |
| Adding UI Code | 12-32 |
| Incorporating Validation into the User Interface | 12-33 |
| Describing the Course Application: UI Functionality | 12-34 |
| Summary | 12-35 |
| Practice 12 Overview: Defining Task Flows | 12-36 |

13 Adding Functionality to Pages

| | |
|---|-------|
| Objectives | 13-2 |
| Internationalization | 13-3 |
| Resource Bundles | 13-4 |
| Steps to Internationalize an Application | 13-5 |
| Automatically Creating a Resource Bundle | 13-7 |
| Automatically Creating a Text Resource | 13-8 |
| Using Component Facets | 13-9 |
| Using ADF Faces Rich Client Components | 13-10 |
| Using ADF Faces Input Components | 13-11 |
| Defining a List | 13-13 |
| Defining Lists at the Model Layer | 13-14 |
| Selecting a Value from a List | 13-15 |
| Selecting a Date | 13-16 |
| Using ADF Faces Table and Tree Components | 13-18 |
| Using Tables | 13-19 |
| Rendering (Stamping) the Table Data | 13-20 |
| Setting Table Attributes | 13-21 |
| Using Trees | 13-23 |
| Using Tree Tables | 13-24 |
| Providing Data for Trees | 13-26 |
| ADF Faces panelCollection Component | 13-28 |
| Using ADF Faces Output Components | 13-29 |
| Using ADF Faces Query Components | 13-30 |
| Using the af:query Component | 13-31 |
| Using the af:quickQuery Component | 13-33 |
| Creating a Query Search Form | 13-35 |
| Modifying Query Behavior | 13-36 |
| Using ADF Data Visualization Components | 13-37 |
| Visualizing Data | 13-38 |

Summary 13-40
Practice 13 Overview: Using ADF Faces Components 13-41

14 Implementing Navigation on Pages

Objectives 14-2
Using ADF Faces Navigation Components 14-3
Performing Navigation 14-4
Using Buttons and Links 14-5
Defining Access Keys 14-6
Defining Tool Tips 14-8
Using Toolbars, Toolbar Buttons, and Toolboxes 14-9
Using Menus for Navigation 14-10
Creating Menus 14-11
Creating Pop-Up Menus 14-13
Creating Context Menus 14-14
Using a Navigation Pane 14-16
Using Breadcrumbs 14-17
Using Explicitly Defined Breadcrumbs 14-18
Using XML Menu Model for Dynamic Navigation Items 14-19
Creating an ADF Menu Model 14-20
Examining the ADF Menu Model 14-22
Binding the Navigation Pane to an XML Menu Model 14-23
Binding Breadcrumbs to an XML Menu Model 14-25
Defining a Sequence of Steps 14-26
Creating a Train 14-28
Skipping a Train Stop 14-29
Summary 14-30
Practice 14 Overview: Using ADF Faces Navigation Components 14-31

15 Achieving the Required Layout

Objectives 15-2
Using ADF Faces Layout Components 15-3
Adding Spaces and Lines: Spacer and Separator 15-5
Stretching Components 15-6
Enabling Automatic Component Stretching: Panel Splitter or Panel Stretch Layout 15-7
Stretching a Table Column 15-8
Creating Resizable Panes: Panel Splitter 15-9
Printing Layout Panel Content: Show Printable Page Behavior Operation 15-10
Creating Collapsible Panes: Panel Splitter 15-11
Creating Collapsible Panes: Panel Accordion 15-12

| | |
|---|-------|
| Panel Accordion Overflow | 15-13 |
| Setting Panel Accordion Properties | 15-14 |
| Arranging Items in Columns or Grids: Panel Form Layout | 15-16 |
| Creating Stacked Tabs: Panel Tabbed with Show Detail Item | 15-18 |
| Hiding and Displaying Groups of Content: Show Detail | 15-19 |
| Arranging Items Horizontally or Vertically, with Scrollbars: Panel Group Layout | 15-21 |
| Displaying Table Menus, Toolbars, and Status Bars: Panel Collection | 15-23 |
| Creating Titled Sections and Subsections: Panel Header | 15-25 |
| Grouping Related Components: Group | 15-26 |
| Displaying a Bulleted List: Panel List | 15-27 |
| Displaying Items in a Content Container Offset by Color: Panel Box | 15-28 |
| Arranging Content Around a Central Area: Panel Border Layout | 15-29 |
| Arranging Content Around a Central Area: Panel Stretch Layout | 15-31 |
| Using ADF Faces Skins | 15-32 |
| ADF Faces Skins | 15-33 |
| Using Dynamic Page Layout | 15-34 |
| Using Expression Language to Conditionally Display Components | 15-35 |
| Characteristics of Partial Page Rendering (PPR) | 15-36 |
| Enabling PPR Declaratively | 15-37 |
| Native PPR: Example | 15-39 |
| Declarative PPR: Example | 15-40 |
| Enabling PPR Programmatically | 15-42 |
| Enabling Automatic PPR | 15-43 |
| Conforming to PPR Guidelines | 15-44 |
| Summary | 15-45 |
| Practice 15 Overview: Using ADF Faces Layout Components | 15-46 |

16 Ensuring Reusability

| | |
|--|-------|
| Objectives | 16-2 |
| Benefits of Reusability | 16-3 |
| Designing for Reuse | 16-4 |
| Using a Resource Catalog | 16-5 |
| Creating a Resource Catalog | 16-6 |
| Reusing Components | 16-7 |
| Creating an ADF Library | 16-8 |
| Adding an ADF Library to a Project by Using the Resource Palette | 16-9 |
| Removing an ADF Library from a Project | 16-10 |
| Restricting BC Visibility in Libraries | 16-11 |
| Types of Reusable Components | 16-12 |
| Using Task Flow Templates | 16-13 |

| | |
|--|-------|
| Characteristics of Page Templates | 16-14 |
| Creating a Page Template | 16-15 |
| Editing a Page Template | 16-17 |
| Applying a Page Template to a Page | 16-18 |
| Characteristics of Declarative Components | 16-19 |
| Creating a Declarative Component | 16-21 |
| Using a Declarative Component on a Page | 16-22 |
| Characteristics of Page Fragments | 16-23 |
| Creating a Page Fragment | 16-24 |
| Using a Page Fragment on a Page | 16-25 |
| Characteristics of Regions | 16-26 |
| Wrapping a Task Flow as a Region | 16-27 |
| Converting a Bounded Task Flow to Use Page Fragments | 16-28 |
| Deciding Which to Use | 16-29 |
| Summary | 16-30 |
| Practice 16 Overview: Implementing Reusability | 16-31 |
| 17 Passing Values Between UI Elements | |
| Objectives | 17-2 |
| Holding Values in the Data Model (Business Components) | 17-3 |
| Holding Values in Managed Beans | 17-4 |
| Using Managed Properties | 17-5 |
| Managed Properties: Examples | 17-6 |
| Using Memory-Scoped Attributes | 17-7 |
| Memory Scope Duration with a Called Task Flow | 17-9 |
| Memory Scope Duration with a Region | 17-10 |
| Accessing Memory-Scoped Attribute Values | 17-11 |
| Using Memory-Scoped Attributes Without Writing Java Code | 17-12 |
| Overview of Parameters | 17-13 |
| Using Page Parameters | 17-14 |
| Job of the Page Parameter | 17-15 |
| Using Task Flow Parameters | 17-16 |
| Job of the Task Flow Parameter | 17-18 |
| Using Region Parameters | 17-19 |
| Job of the Region Parameter | 17-20 |
| Developing a Page Independently of a Task Flow | 17-21 |
| Using View Activity Parameters | 17-22 |
| Job of the View Activity Parameter | 17-23 |
| Summary: Passing a Value from a Containing Page to a Reusable Page Fragment in a Region | 17-24 |
| Passing Values to a Task Flow from a Task Flow Call Activity | 17-25 |

Returning Values to a Calling Task Flow 17-27
Deciding Which Type of Parameter to Use 17-29
Summary 17-30
Practice 17 Overview: Passing Values Between Pages 17-31

18 Responding to Application Events

Objectives 18-2
Adding UI Code 18-3
Creating Managed Beans 18-4
Registering Existing Java Classes as Managed Beans 18-5
Configuring Managed Beans 18-6
Referencing Managed Beans 18-7
Describing JSF and ADF Life-Cycle Roles 18-8
Coordinating JSF and ADF Life Cycles 18-9
Specifying When to Refresh Binding Executables 18-11
Specifying Whether to Refresh Binding Executables 18-13
Describing Types of Events 18-14
Using Phase Listeners 18-15
Using Event Listeners 18-16
Responding to Action Events 18-17
Creating Action Methods 18-18
Using Action Listeners 18-19
Value Change Events 18-20
Listening for Value Change Events 18-21
Event and Listener Execution Order 18-22
ADF Faces Enhanced Event Handling 18-23
Using JavaScript in ADF Faces Applications 18-24
Other ADF Faces Server Events 18-25
Using Table Model Methods in a Selection Listener 18-27
Using Tree Model Methods in a Selection Listener 18-28
Additional AJAX Events 18-29
Characteristics of the Contextual Event Framework 18-30
Contextual Events: Overview 18-31
Using the Contextual Event Framework to Coordinate Page Regions 18-32
Using the Contextual Event Framework to Coordinate Page Regions: Step 1 18-33
Using the Contextual Event Framework to Coordinate Page Regions: Step 2 18-34
Using the Contextual Event Framework to Coordinate Page Regions: Step 3 18-35
Using the Contextual Event Framework to Coordinate Page Regions: Step 4 18-36
Using the Contextual Event Framework to Coordinate Page Regions: Step 5 18-37
Using the Contextual Events Tab to Define an Event 18-38
Using the Contextual Event Framework to Coordinate Page Regions: Step 6 18-39

| | |
|---|-------|
| Using the Contextual Event Framework to Coordinate Page Regions: Step 7 | 18-40 |
| Using the Contextual Event Framework to Coordinate Page Regions: Step 8 | 18-41 |
| Using the Contextual Events Tab to Map the Event | 18-42 |
| Using the Contextual Event Framework to Coordinate Page Regions: Step 9 | 18-44 |
| Summary | 18-45 |
| Practice 18 Overview: Responding to Events | 18-46 |
| 19 Implementing Transactional Capabilities | |
| Objectives | 19-2 |
| Handling Transactions with ADF BC | 19-3 |
| Default ADF Model Transactions | 19-4 |
| Transactions in Task Flows | 19-5 |
| Controlling Transactions in Task Flows | 19-6 |
| Transaction Support Features of Bounded Task Flows | 19-7 |
| Specifying Task Flow Transaction Start Options | 19-8 |
| Specifying Task Flow Return Options | 19-9 |
| Enabling Transactions on a Task Flow | 19-11 |
| Sharing Data Controls | 19-13 |
| Handling Transaction Exceptions | 19-15 |
| Designating an Exception Handler Activity | 19-16 |
| Defining Response to the Back Button | 19-17 |
| Saving for Later | 19-19 |
| Enabling Explicit Save for Later | 19-21 |
| Enabling Implicit Save for Later | 19-23 |
| Restoring Savepoints | 19-24 |
| Setting Global Save for Later Properties | 19-26 |
| Summary | 19-27 |
| Practice 19 Overview: Controlling Transactions | 19-28 |
| 20 Implementing Security in ADF Applications | |
| Objectives | 20-2 |
| Benefits of Securing Web Applications | 20-3 |
| Examining Security Aspects | 20-4 |
| ADF Security Framework: Overview | 20-5 |
| Configure ADF Security Wizard: Configuring ADF Security Authentication | 20-6 |
| Configure ADF Security Wizard: Choosing the Authentication Type | 20-7 |
| Using Form-Based Authentication | 20-8 |
| Configure ADF Security Wizard: Choosing the Welcome Page | 20-9 |
| Configure ADF Security Wizard: Enabling ADF Authorization | 20-10 |
| Files Modified by Configure ADF Security Wizard: web.xml | 20-11 |
| Other Files Modified or Created by Configure ADF Security Wizard | 20-12 |

| | |
|---|-------|
| Enabling Users to Access Resources | 20-13 |
| Defining Users and Roles in the Identity Store | 20-14 |
| Defining Security Policies | 20-15 |
| Defining Application Roles in the Policy Store | 20-16 |
| Assigning Identity Store Roles to Application Roles | 20-17 |
| Granting Permissions to Roles | 20-18 |
| Securing Groups of Pages (Bounded Task Flows) | 20-19 |
| Securing Individual Pages (Page Definitions) | 20-20 |
| ADF BC Model Authorization | 20-21 |
| Securing Row Data (Entity Objects or Attributes) | 20-22 |
| Granting Privileges on Entity Objects or Attributes | 20-23 |
| Application Authentication at Run Time | 20-24 |
| ADF Security: Implicit Authentication | 20-25 |
| ADF Security: Explicit Authentication | 20-27 |
| ADF Security: Authorization at Run Time | 20-28 |
| Programmatically Accessing ADF Security Context | 20-29 |
| Using Expression Language to Extend Security Capabilities | 20-30 |
| Using Global Security Expressions | 20-31 |
| Using a Security Proxy Bean | 20-32 |
| Summary | 20-33 |
| Practice 20 Overview: Implementing ADF Seurity | 20-34 |

Appendix A: Modeling the Database Schema

| | |
|---|------|
| Objectives | A-2 |
| Modeling Database Schemas | A-3 |
| Goals of Database Modeling | A-4 |
| Characteristics of the JDeveloper Database Modeler | A-5 |
| Database Modeling Tools in JDeveloper | A-6 |
| Modeling Database Objects Offline | A-7 |
| Creating a New Offline Database | A-8 |
| Creating New Schema Objects in an Offline Database | A-9 |
| Creating a Database Diagram | A-10 |
| Importing Tables to the Diagram from an Offline Database | A-11 |
| Editing Objects on the Diagram | A-12 |
| Generating Changes from the Diagram to the Database | A-13 |
| Reconciling Changes to the Database | A-14 |
| Generating Changes from the Offline Database Object to the Database | A-15 |
| Importing Database Objects Without a Diagram | A-16 |
| Presenting the Storefront Schema | A-17 |
| Summary | A-18 |
| Practice Overview: Modeling the Schema for the Course Application | A-19 |

Appendix B: Deploying ADF Applications

| | |
|---|------|
| Objectives | B-2 |
| Steps in the Deployment Process | B-3 |
| Configuring Deployment Options | B-5 |
| Creating Deployment Profiles | B-6 |
| Specifying Deployment Profile Options | B-7 |
| Creating a Business Components Deployment Profile | B-8 |
| Web Module Deployment | B-9 |
| Typical Web Application Deployment Example | B-10 |
| Example: Creating a WAR Deployment Profile for a UI Project | B-11 |
| Example: Creating an EAR Deployment Profile for the Application | B-12 |
| Using Deployment Descriptors | B-13 |
| Steps in the Deployment Process | B-14 |
| Preparing the Oracle WebLogic Server | B-15 |
| Installing the ADF Runtime to the WebLogic Installation | B-16 |
| Creating and Configuring the WebLogic Domain | B-17 |
| Creating a JDBC Data Source | B-19 |
| Configuring the Data Control to Use the Data Source | B-21 |
| Steps in the Deployment Process | B-23 |
| Creating a Connection to an Application Server | B-24 |
| Example: Deploying the Application | B-26 |
| Steps in the Deployment Process | B-27 |
| Deploying the Application from the WebLogic Administration Server Console | B-28 |
| Steps in the Deployment Process | B-29 |
| Using Ant to Automate the Deployment Process | B-30 |
| Creating an Ant Buildfile in JDeveloper | B-31 |
| Defining Ant Deployment Tasks | B-32 |
| Adding Elements to the Buildfile | B-33 |
| Running Ant on Buildfile Targets | B-34 |
| Creating an External Ant Tool | B-36 |
| Implementing Security in Deployed Applications | B-37 |
| Deployment Testing During Development | B-38 |
| Deployment Testing for Production | B-39 |
| Summary | B-40 |
| Practice Overview: Deploying the Web Application | B-41 |

12

Planning the User Interface

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe the Model-View-Controller design pattern
- Explain the role of the ADF Controller
- Differentiate between bounded and unbounded task flows
- Create task flows
- Define control flows
- Define global navigation
- Create routers for conditional navigation
- Call methods and other task flows
- Convert task flows
- Use validation in the user interface

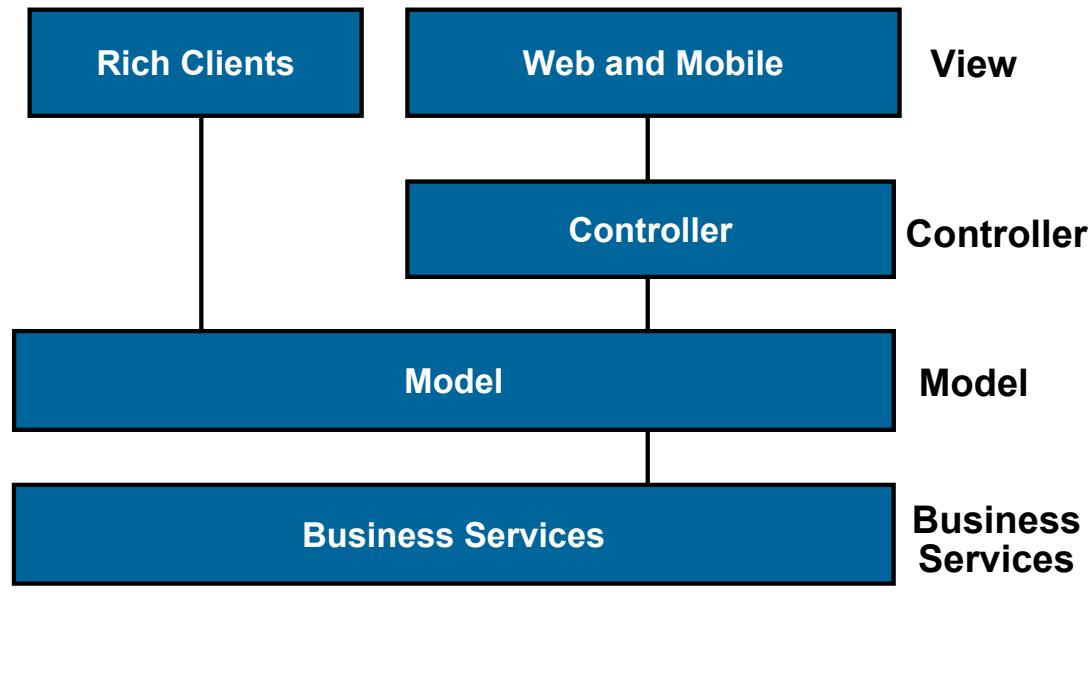


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

The user interface of a Web application usually comprises multiple pages. In this lesson, you learn how to use the ADF Controller to manage page navigation and task flows.

Describing the Model-View-Controller (MVC) Design Pattern



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

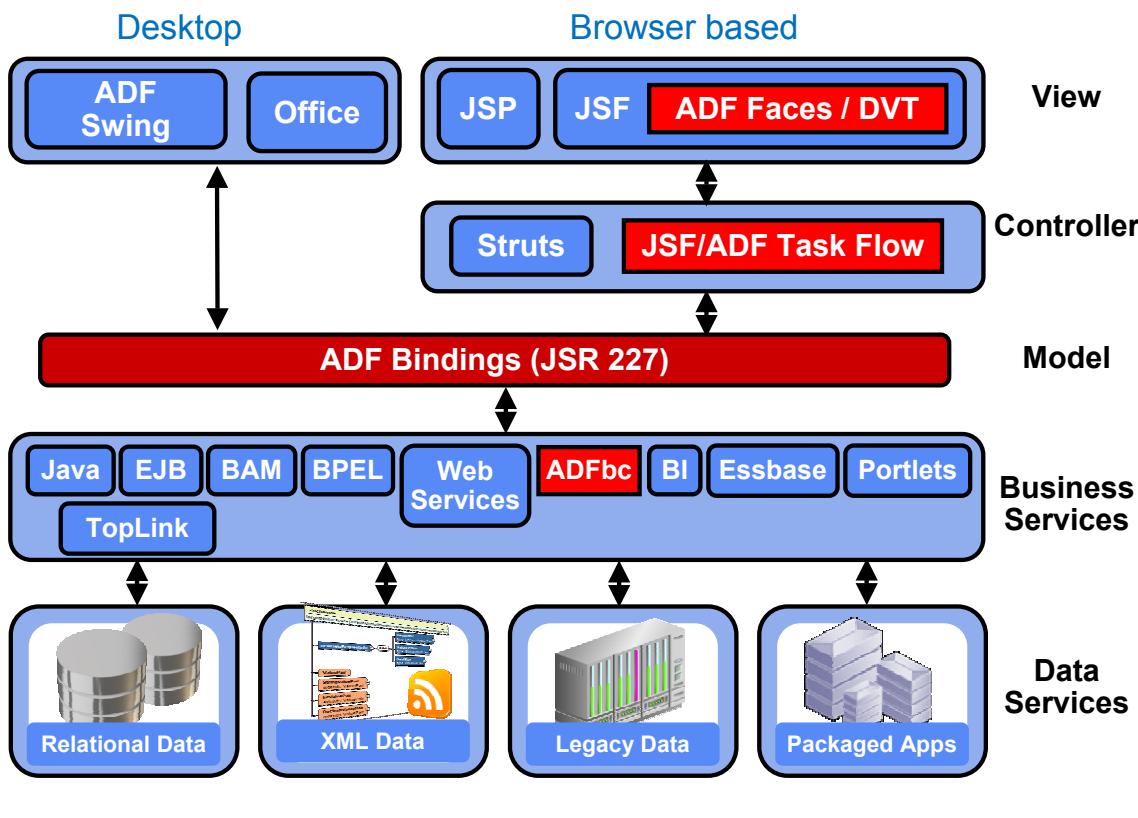
Model-View-Controller Design Pattern

A good practice when developing applications is to employ design patterns. Design patterns are a convenient way of reusing object-oriented concepts between applications and developers. The idea behind design patterns is simple: document and catalog common behavior patterns between objects. Developers can then make use of these patterns rather than re-create them. One of the frequently used design patterns is the Model-View-Controller (MVC) pattern.

In the MVC pattern, the user input, the business logic, and the visual feedback to the user are explicitly separated and handled by three types of objects. Each of these objects is specialized for a particular role in the application:

- The model manages the data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).
- The view manages the presentation of the application output to the user.
- The controller interprets the mouse and keyboard inputs from the user, commanding the model and/or the view to change as appropriate.

Implementing MVC with the ADF Framework



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Implementing MVC with the ADF Framework

Developers often use architectural frameworks to build applications that perform in a standard way. ADF is such a framework, and it implements the MVC design pattern as follows:

- **Data Services:** This can be any data source.
- **Business Services:** This is the back-end data model that interacts with the data source. It may be a set of Java classes or Web services, or can be Enterprise JavaBeans (EJBs), TopLink, or ADF Business Components. This course teaches you to use ADF Business Components as a back-end data model.
- **Model:** ADF data binding utilizes a data binding and data access facility for Java EE that provides a standard for interactions between UI components and methods available on the business services. With this standard data binding, any Java UI rendering technology can declaratively bind to any business service.
- **View:** The user interface can be a rich client with ADF Swing components, or as in this course, can use JavaServer Faces and ADF Faces components.
- **Controller:** Page flow and UI input processing can be implemented in Struts, JSF, or ADF Controller. This course uses the ADF Controller.

The earlier lessons in this course dealt mainly with how to build back-end business services. The remainder of this course explains how to build the front-end user interface (view) and flow control (controller), and how to bind UI components to business services (model).

Characteristics of ADF Task Flows

ADF task flows are logical units of page flows that:

- Offer advantages over JSF page flows
 - An application can be broken into a series of tasks.
 - An application can contain nodes other than pages.
 - You can navigate between activities other than pages.
 - ADF task flows are reusable.
 - ADF task flows have a shared memory scope.
- Can be either unbounded or bounded



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of ADF Task Flows

In the lesson titled “Understanding UI Technologies,” you learned that the ADF Controller enhances the JSF navigation model by providing the concept of task flows.

ADF task flows provide a modular approach for defining control flow in an application. Instead of representing an application as a single large JSF page flow, you can break it up into a collection of reusable task flows. In each task flow, you identify application activities, the work units that must be performed in order for the application to complete. An activity represents a piece of work that can be performed when running the task flow.

The ADF task flow offers the following advantages over standard JSF page flows:

- The application can be broken into a series of tasks, instead of having to represent the application in a single page flow.
- You can use activities other than pages in a task flow.
- You can navigate between activities, not just pages.
- Task flows are modular and reusable within an application or in different applications.
- Shared memory scope enables data to be passed between activities within the task flow.

The ADF controller supports two types of task flows:

- Unbounded task flows
- Bounded task flows

Characteristics of Unbounded ADF Task Flows

Unbounded ADF task flows often serve as the entry point to an application, and have the following characteristics:

- First entry on task flow stack—the outermost task flow
- No well-defined boundary or single point of entry
 - Use an unbounded task flow if your application has multiple points of entry.
- Can be used to define the “top level” flow of an application
- Bookmarkable pages



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of Unbounded ADF Task Flows

Although there can be many bounded task flows in an application, there can be only one unbounded task flow, which is assembled at run time by combining one or more `adfconfig.xml` files. The set of files that is combined to produce the ADF unbounded task flow is referred to as the application’s bootstrap configuration files.

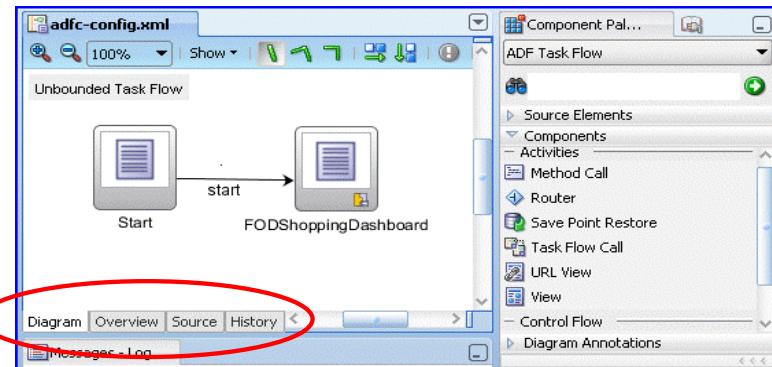
The ADF unbounded task flow represents the outermost task flow of an ADF application. The application’s entry point can be a view within the ADF unbounded task flow.

As a user navigates within an application, a task flow stack is maintained to keep track of the calls from task flow to task flow. An ADF unbounded task flow always logically exists as the first entry on the task flow stack, but would simply be empty if no unbounded task flow is defined.

An unbounded task flow does not contain the well-defined boundary or single entry point of a bounded task flow. You should use an unbounded task flow if your application has multiple points of entry. For example, an end user might enter an application through two different home pages.

Working with Unbounded Task Flows

- The unbounded task flow source file is `adfc-config.xml`.
- The editor contains four tabs to show different views:
 - Diagram
 - Overview
 - Source
 - History



- You can easily test the task flow functionality in the unbounded task flow, and convert to bounded when it is functioning correctly.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Working with Unbounded Task Flows

By default, `adfc-config.xml` is the XML configuration file for an ADF unbounded task flow. You can double-click this file in the Application Navigator to open it in the editor.

There are four tabs at the bottom of the editor that enable you to see different views of the source file for the task flow, all of which remain coordinated as you edit them:

- **Diagram:** Provides a visual view that enables you to drag source elements, components, or diagram notations from the Component Palette to define the task flow
- **Overview:** Contains different panels that enable you to configure the task flow
- **Source:** Displays the XML source code, which you can edit directly
- **History:** Shows a history of changes

A common way of working is to initially create pages and navigation in the unbounded task flow, because that enables you to easily test them in one location. When they are functioning correctly, you can convert portions of the unbounded task flow to a bounded task flow if needed.

A typical application is a combination of an unbounded and one or more bounded task flows. Every Fusion Web application contains an unbounded task flow, even if the unbounded task flow is empty. The application can call bounded task flows from activities in the unbounded one.

Characteristics of Bounded Task Flows

Bounded task flows:

- Are modular blocks of task flow functionality for reuse, with the following characteristics:
 - Single point of entry
 - Well-defined boundary
 - pageFlow Memory scope
 - Declarative transaction management
 - Declarative Back button support
 - Ability to accept input parameters and to return values
 - On-demand metadata loading
 - Fragment reuse through task flow templates
 - Addressable
- Consist of:
 - Activities
 - View
 - Router
 - Control flows between activities



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of Bounded Task Flows

Although there can be only one unbounded task flow in an application, there can be many bounded task flows. You can think of a bounded task flow as a process because it defines a modular block of task flow functionality for reuse. It is the only type of task flow that can be used as a region on a page.

Bounded task flows have the following characteristics:

- Single entry point and zero or more well-defined exit points
- Well-defined boundary with a transaction beginning and end. It consists of its own set of private control flow rules, activities, and managed beans.
- Memory scope (pageFlow scope) variables for passing data between activities within the ADF bounded task flow, with a lifespan that is longer than request, but shorter than the session scope
- Declarative support for transaction management
 - Ability to begin a new transaction upon ADF bounded task flow entry
 - Ability to commit or roll back upon ADF bounded task flow exit

Characteristics of Bounded Task Flows (continued)

- Declarative support for Back button navigation
- Ability to pass input parameters from the ADF bounded task flow caller to store within the pageFlow scope
- Ability to return values back to ADF bounded task flow caller upon exit
- Metadata only loaded on demand
- ADF bounded task flow fragment reuse through task flow templates
- Addressable, in that when you invoke an ADF bounded task flow, you must specify an identifier and the name of the XML document that contains the identifier

Bounded task flows consist of a group of activities and control flows between the activities that enable a user to complete a task. There are different types of task flow activities, including the following:

- **View activity:** Displays a page or page fragment
- **Router activity:** Evaluates a declarative expression to produce a control flow outcome
- **Method call activity:** Invokes application logic (Java method) from within the task flow
- **Task flow call activity:** Calls an ADF bounded task flow
- **Task flow return activity:** Defines an exit point of the task flow
- **Save point restore activity:** Restores an application that supports save for later functionality. This is explained in the lesson titled “Implementing Transactional Capabilities.”
- **URL view activity:** Redirects the view port to any URL addressable resource

Comparing Unbounded and Bounded Task Flows

| Unbounded | Bounded |
|---|--|
| First entry on task flow stack | Added to task flow stack when called |
| No well-defined boundary or single point of entry | Single point of entry, with zero or more exit points |
| Cannot be used as a region on a page | Can be used as region on page with page fragments |
| Does not accept parameters | Can accept parameters and return values |
| Not securable on its own; uses page security | Can be secured separately from pages |
| Cannot manage transactions or save for later | Declarative transaction management, save for later |
| Cannot be called | Must be called to be invoked |
| Can be bookmarked | Not bookmarkable |



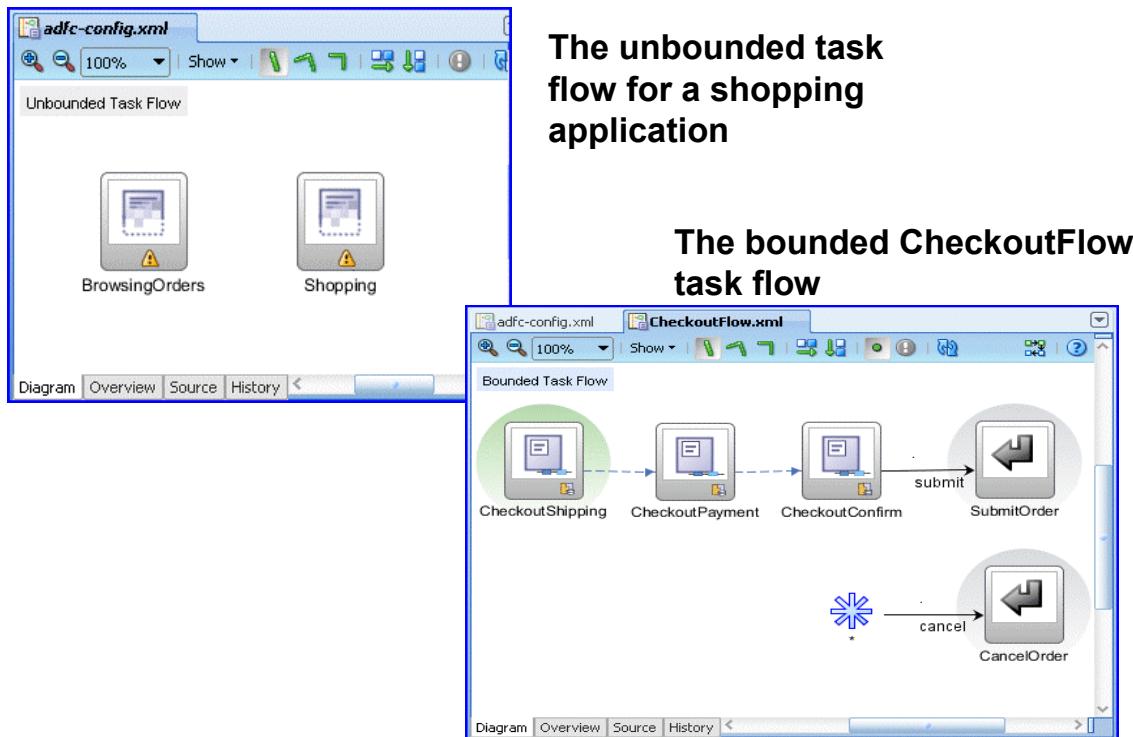
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Comparison of Unbounded and Bounded Task Flows

Unbounded task flow: A set of activities, control flow rules, and managed beans interacting to allow a user to complete a task. An unbounded task flow consists of all activities and control flows in an application that are not included within any bounded task flow.

Bounded task flow: A specialized form of task flow, having a single entry point and one or more exit points. It contains its own set of private control flow rules, activities, and managed beans. An ADF bounded task flow allows reuse, parameters, transaction management, and reentry. It can have zero to many exit points.

Bounded and Unbounded ADF Task Flows: Example



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Bounded and Unbounded ADF Task Flows: Example

The example in the slide shows:

- An unbounded task flow that consists of two pages; there is no entry or exit point in this task flow. It defines the “top level” flow.
- The CheckoutFlow bounded task flow, which is a train with a single entry and two exit points. It also includes a global control flow rule.

A typical application is a combination of an unbounded and one or more bounded task flows. The application can then call bounded task flows from activities within the unbounded task flow. You can include bounded task flows in a top-level (unbounded) task flow diagram. This has various advantages over putting everything in one diagram:

- It breaks the diagram into modules and makes it more readable.
- The individual modules (bounded task flows) make it easier for multiple developers to work on pieces of the application.
- Bounded task flows can be reused by other application developers.

Creating an Unbounded Task Flow

You can create an unbounded task flow by:

- Adding task flow components to `adf-c-config.xml`
- Using the New Gallery
- Converting a bounded task flow



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating an Unbounded Task Flow

Depending on standards in use at your organization, you may never need to create a new unbounded task flow. You may place all unbounded task flow components in the `adf-c-config.xml` file and use that single unbounded task flow for the application.

If you need to create a new unbounded task flow, you can use the New Gallery (Web Tier > JSF > ADF Task Flow). In the Create ADF Task Flow dialog box, you perform the following:

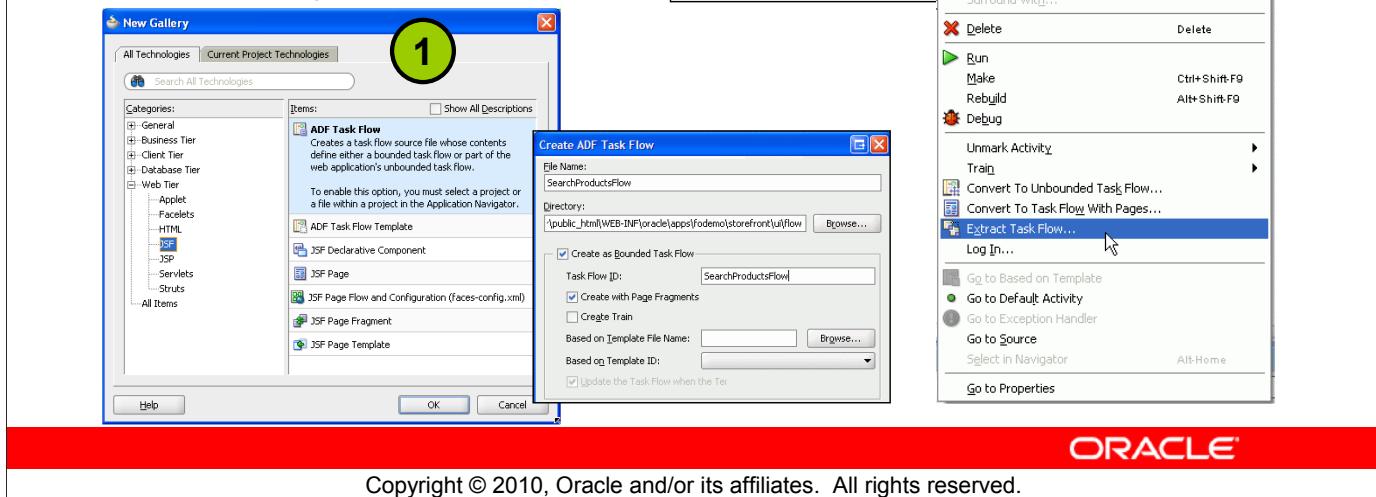
- Give the task flow a name and location.
- Deselect the “Create as Bounded Task Flow” check box. Note that when you do this, the default name, if you have not already changed it, changes to a variation of `adf-c-config.xml`.

A third option, discussed shortly, converts a bounded task flow to an unbounded one.

Creating a Bounded Task Flow

You can create a bounded task flow by doing one of the following:

1. Using the New Gallery
OR
2. Extracting part of an existing task flow



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Creating a Bounded Task Flow

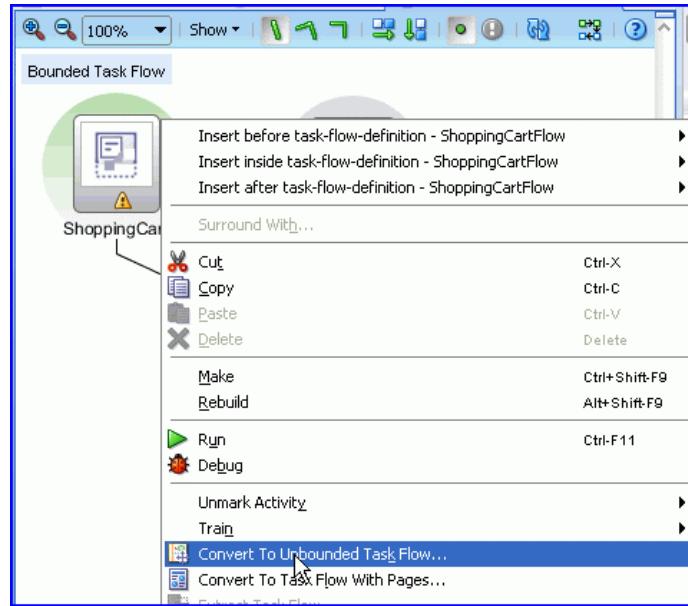
To create a new task flow that is bounded, you can invoke the New Gallery and select Web Tier > JSF > ADF Task Flow. In the Create ADF Task Flow dialog box, you perform the following:

- Give the task flow a name, location, and unique ID.
- Select the “Create as Bounded Task Flow” check box.
- Indicate whether to create it with page fragments. If you want to use page fragments (discussed in the lesson titled “Ensuring Reusability”), you need to be aware that you cannot run a page fragment by itself. You may initially create the bounded task flow with pages and convert it to page fragments after the testing of the task flow is complete.
- Indicate whether to create the task flow as a train, which is a progression of related pages guiding the end user through a series of steps. Each step in a train contains a UI component showing the user’s progress and enabling the user to return to earlier steps.
- Indicate whether to use a template and whether to update the task flow when the template changes.

Another way to create a bounded task flow is to extract part of an existing task flow into its own bounded task flow. To do this, click the task flow diagram and drag out an area to surround the activities that you want to include in the separate task flow. Then right-click one of the activities and select Extract Task Flow from the context menu.

Converting Task Flows

You can convert bounded task flows to unbounded:



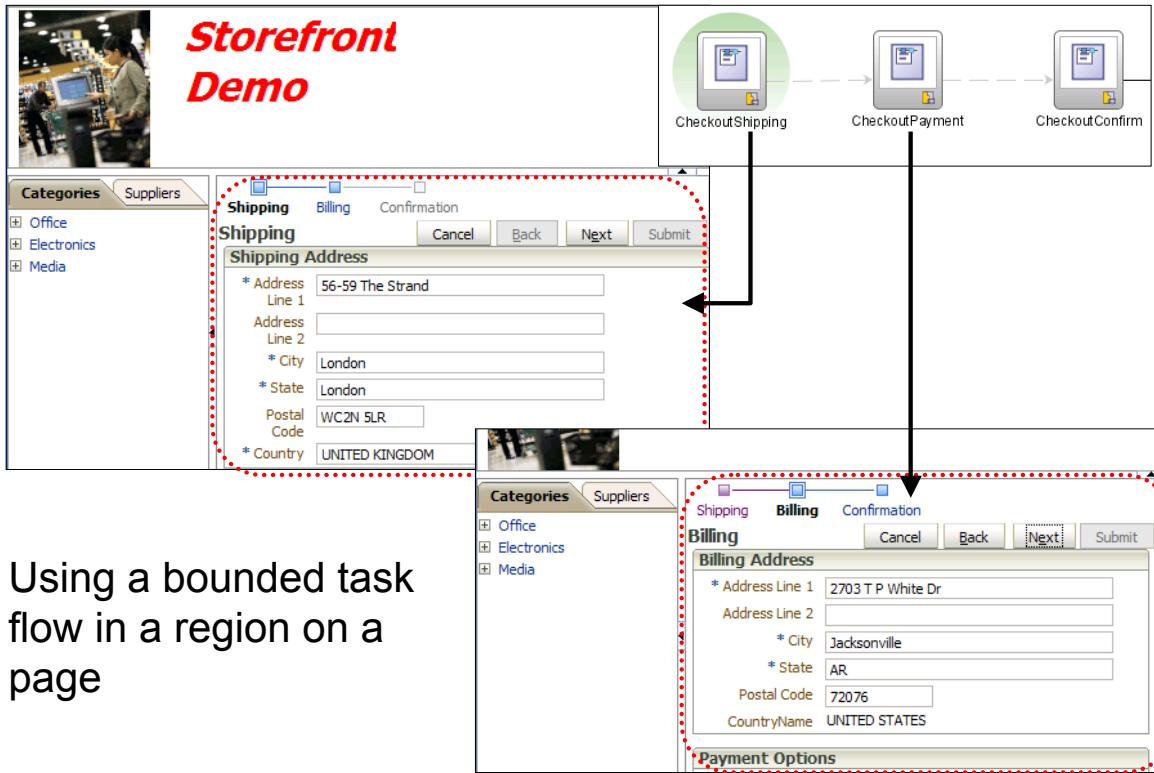
ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Converting Task Flows

JDeveloper makes it easy to convert bounded task flows to unbounded ones. You right-click in the task flow diagram and select “Convert To Unbounded Task Flow.” When you convert a bounded task flow, if it uses page fragments (discussed in the lesson titled “Ensuring Reusability”), then the fragments are converted to pages.

Using a Bounded Task Flow



Using a bounded task flow in a region on a page

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using a Bounded Task Flow

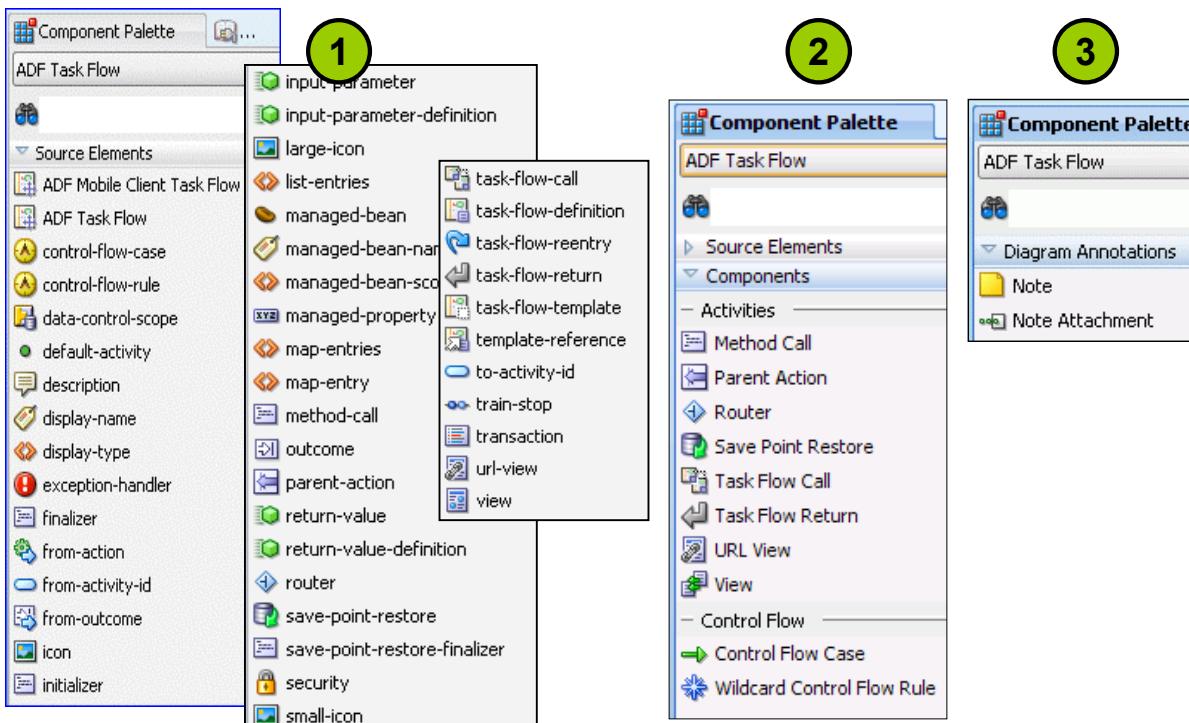
A bounded task flow can be incorporated in an application in various ways:

- As a set of pages and other activities in a larger application flow
- As a region, providing navigation between page fragments on a single containing page
- Within a modal dialog box, launched from a page

You can declaratively manage the transaction of a task flow by specifying whether a bounded task flow starts a new transaction or inherits an existing transaction.

You learn more about regions, page fragments, and transaction control in later lessons.

Using ADF Task Flow Components



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using ADF Task Flow Components

ADF task flow components are divided into three categories in the Component Palette. You can access them by expanding the corresponding categories:

1. Source Elements: (You would not normally use most of these, because it is easier to create them by dragging component elements or by setting properties.)
 - ADF Mobile Client Task Flow: Root element for an ADF Mobile Client task flow source file
 - ADF Task Flow: Root element for an ADF task flow source file and container for unbounded task flow elements
 - bookmark: Identifies a view activity as bookmarkable, so that a user can create a bookmark for it, saving the URL for the page as a bookmark
 - control-flow-case: Defines a possible control flow originating from the view
 - control-flow-rule: Identifies how control passes from one activity to the next
 - data-control-scope: Enables sharing data control instances between task flows
 - default-activity: Specifies the starting activity of a bounded task flow
 - description: Provides a long description of the task flow that is displayed during design time
 - display-name: Provides a short description of the task flow that is displayed during design time

Using ADF Task Flow Components (continued)

- display-type: Specifies whether this task flow call activity runs as an inline pop-up or an external window
- exception-handler: Identifies the activity as the exception handler for the task flow
- finalizer: Specifies code to be invoked when exiting a bounded task flow through a return activity or an exception
- from-action: Limits the application of a control flow rule to outcomes from the specified action method
- from-activity-id: Identifies the activity where a control flow rule originates
- from-outcome: Specifies an outcome value on a control flow case that is matched against a value specified in the action attribute of a UI component
- icon: Specifies the path and file name for an icon to be displayed for the activity during design time
- initializer: Specifies code to be invoked immediately when entering an ADF bounded task flow
- input-parameter: Specifies the name of an input parameter on the bounded task flow
- input-parameter-definition: Specifies the values for an input parameter on the bounded task flow
- large-icon: Specifies the path and file name for a large icon to be displayed for the activity during design time
- list-entries: Defines the values in a list
- managed-bean: Registers a bean to be managed by the configuration of the task flow
- managed-bean-name: Specifies the logical name of a managed bean
- managed-bean-scope: Specifies the memory scope in which a managed bean is stored
- managed-property: Specifies a managed bean property
- map-entries: Defines the values in a map
- map-entry: Defines a key-value pair that comprises an entry in a map
- method-call: Specifies a call to a Method Call activity to invoke application logic
- outcome: Specifies a value to be returned by the router activity if its corresponding expression evaluates to true
- parent-action: Specifies a navigation outcome for a parent task flow
- return-value: Specifies how values are passed back from a called task flow definition to the calling bounded or unbounded task flow
- return-value-definition: Specifies where a calling task flow can obtain return values when a called task flow definition is exited
- router: Specifies an activity to evaluate declarative case logic at run time and route control based on logic specified in an EL expression
- save-point-restore: Is used to restore a previous persistent save point in an application, supporting save for later functionality
- security: Specifies a permission for an ADF bounded task flow
- small-icon: Specifies a small icon to be displayed for an activity or ADF task flow at design time
- task-flow-call: Is used to call an ADF bounded task flow
- task-flow-definition: Specifies properties for a task flow definition

Using ADF Task Flow Components (continued)

- task-flow-reentry: Is used to set the default behavior when an ADF bounded task flow is reentered via browser Back button navigation
 - task-flow-return: Indicates the return from a called bounded task flow
 - task-flow-template: Indicates that the task flow is to be used as a template
 - template-reference: Identifies the ADF task flow template on which a bounded task flow or another task flow template is based
 - to-activity-id: Specifies the activity to navigate to as a result of a control flow case
 - train-stop: Specifies that the activity is one of a sequence of defined steps
 - transaction: Specifies whether a called ADF bounded task flow should join an existing transaction, create a new one, or create a new one only if there is no existing transaction
 - url-view: Is used to redirect the root view port (for example, a browser page) to any URL addressable resource
 - view: Is used to display a JSF page or page fragment (Page fragments are explained in the lesson titled “Ensuring Reusability.”)
2. Components:
- Activities:
 - Method Call: Invokes a Java method from within a task flow
 - Router: Performs conditional routing depending on the outcome of an expression
 - Save Point Restore: Enables restoration of a previous persistent save point in an application that supports save for later functionality (This functionality is described in the lesson titled “Implementing Transactional Capabilities.”)
 - Task Flow Call: Invokes another task flow
 - URL View: Redirects the root view port, such as a browser page, to any URL-addressable resource
 - View: Displays a page or page fragment (Page fragments are explained in the lesson titled “Ensuring Reusability.”)
 - Control Flow:
 - Control Flow Case: Identifies how control passes from one activity to another
 - Wildcard Control Flow Rule: Global navigation rule
3. Diagram Annotations:
- Note: Adds a note to the task flow diagram.
 - Note Attachment: Attaches an exiting note to an activity or control flow case in the diagram

To get more complete descriptions of any of these elements, right-click the component in the Component Palette and select Help from the context menu.

Defining ADF Control Flow Rules

ADF control flow rules:

- Define flow through activities, such as views (pages)
- Are stored in task flow configuration files
- Can be defined by using:
 - The visual editor (Navigation Modeler)
 - The .xml console (Configuration Editor)
 - The .xml file directly
 - The Structure window
- Are invoked by:
 - Command components (button and link)
 - Tabs or breadcrumbs
 - Trains



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining ADF Control Flow Rules

The overall navigation through a JSF application uses a set of rules for choosing the next page to display when, for example, the user clicks a button or link. To define the navigation for your application, you need to define the control flow rules themselves, and then include on the pages the controls that a user can interact with to navigate to another place in the application at run time.

ADF Faces components that can be used to implement navigation features on pages include:

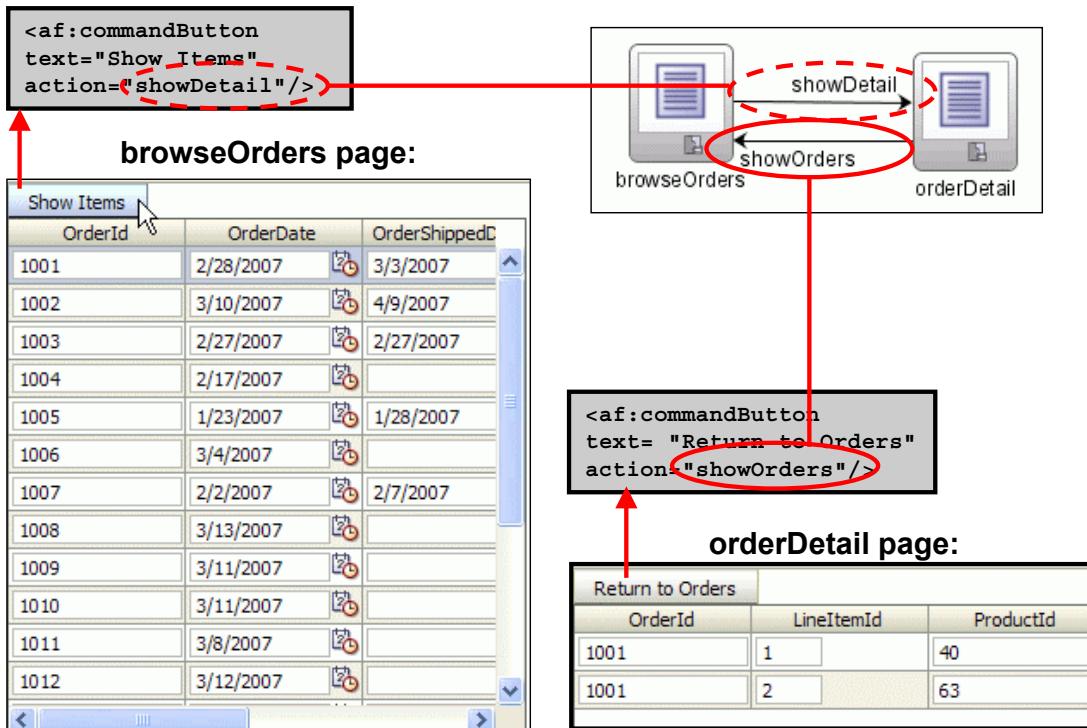
- Buttons and links, which users click to navigate to another place in the application
- Components such as tabs and breadcrumbs
- Train components for walking users through sequential steps of a process

Control flow rule definitions are stored in configuration files (`adfc-config.xml` or a similar `.xml` file). You can define the rules directly in the `.xml` configuration file, or you can use the Navigation Modeler and the Configuration Editor, which offer the following advantages:

- Provide a GUI environment for modeling and editing the navigation between pages
- Enable you to map out your application navigation using a visual diagram
- Update the configuration file for you automatically

Use the Navigation Modeler to initially create control flow rules; use the Configuration Editor to create global or pattern-based rules for multiple pages, create default navigation cases, and edit control flow rules. These tools, along with the Structure window, are explained further in subsequent slides.

Example of ADF Control Flow Rules



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Example of ADF Control Flow Rules

An example of an ADF unbounded task flow showing simple navigation:

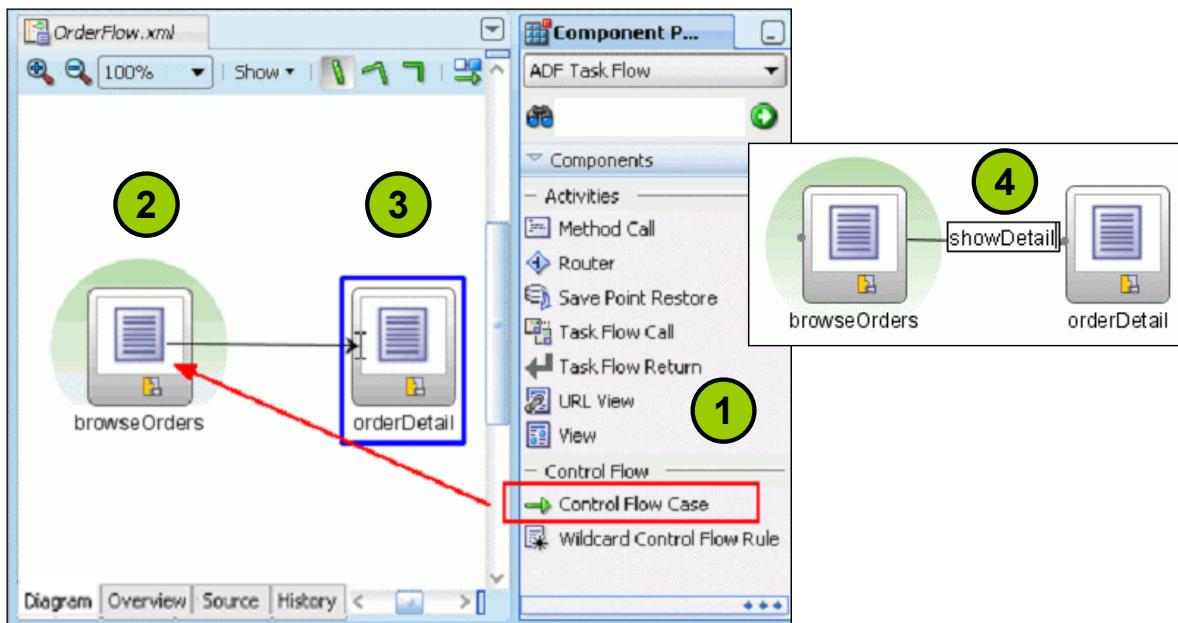
```

<control-flow-rule>
  <from-activity-id>browseOrders</from-activity-id>
  <control-flow-case>
    <from-outcome>showDetail</from-outcome>
    <to-activity-id>orderDetail</to-activity-id>
  </control-flow-case>
</control-flow-rule>
<control-flow-rule>
  <from-activity-id>orderDetail</from-activity-id>
  <control-flow-case>
    <from-outcome>showOrders</from-outcome>
    <to-activity-id>browseOrders</to-activity-id>
  </control-flow-case>
</control-flow-rule>
  
```

A button on each page fragment invokes the appropriate action. For example, the Show Items button on the **browseOrders** page fragment is defined as follows:

```
<af:commandButton text="Show Items" action="showDetail"/>
```

Using the Navigation Modeler to Define Control Flow



ORACLE

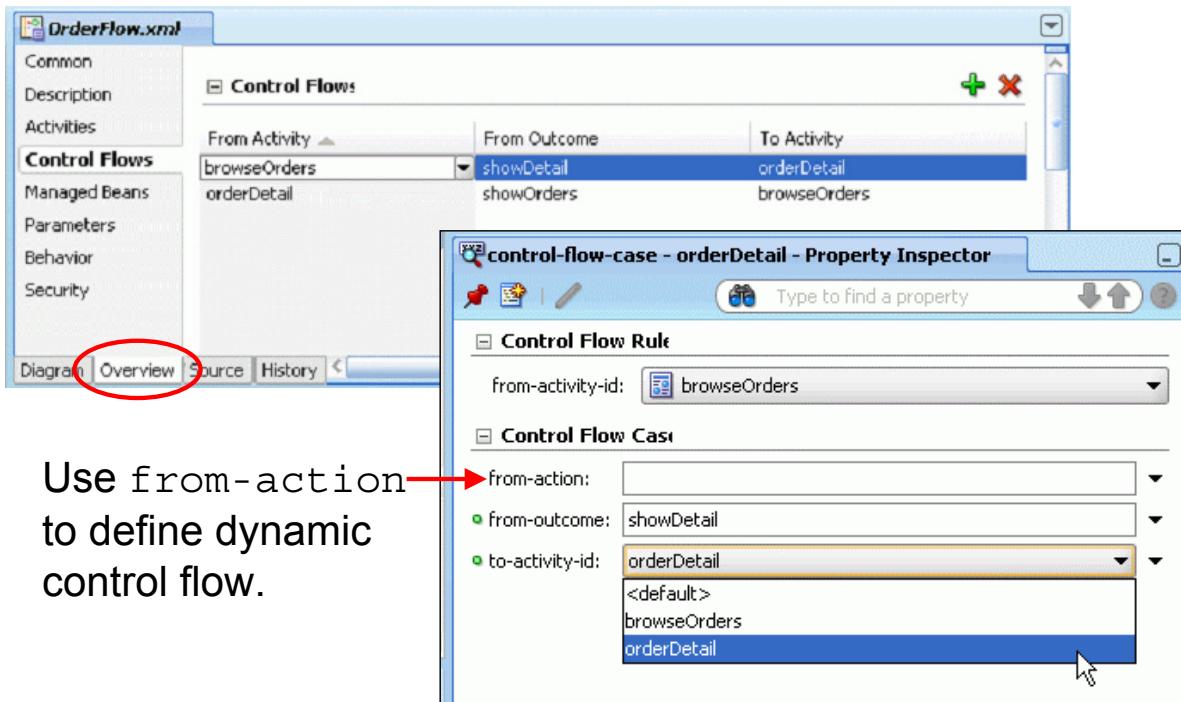
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using the Navigation Modeler to Define Control Flow

To define a control flow rule by using the Navigation Modeler, perform the following steps:

1. Select Control Flow Case in the Component Palette (grouped under Components > Control Flow).
2. Click the source activity (browseOrders in the example in the slide).
3. Click the destination activity (orderDetail in the example).
4. Enter a name for the control flow case.

Using the Configuration Editor to Define Control Flow



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Using the Configuration Editor to Define Control Flow

This console provides a visual way to edit all elements of the .xml configuration file, as opposed to just a navigation model. It is especially useful for creating global or pattern-based rules for multiple pages, creating default navigation cases, and editing control flow rules. From this console, you can edit or remove existing control flow cases or create new ones.

Defining Dynamic Control Flow

If you want the outcome of a control flow to be determined dynamically, you can bind it to a method on a managed bean by setting its `from-action` in the Property Inspector to an expression that evaluates to a managed bean method. The managed bean can execute some application logic and, depending on the results, return an outcome. The returned outcome determines the control flow case that is implemented.

A `from-action` is an EL expression (such as

`#{backing_OrderCreate.cancelButton_action}`) that binds to the managed bean method that should be executed to determine the `from-outcome`. You learn about EL expressions in the lesson titled “Adding Functionality to Pages.”

Editing the .xml File to Define Control Flow

Source code editor shows errors:

```
...  
<control-flow>  
  <from-activity-id>orderDetail</from-activity-id>  
  <control-flow-case>  
    <from-outcome>showOrders</from-outcome>  
    <to-activity-id>browseOrders</to-activity-id>  
  </control-flow-case>  
</control-flow-rule>  
<control-flow-rule>  
  <from-activity-id>orderDetail</from-activity-id>  
  <control-flow-case>  
    <from-outcome>showOrders</from-outcome>  
    <to-activity-id>browseOrders</to-activity-id>  
  </control-flow-case>  
</control-flow-rule>  
...  
Diagram Source Overview History
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

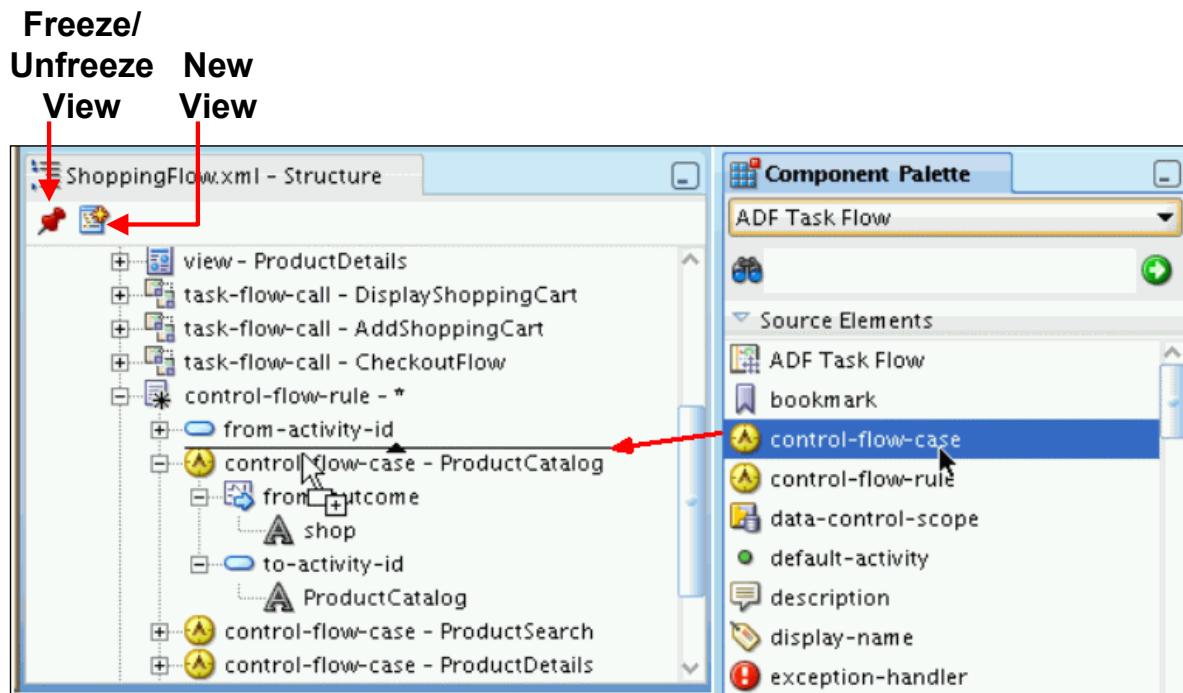
Editing the .xml File to Define Control Flow

You can click the Source tab in the editor to see the XML source. You also can edit the source there.

Errors in XML syntax are underlined with a red squiggly line, and there is also a red box in the right margin. You can see a description of the error when you hold the cursor over the box or over the code that contains the error.

You would probably want to edit the XML directly only in certain cases. For example, you may find that cutting, pasting, and editing blocks of code is easier for you than drawing and labeling elements on the diagram.

Using the Structure Window to Modify a Task Flow



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using the Structure Window

The Structure window offers a structural view of the data in the document currently selected in the active window of those windows that participate in providing structure: the diagrams, the navigators, the editors and viewers, and the Property Inspector.

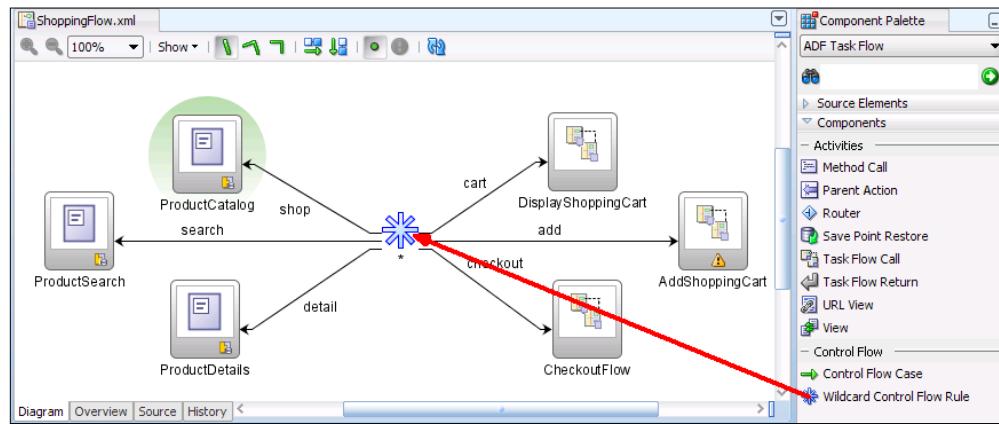
The windows that participate in providing structure also follow selections made in the Structure window. Double-clicking the node for a method in the Structure window, for instance, makes the source editor the active view and takes you directly to the definition for that method.

You can open multiple instances of the Structure window, freezing the contents of any number of them, to compare the structures of different files. You can also switch structure views without changing editors.

It is sometimes easier to select elements in the Structure window and either delete them or drag other elements to them from the Component Palette. So you can use the Structure window not only to view the structure of a document, but to also modify it. For example, the slide shows dragging a control-flow-case tag from the Component Palette to the Structure window to create a new control flow. You can also right-click an element in the Structure window and from the context menu choose to surround it with another element or insert another element inside, before or after it.

Using Wildcards to Define Global Navigation

To define a global (for all pages) control flow rule, drag a wildcard control flow rule to the page:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Wildcards to Define Global Navigation

Most control flow rules specify navigation from one activity to another. However, you can also define global rules that affect all pages by setting the `<from-activity-id>` element to an asterisk, so that the rule can be called from any page. You can create a global rule by dragging a wildcard control flow rule element from the Component Palette, and then define the flow outcomes for the wildcard rule.

The example in the slide shows a wildcard control flow rule that has from-outcome elements to navigate to every page in the task flow. Because of this wildcard rule, any page in the task flow can navigate to any other page. The source XML looks like this:

```

<control-flow-rule>
  <from-activity-id>*</from-activity-id>
  <control-flow-case>
    <from-outcome>shop</from-outcome>
    <to-activity-id>ProductCatalog</to-activity-id>
  </control-flow-case>
  <control-flow-case>
    <from-outcome>search</from-outcome>
    <to-activity-id>ProductSearch</to-activity-id>
  </control-flow-case>

```

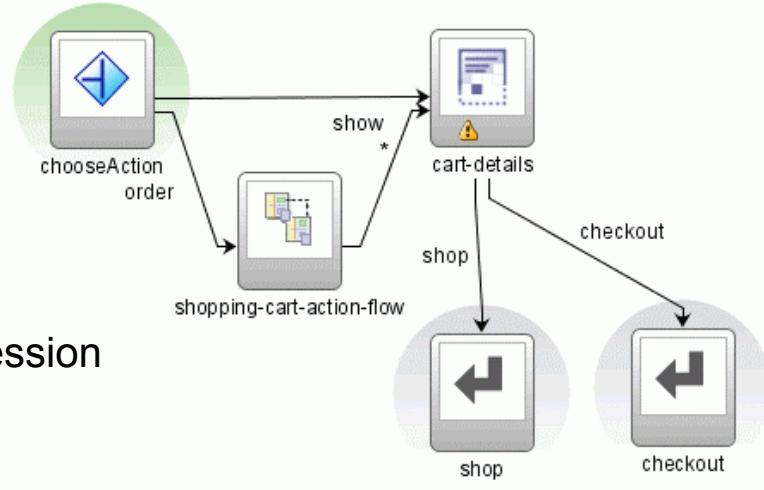
Using Wildcards to Define Global Navigation (continued)

```
<control-flow-case>
    <from-outcome>details</from-outcome>
    <to-activity-id>ProductDetails</to-activity-id>
</control-flow-case>
<control-flow-case>
    <from-outcome>cart</from-outcome>
    <to-activity-id>DisplayShoppingCart</to-activity-id>
</control-flow-case>
<control-flow-case>
    <from-outcome>add</from-outcome>
    <to-activity-id>AddShoppingCart</to-activity-id>
</control-flow-case>
<control-flow-case>
    <from-outcome>checkout</from-outcome>
    <to-activity-id>CheckoutFlow</to-activity-id>
</control-flow-case>
</control-flow-rule>
```

Using Routers for Conditional Navigation

Router activities:

- Use expressions that evaluate to true or false
- Define from-outcomes based on the value of the expression



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Routers for Conditional Navigation

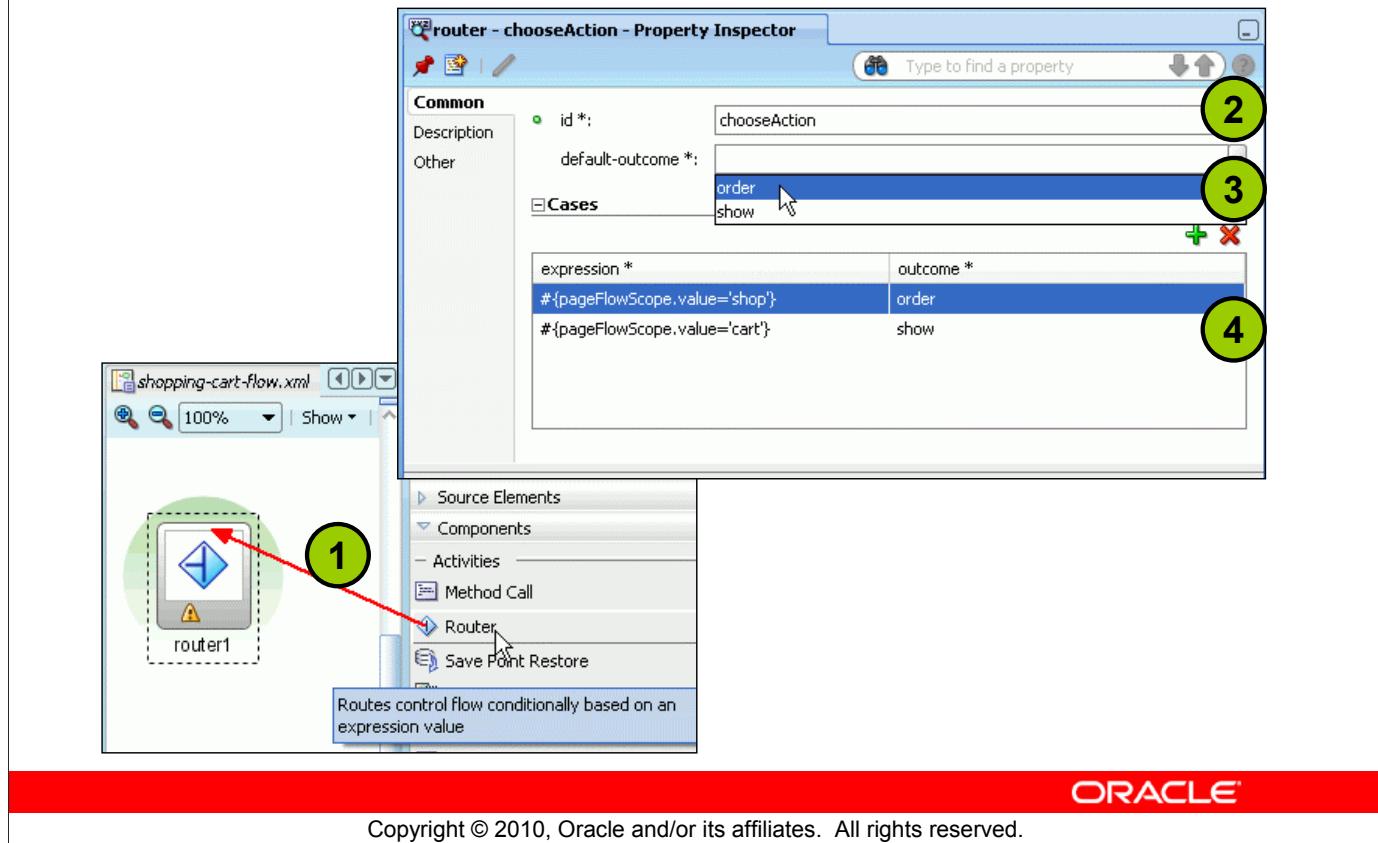
You can use the router activity in a task flow to declaratively route control to activities based on logic specified in an EL expression. As shown in the example in the slide, a router has multiple control flows leading from it to different activities.

Each control flow can correspond to a different router case. Each router case contains the following elements, which are used to choose the activity to which control is next routed.

- **Expression:** An EL expression evaluating to either `true` or `false`, for example,
`# {user.choice == "shop" }`
- **Outcome:** A value returned by the router activity if the EL expression evaluates to `true`, for example, `order`. If the outcome matches a from-outcome on a control flow case, control passes to the activity that the control flow case points to.

For example, suppose you want to base control flow on the content of an input text field on a Start page. You could add a conditional expression for each router case. For each expression, you would specify an expected outcome, which in the example in the slide is `order`. If the expression evaluates to `true`, control passes to the `shopping-cart-action-flow` activity, based on the control flow case from-outcome. If the expression evaluates to `false`, the from-outcome is `show`. Control passes to a page corresponding to the `cart-details` activity.

Defining Router Activities



ORACLE

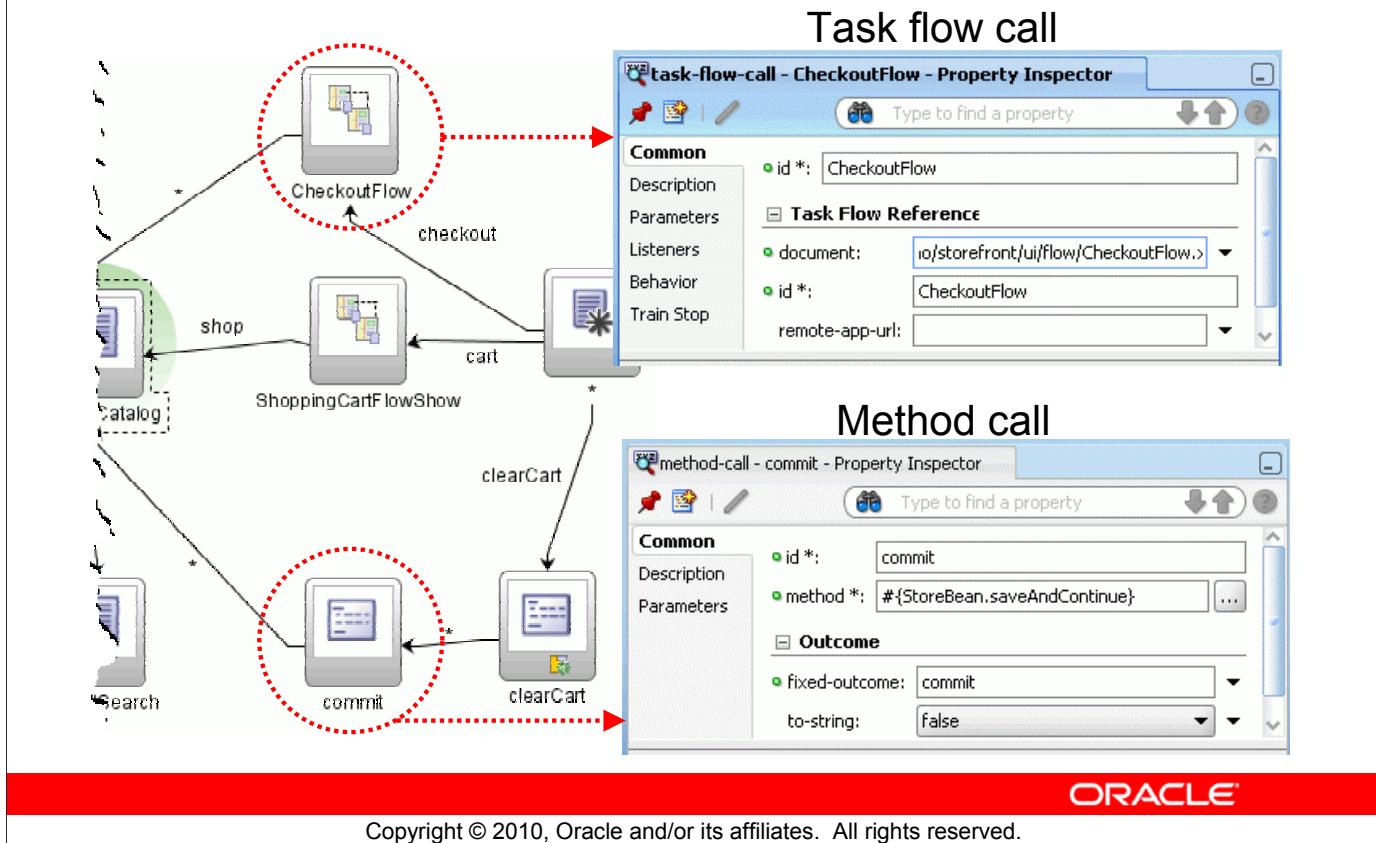
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Router Activities

To define a control flow using the router activity, perform the following steps:

1. In the Component Palette, drag the router activity to the task flow diagram.
2. In the Property Inspector, enter an id. The id is an identifier that is used to reference the router activity within the metadata, for example, chooseAction.
3. In the Property Inspector, enter a default outcome. This outcome is returned if none of the cases for the router activity evaluates to true, or if no cases are specified. The default outcome should specify a case already defined for the router. If no case is specified for the default outcome, an error occurs if the default outcome is returned.
4. In the case section of the Property Inspector, specify values for each of the router's cases. A case is a condition that, when evaluated to true, returns an outcome. For each case, you must enter:
 - expression: An EL expression evaluating to true or false. The expression can reference an input text field in the Start activity.
 - outcome: Returned by the router activity when its corresponding expression evaluates to true. You must account for each outcome with a matching control flow case or a wildcard control flow rule in your task flow diagram.

Calling Methods and Other Task Flows



Calling Methods and Other Task Flows

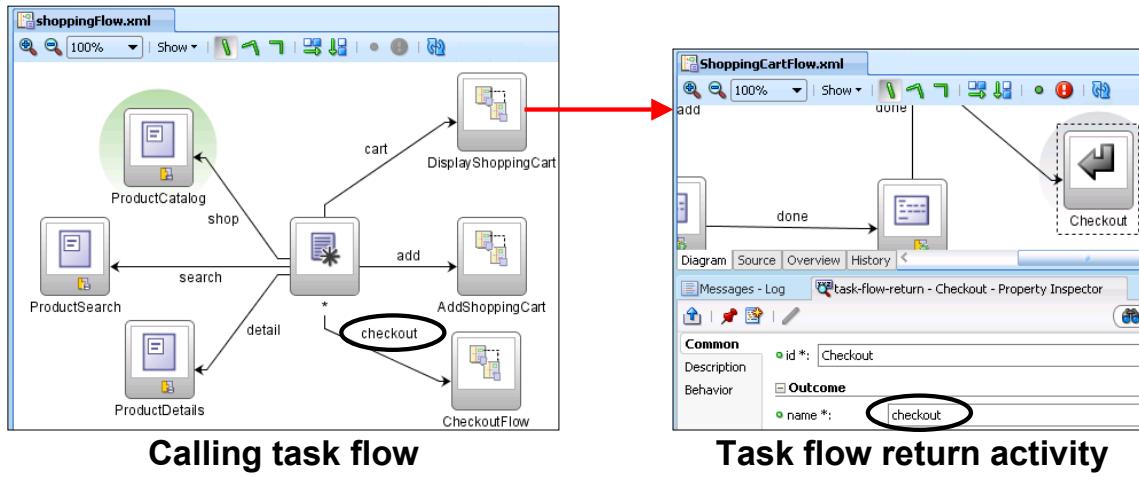
In addition to the view and router activities, you can call methods and task flows from within a task flow by using method call and task flow call activities. You can pass parameters to the called activities, and control returns to the calling task flow when the called activity is completed.

The example in the slide shows part of a ShoppingFlow bounded task flow. It contains a task flow call activity that calls another bounded task flow, CheckoutFlow. It also contains a method call activity, commit, which calls a managed bean method to commit the transaction.

The slide shows the Common panel of the Property Inspector. Note that both the task flow and the method call Property Inspectors also have a Parameters panel, which you use to define parameters to pass to the task flow or method.

Defining a Task Flow Return Activity

When you return from a called task flow, the task flow return activity specifies the outcome that is returned to the caller.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining a Task Flow Return Activity

When you define a task flow return activity, you need to specify the outcome that is returned to the calling task flow. You can have only one outcome per task flow return. The calling flow should have a control flow rule that corresponds to the outcome that you name in the task flow return activity. If a bounded task flow has its own transaction scope, then the return activity will also control whether a commit or rollback will happen when the task flow returns to its caller.

The example in the slide shows the task flow that calls the shopping cart flow. The shopping cart flow has a return activity, depicted on the right of the slide, whose outcome is defined as `checkout`. The calling task flow has a control flow rule named `checkout`, so when the called shopping cart flow is exited via the `Checkout` task flow return activity, navigation in the calling task flow proceeds to the `CheckoutFlow`.

Making View Activities Bookmarkable (or Redirecting)

- Saving a view activity as a bookmark is available only in unbounded task flows.
- You can:
 - Designate in Property Inspector at design time
 - Designate at run time with the `ViewBookmarkable()` method
 - Optionally specify:
 - URL parameters
 - Method to invoke before view is rendered
 - Use the `redirect` option for a view activity instead of making it bookmarkable



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Making View Activities Bookmarkable

You can enable users to easily return to a view activity in an unbounded task flow by designating the activity as bookmarkable. On the Bookmark tab of the Property Inspector, you can set the `bookmark` property to `true`. You can optionally add parameters in the URL Parameters section and add an optional method binding in the converter field for a parameter's value. You can also optionally specify a method in the method field that the ADF Controller should invoke before rendering the view. You can use this method to retrieve additional information based on the parameter values.

Instead of making the view activity bookmarkable, you can choose to use the `redirect` option to create a new browser URL for the view activity. You do this by setting the `redirect` property to `true` on the Common panel of the Property Inspector for the view activity.

Adding UI Code

Managed beans:

- Are configured in `adf-c-config.xml` or other task flow `.xml` file
- Consist of Plain Old Java Objects (POJOs), Lists, and Maps
- Have a no-argument constructor
- Use lazy initialization by JavaServer Faces framework “as needed”



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Adding UI Code

You can add UI code to the task flow by using managed beans. You learn how to code and register managed beans in the lesson titled “Responding to Application Events.” However, a brief mention of managed beans is included in this lesson on task flows because managed beans are specific to task flows and are configured in the task flow’s XML file.

Managed beans are Java classes with a no-argument constructor that you register with the application in task flow `.xml` files. When the JSF application starts up, it parses these files and makes the beans available to be referenced in EL expressions or Java code, enabling access to the beans’ properties and methods.

Whenever a managed bean is referenced for the first time and it does not already exist, the Managed Bean Creation Facility instantiates the bean by calling the default constructor method on the bean. If any properties are also declared, they are populated with the declared default values.

Incorporating Validation into the User Interface

- ADF Faces provides the following types of validation:
 - UI component attributes
 - Default ADF Faces validators
 - Custom ADF Faces validators
- Use ADF Faces validation to provide immediate feedback to users, instead of waiting for commit.
- You should always define equivalent validation at the business component level.

The Oracle logo, consisting of the word "ORACLE" in a white sans-serif font inside a red horizontal bar.

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Incorporating Validation into the User Interface

ADF Faces provides the following types of validation:

- **UI component attributes:** ADF Faces input components have attributes that can be used to validate data. For example, you can set the `required` attribute on a component to specify whether a value must be entered. When set to `true`, the component must have a value.
- **Default ADF Faces validators:** The standard validators supplied by ADF Faces and the JSF reference implementation provide common validation checks, such as validating date ranges and validating the length of entered data.
- **Custom ADF Faces validators:** You can create your own validators, and then select them to be used in conjunction with UI components.

You can use UI validation to provide immediate feedback and meaningful error messages to users. However, UI validation should always have equivalent validation at the business services layer, so that the same validation is applied when the model is exposed in other ways.

Describing the Course Application: UI Functionality

ProductName: Plasma HD Television
Category Name: Audio and Video
Price: £1,999.99

| ProductName | Price | Quantity | LineTotal |
|----------------------|-----------|----------|-----------|
| Ipod Speakers | £89.99 | 1 | £89.99 |
| Plasma HD Television | £1,999.99 | 2 | £3,999.98 |
| Blackberry 8100 | £499.99 | 1 | £499.99 |
| Wii Remote | £71.99 | 1 | £71.99 |
| Tchaikovsky's Ballet | £16.99 | 2 | £33.98 |
| Mini Mac Computer | £799.99 | 1 | £799.99 |

| ProductName | Category Name | Price | Description |
|--------------------------|-----------------|-----------|-------------------|
| Plasma HD Television | Audio and Video | £1,999.99 | The TH-42PX60U 4 |
| PlayStation 2 Video Game | Games | £199.95 | Whether you're a |
| Playstation Portable | Games | £199.99 | The era of no-com |

* Address Line 1: 56-59 The Strand
Address Line 2: _____
* City: London
* State: London
Postal Code: WC2N 5LR
* Country: UNITED KINGDOM

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Describing the Course Application: UI Functionality

The user interface consists of the following main functions:

1. A category tree, where users can click a category to display either subcategories (if the clicked category is a root category) or related products
2. A product search, where users enter query criteria and search for matching products
3. A product detail page, where users can add the product to the cart
4. A shopping cart, where users can update quantities of items ordered, delete items from their carts, or display details about a selected item
5. A checkout process, where users complete a multistep process to complete their orders

There is also another tab that enables users to browse, update, delete, or create suppliers (image 1).

Summary

In this lesson, you should have learned how to:

- Describe the Model-View-Controller design pattern
- Explain the role of the ADF Controller
- Differentiate between bounded and unbounded task flows
- Create task flows
- Define control flows
- Define global navigation
- Create routers for conditional navigation
- Call methods and other task flows
- Convert task flows
- Use validation in the user interface



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 12 Overview: Defining Task Flows

This practice covers the following topics:

- Creating an Unbounded Task Flow
- Creating Bounded Task Flows
- Extracting Part of a Task Flow



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 12 Overview: Defining Task Flows

In the practices for this lesson, you create both unbounded and bounded task flows, along with various types of activities and control flows for each. You also extract part of an unbounded task flow into its own flow, and convert an unbounded task flow to a bounded one.

13

Adding Functionality to Pages

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Internationalize the user interface
- Use component facets
- Enable users to:
 - Select a value from a list
 - Select a date from a calendar
- Display tabular data in tables
- Display hierarchical data in trees
- Display text or media with icons and images
- Define search forms and display results
- Display data graphically



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

This lesson teaches you to use advanced ADF Faces components to add functionality to Web pages.

Internationalization

- Internationalization is the support for multiple locales.
- Localization is the process in which support for a specific locale is added.
- ADF Faces components provide automatic translation (into 28 languages).



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Internationalization

If you want your application to be viewed by users in more than one country, you can configure it to different locales so that it displays the correct language for the language setting of a user's browser. For example, if you know that your application will be viewed in Germany, you can localize your application so that when a user's browser is set to use the German language, the text strings in the application will appear in German.

ADF Faces components provide automatic translation. The resource bundles used for the components' skin (which determines the look and feel as well as the text within it) are translated into 28 languages. If a user sets the browser to use the German language, any text contained within the components is automatically displayed in German.

Resource Bundles

- Resource bundles contain locale-specific strings used in an application.
- Translation strings can include parameters.
- You do not need to load the base resource bundle on each page of your application.

UIResources.properties

```
str.browserTitle=StoreFront Sample Application  
str.about=About this sample  
str.copyright=\u00a9 Oracle Corp, 2007  
str.contact>Contact Us
```

UIResources_de.properties

```
str.browserTitle=StoreFront Beispielanwendung  
str.about=Über dieses Beispiel  
str.copyright=\u00a9 Oracle Corp, 2007  
str.contact=Kontakt
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Resource Bundles

Resource bundles contain the actual text that you have added to the application. You need to create a version of the resource bundle for each locale where your application will be used, and add a `<locale-config>` element to define the default and support locales in the application's `faces-config.xml` file. You also need to add a `<resource-bundle>` element to your application's `faces-config.xml` file to make the resource bundles available to all the pages in your application without using an `<f:loadBundle>` tag on every page. (With JSF1.2, you do not need to load the base resource bundle on each page of your application with the `<f:loadBundle>` tag.)

After you have configured and registered a resource bundle, the Expression Language editor will display the key from the bundle, making it easier to reference the bundle on application pages. The first example in the slide shows part of the `UIResources.properties` file, which displays some of the English-language entries for static text in the StoreFront application. The second example shows the same file named `UIResources_de.properties`, but with the browser set to use the German language.

Steps to Internationalize an Application

1. Create a base resource bundle containing all text strings that are not part of the components themselves.
2. Create a localized resource bundle for each locale supported by the application.
3. Register the locales with the application.
4. Use the resource bundle on your page.

Part of
the
component



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Steps to Internationalize an Application

To internationalize your application, you need to do the following:

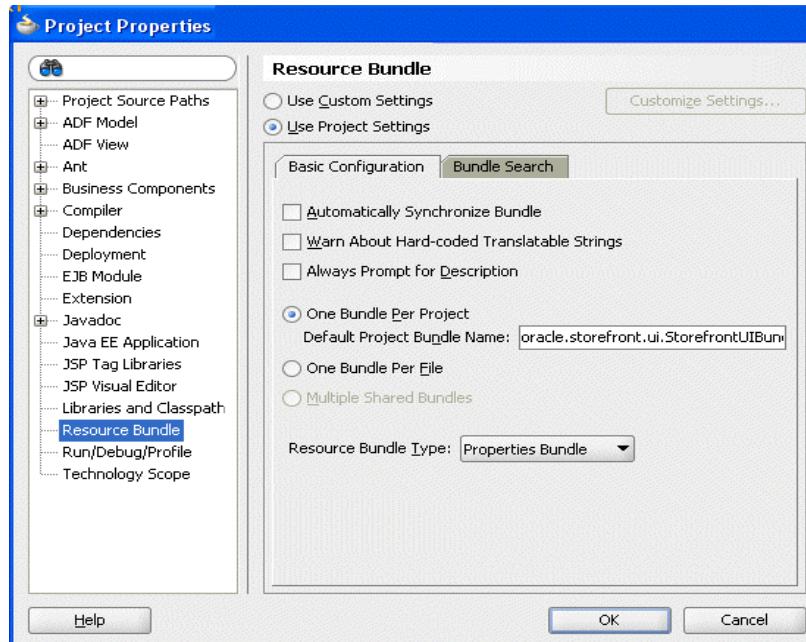
1. Create a base resource bundle that contains all the text strings that are not part of the components themselves. This is a text file with a .properties extension that contains key/value pairs: the key is the name of the text resource and the value is the translation—for example, CANCEL=Cancel. This bundle should be in the default language of the application.
2. Create a localized version of the bundle for each locale supported by the application. You can copy the resource bundle file to another file and append the file name with an underscore and the language code, such as _de for German. For example, if the resource bundle file name is StorefrontUIBundle.properties, you would name the German resource bundle StorefrontUIBundle_de.properties. Then translate the values in the file—for example, CANCEL=Annullieren.
3. Register the locales with the application. You do this on the Application page of faces-config.xml. (Click the Overview tab at the foot of the faces-config.xml file, and then select Application from the list at the top left. You can specify a default locale and add supported locales in the Locale Config area.)

Steps to Internationalize an Application (continued)

4. Use the bundle on your page.
 - a) Note that you need to set your page encoding to be a superset of all supported languages. By default, JDeveloper sets the page encoding to windows-1252. To set the default to a different page encoding, perform the following steps:
 1. Select Tools > Preferences.
 2. Select Environment from the list at the left, if it is not already selected.
 3. Set Encoding to the preferred default.
 - b) Bind all attributes that represent strings of static text on the page to the appropriate key in the resource bundle, using the variable defined in the faces-config.xml file for the <resource-bundle> element.

Automatically Creating a Resource Bundle

Set project properties to create a resource bundle automatically.



ORACLE

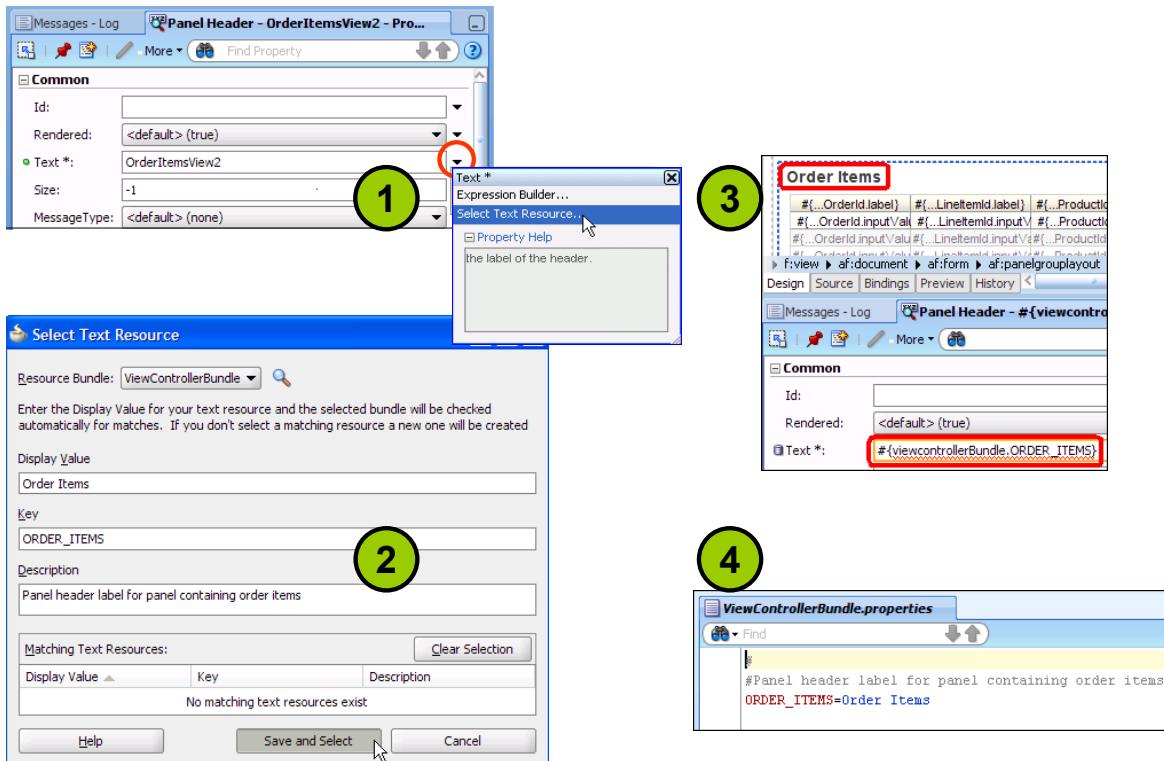
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Automatically Creating a Resource Bundle

You can set project properties to create a resource bundle automatically, either one per project, or one per file. There are three settings that help enforce the use of text resources by developers:

- **Automatically Synchronize Bundle:** Automatically creates text resources on the page's resource bundle when editing UI components in the visual editor
- **Warn About Hard-coded Translatable Strings:** Displays a message when a UI component that is being edited in the visual editor has an associated translatable string
- **Always Prompt for Description:** Displays a message prompting a text resource when editing UI components in the visual editor

Automatically Creating a Text Resource



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Automatically Creating a Text Resource

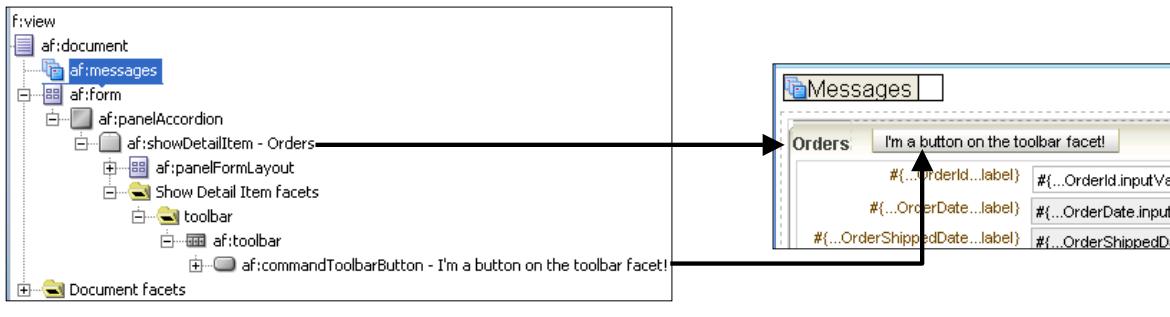
You do not need to manually edit the resource bundle file to create a text resource (name–value pair). The JDeveloper IDE provides a simpler way to do this.

1. In the Property Inspector, click the arrow to the right of the text attribute for which you want to enter a text resource and select Select Text Resource. The first time you do this, the appropriate resource bundle file is created if it does not already exist.
2. In the Select Text Resource dialog box, you can either select an existing text resource or create a new one.
 - a. Enter a Display Value (the text that should be displayed in the UI and that is translatable).
 - b. Enter a key (defaults based on the Display Value, but you can change it).
 - c. Enter a Description (how the text resource is to be used).
 - d. Click “Save and Select” (or Select when choosing an existing text resource).
3. The UI reflects the string, whereas the property shows the EL expression that refers to the text resource.
4. The text resource is added to the resource bundle.

Using Component Facets

Facets are:

- Placeholders for subcomponents
- Similar to detail elements
- Used to specify subordinate elements such as toolbars, headers, or footers
- Displayed with the component



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Facets

When you begin to use components, you see in the Structure window that many of them contain facets. A facet is a placeholder where a developer can add content.

For example, the `af : showDetailItem` component has a facet called `toolbar`. As the name suggests, the facet is intended as a placeholder for a toolbar, although you could place any content there.

In the example in the slide, the application developer has added a `toolbar` component with a `toolbarButton` to the `toolbar` facet of `showDetailItem` and set the button's `text` property value to "I'm a button on the toolbar facet!" At run time, that text is displayed at the place designated for the `toolbar` facet on the `showDetailItem` component.

It is important to note that a facet can have only one direct child. If the developer adds more buttons to the `toolbar` facet, they must be added to the direct child of the facet, which is the `toolbar` component.

Using ADF Faces Rich Client Components

This lesson discusses the following types of components:

- Input
 - List
 - Dates
- Table and tree
- Output
- Query
- Data visualization



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

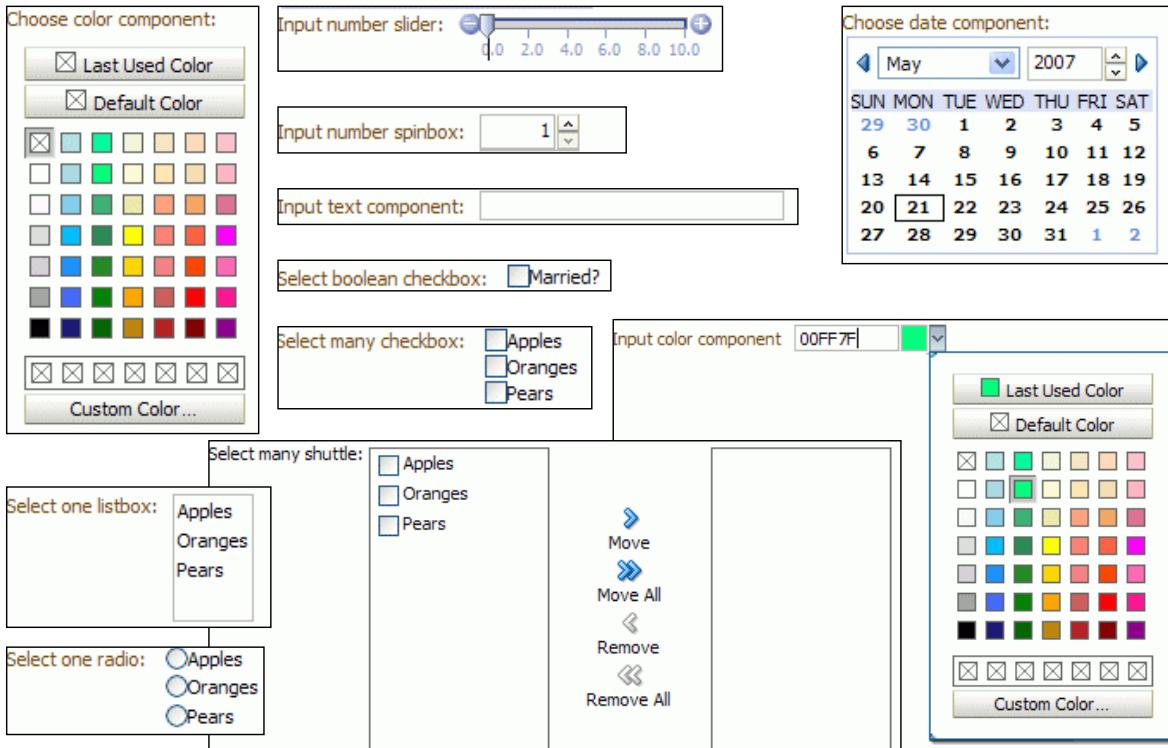
Using ADF Faces Rich Client Components

You have already learned how to use a few basic ADF Faces components on a page. There are over a hundred components from which to choose, many of which are discussed in this and later lessons. You can obtain a demo of all the components at http://www.oracle.com/technology/products/adf/adffaces/11/doc/demo/adf_faces_rc_demo.html.

These components can be categorized as follows:

- **Input components:** For accepting user input
- **Table and tree components:** For displaying structured data
- **Output components:** For displaying text, icons, and images, and for playing audio and video clips on application pages
- **Query components:** For constructing search forms
- **Data visualization components:** For utilizing graphical and tabular capabilities for analyzing data
- **Navigation components:** For implementing navigation features on pages (described in the lesson titled “Implementing Navigation on Pages”)
- **Layout components:** For arranging data on a page (described in the lesson titled “Achieving the Required Layout”)

Using ADF Faces Input Components



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Using ADF Faces Input Components

Input components accept user input in a variety of formats. The most common formats are text, numbers, date, and selection lists. The entered values or selections can be validated and converted before they are processed further. These values may be data that will eventually be committed to a database or search criteria values in a search form that will be used to construct a query statement.

ADF Faces input components provide a variety of UI devices for entering data, such as:

- **Text:** The basic input component for text is `inputText`.
- **Numbers:** They can be entered by using a number slider (`inputNumberSlider`), or a number spinbox (`inputNumberSpinbox`), or for entering the endpoints of a range of numbers, a horizontal or a vertical range slider (`inputRangeSlider`).
- **Dates:** Users can select the date from a calendar (`chooseDate`), and can have the date entered into an input field (`inputDate`).
- **Colors:** Users can select the color from a color chooser (`chooseColor`), and can have the color name entered into an input field (`inputColor`).

All input components (except `af:selectItem`) have a `changed` attribute that when set to `true` (the default is `false`) would enable a change indicator icon to be displayed upon changes in the Value field. This feature is an easy way to enable users to see which input values have changed.

Using ADF Faces Input Components (continued)

- Selecting one or more values from lists: The selection features that you can use include the following:
 - Select a value by using a check box (`selectBooleanCheckbox`) or select an option button (`selectOneRadio` or `selectBooleanRadio`).
 - Select a single value from a list or menu (`inputListOfValues` or `selectOneChoice`).
 - Select from a list that includes a search and select dialog box (`inputComboboxListOfValues`).
 - Select multiple values from a list or menu (`selectManyCheckbox`, `selectManyChoice`, or `selectManyListbox`).
 - Select multiple values from a list of values by shuttling values between two lists, adding and removing values from the target list (`selectManyShuttle` or `selectOrderShuttle`).
- The ADF Faces Rich Text Editor component provides users with the ability to edit rich text. Text formatting features supported include:
 - Text and background colors
 - Font name, size, and weight
 - Text decoration, such as underline and strikethrough
 - Ordered and unordered lists
 - Text justification (left, right, center, or full)
 - Undo and redo
 - Link and unlink
 - Clear styling
 - Undo and redo
 - Subscript and superscript
- You can provide features for users to upload files (`inputFile`).

Input components are usually contained within a `form` or `subform` component. A form is a component that serves as a container for other components. When you add input components, you need to embed those components within a form structure. For example, if you add an `af:inputNumberSlider` to a page, it must be enclosed or be a child component within an `af:form`.

Within a form, you can also add an `af:subform`, which represents a portion of the page that can be independently submitted. The contents of a subform are validated (or otherwise, processed) only if a component inside the subform is responsible for submitting the page. This allows for comparatively fine-grained control of which components will be validated and pushed into the model without the compromises of using entirely separate form elements.

Oracle recommends the use of a single `<af:form>` per page, along with using `<af:subform>` instead of multiple forms. Multiple forms require multiple copies of page state, and user edits in forms that are not submitted are always lost. When a page using subforms is submitted, page state is only written once, and all user edits are preserved.

You can include a button to reset the content of all input items to their default values (`af:resetButton`).

To see descriptions of how to use these input components, you can place the cursor over the name of the component in the Component Palette.

Defining a List

You can define a list:

- At the model layer
- In the UI



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining a List

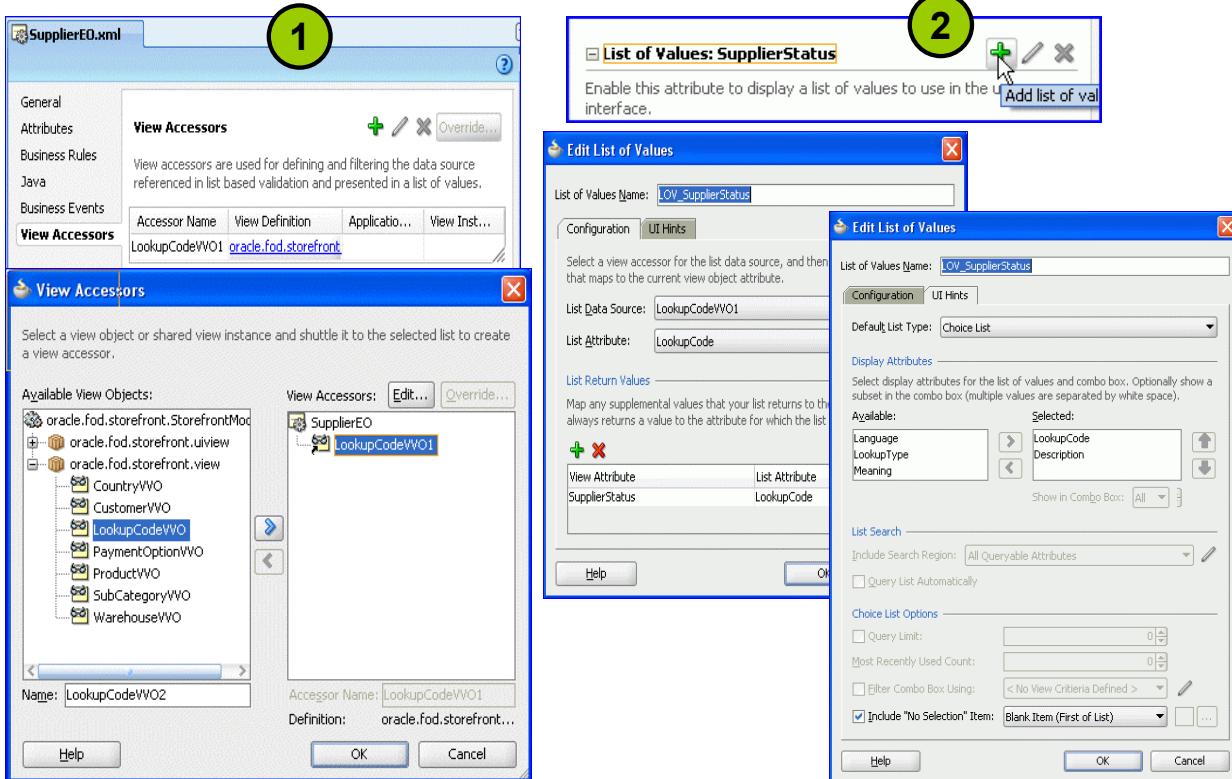
You can define a list in one of two ways:

- At the model layer: You can define lists of values (LOVs) on view object attributes declaratively, including the ability to:
 - Configure the display style (including both inline drop-down lists for shorter lists and pop-up windows for longer lists)
 - Filter the choices in the list based on user input
 - Autofill the attribute value if user input identifies a unique match
 - Automatically use other attribute values in the current row to parameterize “cascading” lists
- In the UI, by binding a selection component to model data

When you define lists at the model layer, they appear in a consistent manner regardless of the type of UI that you use. In addition, you can test them with the Business Components Browser.

Because defining lists at the model layer is a best practice, that is the method discussed in this lesson.

Defining Lists at the Model Layer



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Lists at the Model Layer

For a consistent UI, it is preferable to specify and define lists in the model layer. You can do this by performing the following steps:

1. Define a view accessor on the entity object to reference the data that composes the list. (This is the same view accessor that would also be used by a Key Exists validator or a List validator.)
2. For the view object attribute where you want to use the list for input, define List of Values properties:
 - a. Select the attribute on the Attribute tab of the view object.
 - b. In the List of Values section of the Attribute tab, click Add.
 - c. Under List Data Source, select the view accessor to use and the attribute whose value is to be returned to this attribute.
 - d. Under List Return Values, optionally add any other attribute values that you want to return to the base view object.
 - e. Click Edit List UI Hints to specify the list type and display options.

Changing the type of list defined in the model does not automatically change any list components that already exist on pages. To change the type of list used for an existing component on a page, you need to delete the component, change the list type that is defined in the model, and drag the data element to the page again.

Selecting a Value from a List

Input select components:

- **Item:** af:selectItem
- **List of values:** af:inputListOfValues, af:selectOneListBox, af:selectManyListbox
- **Combo Box:** af:inputComboboxListOfValues
- **Check Box:** af:selectBooleanCheckbox, af:selectManyCheckbox
- **Menu:** af:selectOneChoice, af:selectManyChoice
- **Radio group:** af:selectBooleanRadio
- **Radio item:** af:selectOneRadio
- **Shuttles:** af:selectManyShuttle, af:selectOrderShuttle



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Selecting a Value from a List

For input fields where the set of allowable values is held in a managed bean or in your application model, you can allow users to select the value they want from a provided list of values. ADF Faces provides several selection components:

- The af:selectItem component represents a single item that users can select from a list, choice, option buttons, or shuttle ADF control.
- The af:inputListOfValues component provides users with a search dialog box from which they can select a value to enter. This component enables a user to select from a large list of values to populate the LOV field (and possibly other fields) on a page. It is typically used in situations where the list of values is too large to display in a drop-down list. In addition to rendering an af:inputText component, the af:inputListOfValues component also renders a Search icon. Clicking the icon launches a “Search and Select” dialog box that enables the user to find an option that goes into the LOV field on the base page.
- The af:inputComboboxListOfValues component gives the user two different ways to select an item to input: from a drop-down list, or by searching a list.

Selecting a Date

Date attributes are automatically created as `af:inputDate` components:

```
<af:inputDate value="#{bindings.OrderDate.inputValue}"
    label="#{bindings.OrderDate.label}"
    required="#{bindings.OrderDate.mandatory}">
<af:convertDateTime pattern="#{bindings.OrderDate.format}" />
</af:inputDate>
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Selecting a Date

As with other components, you can create a calendar component by dragging from the Component Palette or the Data Controls panel. However, when a default component is created for a date attribute, it is automatically created as an `af:inputDate` component. This component presents a text input field for entering dates and a button for picking dates from a calendar. The default date format is the short date format appropriate for the current locale. For example, in U.S. English, the format is `mm/dd/yy`. However, you can override the format using `af:convertDateTime`. The same component can also include the selection of time if the column is of the `timeStamp` type.

When you use the `af:inputDate` component, an `af:convertDateTime` subcomponent is included. This subcomponent converts a `String` into `java.util.Date`, and vice versa, based on the pattern and style set.

You can set the following attributes on the `af:inputDate` component, either programmatically or by using the Property Inspector:

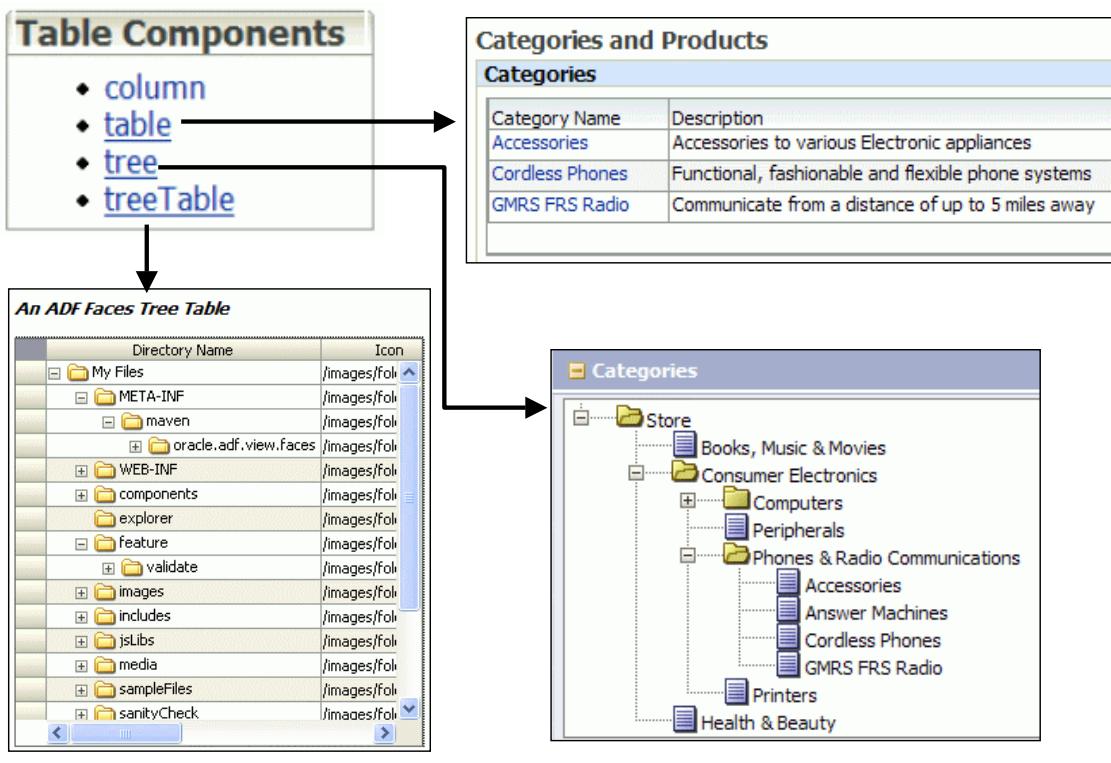
- `label`: Specify a label for the component.
- `chooseId`: Specify the ID of the `af:chooseDate` component, which can be used to pick the Date value for this `af:inputDate`.
- `value`: Specify the initial value of the component.

Selecting a Date (continued)

- `minValue`: Specify the minimum value allowed for the date value. When set to a fixed value on a tag, this is parsed as an ISO 8601 date. ISO 8601 dates are of the format yyyy-MM-dd (for example, 2002-02-15). All other uses require `java.util.Date` objects.
- `maxValue`: Specify the maximum value allowed for the date value. When set to a fixed value on a tag, this is parsed as an ISO 8601 date. ISO 8601 dates are of the format yyyy-MM-dd (for example, 2002-02-15). All other uses require `java.util.Date` objects.
- `disabledDays` Specify a binding to an implementation of the `org.apache.myfaces.trinidad.model.DateListProvider` interface. The `getDateList` method should generate a `List` of individual `java.util.Date` objects, which should be rendered as disabled. The dates must be in the context of the given base calendar.
Note: This binding requires periodic round-trips. If you just want to disable certain weekdays (for example, Saturday and Sunday), use the `disabledDaysOfWeek` attribute.
- `disabledDaysOfWeek`: Specify a space-delimited list of weekdays that should be rendered as disabled in every week. The list should consist of one or more of the following abbreviations: sun, mon, tue, wed, thu, fri, sat. By default, all days are enabled.
- `disabledMonths`: Specify a space-delimited list of months that should be rendered as disabled in every year. The list should consist of one or more of the following abbreviations: jan, feb, mar, apr, may, jun, jul, aug, sep, oct, nov, dec. By default, all months are enabled.

Note: ADF Faces pop-up components, such as dialog boxes, pop-up windows (`af:popup`), LOVs, and calendar components, are not normal browser windows, but rather DHTML overlays that current pop-up blocker technology does not detect. They are, therefore, not affected by pop-up blockers.

Using ADF Faces Table and Tree Components



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using ADF Faces Table and Tree Components

You can easily add a databound table to a page by using the `af:table` component to display structured data as a formatted table. This is done for you automatically when you drag a view object instance from the Data Controls panel and choose to create it as an ADF table.

There are two additional types of components for displaying rows of data:

- `af:tree`: Is used to display hierarchical data; the nested levels of data are indented, and the user can expand and collapse the nodes
- `af:treeTable`: Combines the features of a table and a tree to present hierarchical data with indenting in the same way as the `af:tree` component, but within a table, offering the column heading, column banding, and other features of a table. Table components include a variety of features for presenting tables, such as grid lines, banding, and grouping of columns with headers spanning several columns.

Using Tables

- ADF Faces table component adds to JSF table by providing:

The table data in both screenshots is as follows:

| | SupplierId | SupplierName | SupplierStatus | Email |
|-----|--------------|--------------|-----------------------|-------|
| 100 | Stuffz | ACTIVE | contact@stuffz.exan | |
| 110 | Wally's Mart | ACTIVE | contact@wallymart.e | |
| 120 | MoreStuffz | ACTIVE | contact@morestuffz. | |
| 200 | BuyMyJunk | ACTIVE | contact@buymyjunk. | |
| 300 | ABC Plumbing | ACTIVE | contact@abcplumbing.e | |

- There are many formatting options for ADF tables.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Tables

The ADF Faces table component `af:table` is similar to the standard JSF component but provides additional features, including the ability to sort columns by clicking the column header, filtering data, reordering columns, and selecting a single row or multiple rows.

Table attributes enable you to specify the look and feel of the table, whether users will be able to select rows, or reorder columns, as well as identifying the data to be displayed.

The immediate children of a table component must all be column components, `af:column`. Each visible column component is displayed as a separate column in the table. For sets of columns that are related in some way, you can group them into column groups, with one heading spanning several columns.

Attributes of each column specify details of the column, such as the alignment of values, whether values are allowed to wrap within cells, and whether the column can be sorted, and if so, on which property it is to be sorted. To specify the headers and footers for columns, header and footer facets are available.

Within a column, both ADF Faces and standard JSF tags can be used for displaying the data.

Rendering (Stamping) the Table Data

Each child component of a column is stamped once per row, so no embedded stamped components:

```
<af:table var="row" value="#{myBean.allEmployees}">
  <af:column>
    <af:outputText value="#{row.firstname}" />
  </af:column>
  <af:column>
    <af:outputText value="#{row.lastname}" />
  </af:column>
  <af:table var="row" value="#{myBean.EmployeeManager}">
    <af:column>
      <af:outputText value="#{row.firstname}" />
    </af:column>
    <af:column>
      <af:outputText value="#{row.lastname}" />
    </af:column>
  </af:table>
</af:table>
```



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Rendering (Stamping) the Table Data

The immediate children of an `af:table` component must all be `af:column` components. Each visible `af:column` component creates a separate column in the table.

The child components of each column display the data for each row in that column. The column does not create child components per row; instead, each child is repeatedly rendered (stamped) once per row.

Because of this stamping behavior, some components may not work inside the table. Any component that is pure output, with no behavior, will work without problems. ADF Faces input components work properly. Components that themselves support stamping are not supported, such as tables within a table.

As each row is stamped, the data for the current row is copied into a property that can be addressed using an EL expression. You specify the name to use for this property in the `var` property on the table. After the table has completed rendering, this property is removed or reverted to its previous value.

In the example in the slide, the data for each row is referenced by using the variable `row`, which identifies the data to be displayed in the table. Each column displays the data for each row as defined by a particular attribute.

Setting Table Attributes

| Type | Description | Attribute(s) | Setting(s) |
|---|--------------------------------------|--|---|
| Grid Lines (table attribute) | Lines separating cells | verticalGridVisible horizontalGridVisible | true or false |
| Banding (table attribute) | Alternating background colors | rowBandingInterval columnBandingInterval | Number of rows or columns you want to be in each band |
| Column or row headers | Labels for columns or rows | header facet or headerText attribute | Text to display in header |
| | | rowHeader column attribute | true or false |
| Column groups | Common label for group of columns | To group columns, nest af:column tags, using the outer af:column tag as the group, and the inner tags as the columns within the group. | |
| Column formatting | Align, wrap, width, height | Align nowrap width, height | start, end, left, right, center, true or false number or % |
| Row selection | Selection of single or multiple rows | rowSelection | single or multiple |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Setting Table Attributes

There are many attributes of the af :table component, a few of which are shown in the slide and in the following examples:

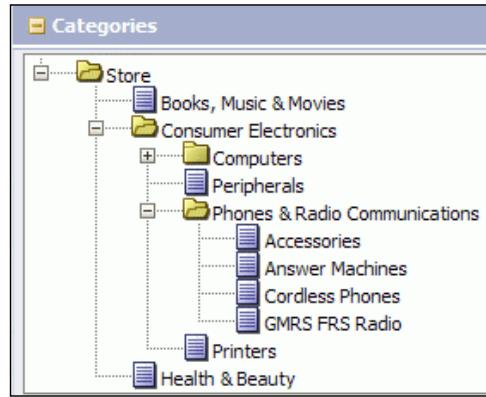
- Grid lines:
`<af:table verticalGridVisible="false" ...>`
- Banding:
`<af:table columnBandingInterval="1" ...>`
- Column headers (one set with header facet and one set with the headerText attribute):
`<af:table>
 <af:column>...
 <f:facet name="header">
 <af:outputText value="First name"/>
 </f:facet>
 ...
 </af:column>
 <af:column headerText="Last name">
 ...
 </af:column>
</af:table>`

Setting Table Attributes (continued)

- Row headers: The first column of a table can be rendered as a column of headers for the table rows. To use row headers, include the data for the row header text as the first column of the table and set the `rowHeader` attribute of the column to `true`.
- Column groups: Shows two name columns grouped into a Name column group. The individual columns retain their own column headers First and Last:

```
<af:table var="row">
    <af:column headerText="Name">
        <af:column headerText="First">
            <af:outputText value="#{row.col1}" />
        </af:column>
        <af:column headerText="Last">
            <af:outputText value="#{row.col2}" />
        </af:column>
    </af:column>
</af:table>
```

Using Trees



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Trees

The ADF Faces tree component `af:tree` displays hierarchical data, such as organization charts or hierarchical directory structures. In data of these types, there may be a series of top-level nodes, and each element in the structure may expand to contain other elements. In the shopping example shown in the slide, each element, that is, each category, in the hierarchy may have any number of child elements (products). In addition, several parent elements may share the same child elements.

The `af:tree` component supports multiple root elements. It displays the data in a form that represents the structure, with each element indented to the appropriate level to indicate its level in the hierarchy, and connected to its parent. Users can expand and collapse portions of the hierarchy.

Using Tree Tables

Tree tables:

- Display hierarchical data in a table
- Combine features of trees and tables

| An ADF Faces Tree Table | |
|-------------------------|-------------|
| Directory Name | Icon |
| My Files | /images/fol |
| META-INF | /images/fol |
| maven | /images/fol |
| oracle.adf.view.faces | /images/fol |
| WEB-INF | /images/fol |
| components | /images/fol |
| explorer | /images/fol |
| feature | /images/fol |
| validate | /images/fol |
| images | /images/fol |
| includes | /images/fol |
| jsLibs | /images/fol |
| media | /images/fol |
| sampleFiles | /images/fol |
| sanityCheck | /images/fol |



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Tree Tables

The ADF Faces tree table component `af:treeTable` displays hierarchical data in the form of a table. The display is more elaborate than the display of an `af:tree` component. The `af:treeTable` component can display columns of data for each tree node in the hierarchy. The component includes mechanisms for focusing in on subtrees within the main tree, as well as expanding and collapsing elements in the hierarchy.

The immediate children of an `af:treeTable` component must be `af:column` components, in the same way as for `af:table` components. The `af:treeTable` component has a `nodeStamp` facet, which renders the object name column. This is the column that contains the primary identifier of an element in the hierarchy. For example, in a hierarchy of products, the object name column might be the category name or the product name.

The `af:treeTable` component supports the same vertical and horizontal grid lines and banding as the table component.

Using Tree Tables (continued)

Both the `af:tree` and `af:treeTable` components render icons that the user can click to expand or collapse a subtree. When the user clicks one of these icons, the component generates a `RowDisclosureEvent`. (You learn more about events in the lesson titled “Responding to Application Events.”) The event has two `RowKeySet` objects: `RemovedSet` for all the collapsed nodes and `AddedSet` for all the expanded nodes. The component expands the subtrees under all nodes in the added set and collapses the subtrees under all nodes in the removed set.

You can register custom `RowDisclosureListener` instances, which can do post processing, on the tree component.

The `af:tree` and `af:treeTable` components use an instance of the `oracle.adf.view.rich.model.RowKeySet` class to keep track of which elements are expanded. This instance is stored as the `disclosedRowKeys` attribute on the component. You can use this instance to control the expand or collapse state of an element in the hierarchy programmatically. Any element contained by the `RowKeySet` instance is expanded, and all other elements are collapsed. The `addAll()` method adds all elements to the set, and the `removeAll()` method removes all elements from the set.

Providing Data for Trees

The screenshot shows the Oracle ADF Faces application development interface. On the left, the 'Data Controls' panel lists several components, including 'ShoppingCartAMDataControl', 'ProductAMDataControl', and 'RootCategory1'. 'RootCategory1' is selected and expanded, showing its attributes: CategoryName, CategoryDescription, CategoryId, ParentCategoryId, CategoryLockedFlag, SubCategory1, Operations, Named Criteria, and BrowseCategory1. On the right, the 'RootCategoryVO.xml' configuration page is displayed. It has tabs for General, Entity Objects, Attributes, and Query. The Query tab contains a Java code block with an SQL query:

```

SELECT ProductCategoryEO.CATEGORY_NAME,
       ProductCategoryEO.CATEGORY_DESCRIPTION,
       ProductCategoryEO.CATEGORY_ID,
       ProductCategoryEO.PARENT_CATEGORY_ID,
       ProductCategoryEO.CATEGORY_LOCKED_FLAG
  FROM PRODUCT_CATEGORIES ProductCategoryEO
 WHERE ProductCategoryEO.PARENT_CATEGORY_ID IS NULL
  
```

Below the configuration page, there is a code snippet in a grey box:

```

<af:tree
    value="#{bindings.RootCategory1.treeModel}"
    var="node">
    <f:facet name="nodeStamp">
        <af:outputText value="#{node}" />
    </f:facet>
</af:tree>
  
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Providing Data for Trees

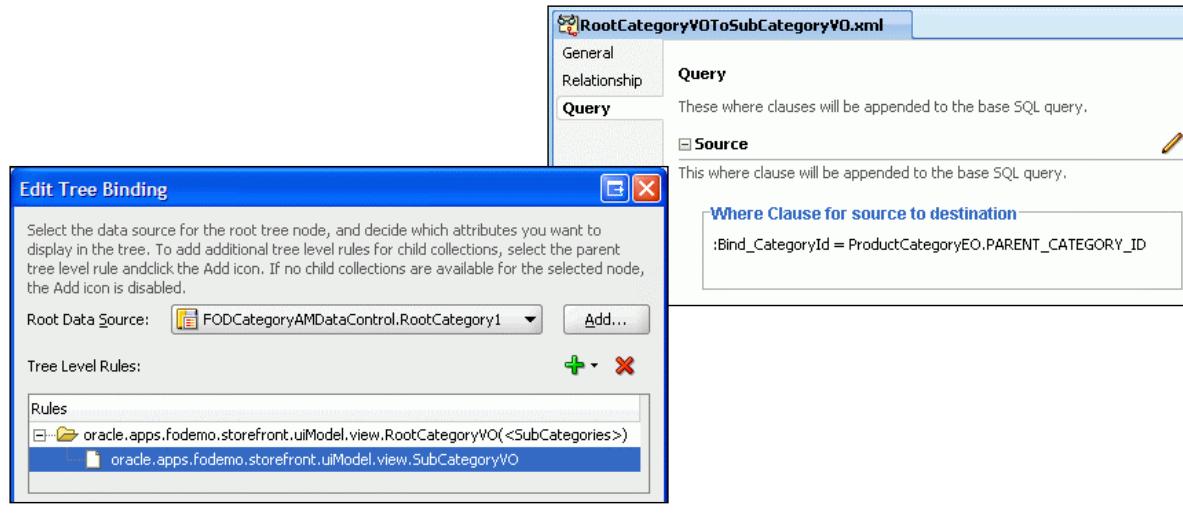
The ADF Faces tree component uses a stamping strategy similar to the ADF Faces table component. The `nodeStamp` facet of the tree is used to display the data for each element in the tree. The tree does not create components for each element; instead, the `nodeStamp` is repeatedly rendered (stamped) once for each element. Because of this stamping behavior, only certain types of components are supported as children inside an ADF Faces tree. All components that have no behavior are supported, as are most components that implement the `ValueHolder` or `ActionSource` interfaces.

Each time the `nodeStamp` is stamped, the data for the current element is copied into a property that can be addressed using an EL expression. You specify the name to use for this property in the `var` property on the tree. After the table has completed rendering, this property is removed or reverted to its previous value.

In the example in the slide, the data for each element is referenced using the variable `node`, which identifies the data to be displayed in the tree. The `nodeStamp` facet displays the data for each element by getting further properties from the `node` variable.

Providing Data for Trees

A tree binding rule for the root tree node, with an additional level for the child collection, provides the hierarchical data.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Providing Data for Trees (continued)

To build a hierarchical tree view in ADF Faces with ADF Business Components, you need to create related view objects that build the basis for the tree binding rules. The data for this particular tree comes from the RootCategory1 view object instance. This VO queries for root categories (those with no parent category). Subnodes of the tree display the subcategories from the instance of the SubCategoryVO view object, which is linked to the RootCategoryVO with a view link. In this way, a typical SQL hierarchical query with a `start with...connect by prior` clause is constructed to use as the tree data. The query starts with categories whose parent category ID is null, and connects category ID to parent category ID.

ADF Faces panelCollection Component

The panelCollection component is one of the ADF layout components.

- It is used in conjunction with collection components, such as `table`, `tree`, and `treeTable`.
- It provides additional capabilities to the components that are placed inside it:
 - Provides default menus and toolbar buttons to `tables`, `trees`, and `treeTables`
 - Allows for multisorting of columns, the ability to hide/wrap columns, and to freeze a column while scrolling other columns



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

panelCollection Component

ADF Faces provides a number of layout components that can be used to arrange other components on a page. Usually, you begin building your page with these components. You then add components that provide other functionality (for example, rendering data or rendering buttons) either inside facets or as child components to these layout components.

The panelCollection component is a layout component that is very useful for displaying collection components, such as `table`, `tree`, and `treeTable`, because it adds more functionality to the components placed inside it. You can add a toolbar and a status bar to `tables`, `trees`, and `tree tables` by surrounding them with the panelCollection component. The top panel contains a standard menu bar as well as a toolbar that holds menu-type components such as menus and menu options, toolbars and toolbar buttons, and status bars. Some buttons and menus are added by default. For example, when you surround a `table`, `tree`, or `tree table` with a panelCollection component, a toolbar that contains the View menu is added. This menu contains menu items that are specific to the `table`, `tree`, or `tree table` component.

Using ADF Faces Output Components

| Output Components: |
|-----------------------------------|
| • icon |
| • image |
| • media |
| • message |
| • messages |
| • outputFormatted |
| • outputLabel |
| • outputText |

With output components, you can:

- Display messages, text, icons, and images
- Play audio and video files

Source:

```
<af:messages/>
<af:outputText value="Error icon:"/>
<af:icon name="error"
         shortDesc="Error Icon"/>
<af:outputText value="Info icon:"/>
<af:icon name="info"/>
<af:outputText value="Media component: "/>
<af:media source="/Brian.wma"
           standbyText="One moment please"/>
```

Design time:



Run time:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using ADF Faces Output Components

ADF Faces provides components for displaying text, icons, and images, and for playing audio and video clips on application pages.

You can output error or warning messages with the `af:message` or `af:messages` component.

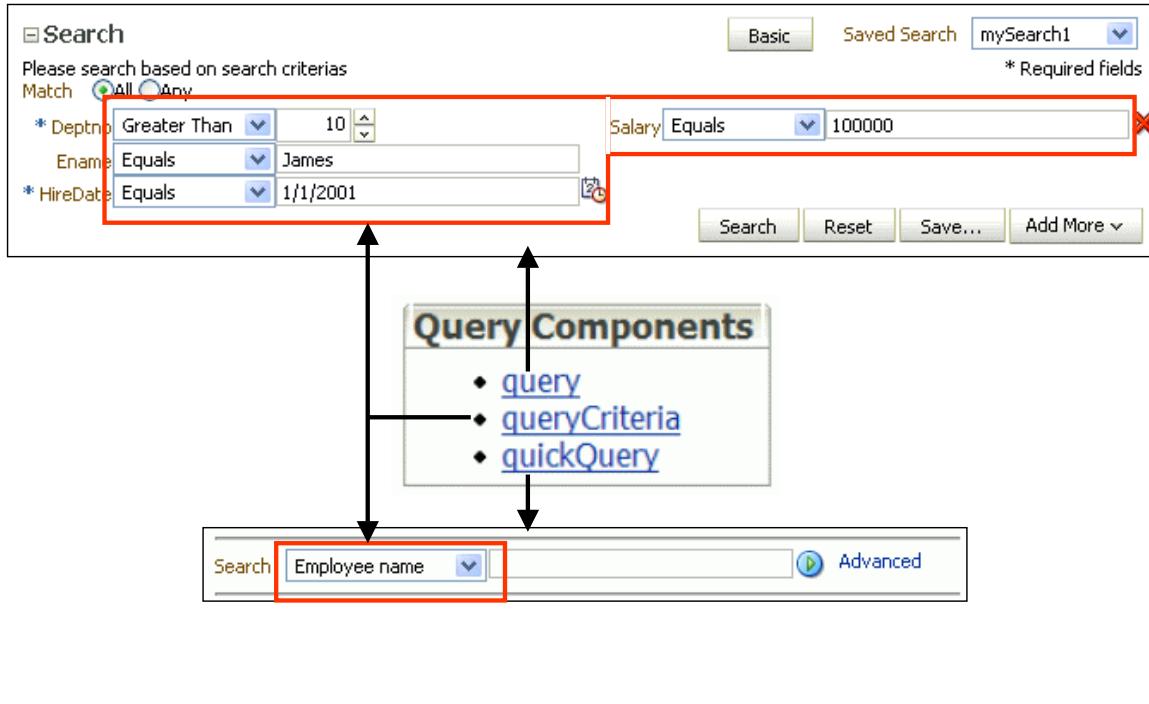
Read-only text, either specified explicitly or from a resource bundle or bean, is displayed using the `af:outputText` and `af:outputFormatted` components. As implied by the names of the components, you can specify a certain amount of formatting for text displayed using the `af:outputFormatted` component. For styling output text, setting the whole of the text to some style, you use the styling features available with both components.

Each skin used for the presentation of an application provides icons representing standard functions, such as an error or a warning, which you can display using the `af:icon` component with the name of the type of icon (`required`, `warning`, `info`, `error`, or `logo`).

Many other ADF Faces components can have icons associated with them, such as a menu where each of the menu items can have an associated icon. You identify the image to use for each one as the value of an `icon` attribute for the component itself.

To play back an audio clip or a video clip, you use the `af:media` component. These components have attributes for you to define how the item is presented on the page.

Using ADF Faces Query Components



Using ADF Faces Query Components

ADF supports two types of search forms:

- The **query** search form, created using the `af:query` component, is a full-featured search form. An `af:query` component includes an `af:queryCriteria` component, which represents the search panel or form. It can be used to render the criteria fields without the toolbar at the top and the button bar at the bottom. The query control enables a user to perform the following tasks:
 - Manage existing searches.
 - Perform operations including creating a search, deleting a search, updating a search, duplicating a search, and resetting a search.
 - Execute a query based on values defined for the current search.
 - Adjust the current search, for example, by adding or deleting search fields, changing the search mode. The quick query search form, created using the `af:quickQuery` component, is a simplified form that has only one search criterion.
- The **quick query** control enables a user to perform a quick search for a textual string across one or more criteria items. The `af:quickQuery` component is usually used in conjunction with a table or a tree table component to display the query results.

Using the af:query Component

The screenshot shows a search interface with the following details:

- Search Mode:** Advanced mode is selected.
- Search Criteria:**
 - * Deptno: Greater Than 10
 - Ename: Equals James
 - * HireDate: Equals 1/1/2001
 - Salary: Equals 100000 (dynamically added)
- Buttons:** Basic, Saved Search (mySearch1), Search, Reset, Save..., Add More.
- Labels:** * Required fields.

Advanced mode query with dynamically added search criterion

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using the af:query Component

The af:query component is used for transactional searches, providing a comprehensive set of search criteria and controls. The query component is a full-featured component that can support multiple search criteria, dynamically adding and deleting criteria, selectable search operators, match all/any selections, seeded or saved searches, basic and advanced modes, and personalization of searches.

It has the following two modes that the user can toggle by clicking a button that appears if the `ModeChangeVisible` property of the query component is set to `true`:

- **Basic mode** query form features include:
 - Drop-down list of selectable search criteria operators
 - Custom list of operators
 - Custom operators
 - Selectable where clause conjunction of either and or or (match all or match any)
 - Saved searches and seeded searches
 - Personalizing saved searches
- **Advanced mode** query form features include the ability for a user to dynamically add search criteria by selecting from a list of searchable attributes. The user can subsequently delete any criteria that were added.

Using the af :query Component (continued)

When you drop an af :query component onto a JSF page, the component provides a search panel that includes one or more input fields for entering search criteria values. The user can select from the drop-down list of operators to create a SQL Query WHERE clause for the search. You can configure an input field to be a list of values (LOV), a number spinner, a date picker, or other control.

A Match All/Match Any radio group further modifies the query. A Match All selection is essentially an AND function. The query returns only rows that match all of the selected criteria. A Match Any selection is an OR function, where the query returns all rows that match any one of the criteria.

After users enter all the search criteria values (including null values) and select the Match All/Any option button, they can click the Search button to initiate the query. The query results can be displayed in any output component. Typically the output component is a table or tree table, but you can associate other display components such as af :form, af :outputText, and graphics to display the results component.

If you enable the Basic/Advanced button, the user can toggle between the two modes. Each mode displays only the search criteria that were defined for that mode. You can define a search criteria field to appear only for basic, only for advanced, or for both modes.

In advanced mode, the control panel also includes an Add More button that displays a pop-up list of searchable attributes. When the user selects any of these attributes, a dynamically generated search criteria input field and drop-down operator appears. All search criteria input fields are positioned in the order in which the attributes are defined in the data source. Any newly created search criteria fields will also follow this rule. For example, if the attributes in the underlying data source are ordered A, B, C, D, and E, and the default search fields are A, C, and E. If the user adds D, the search fields appear in the order A, C, D, E.

This newly created search criteria field also has a delete icon next to it. The user can subsequently click this icon to delete the added field. The originally defined search criteria fields do not have a delete icon and, therefore, the user cannot delete them.

Search criteria can be saved and reaccessed using a given search name, but only if the application is configured to use persistence.

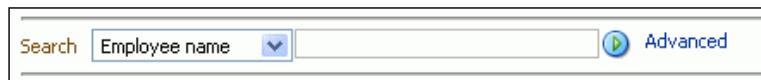
Note: Certain properties set on the view criteria in the ADF BC layer affect the run-time behavior of the query, such as the following:

- **Required property of view criteria row:** Determines whether a view criteria item is a required or optional part of the attribute value comparison in the generated WHERE clause. The default is optional, which means that no exception is generated for null values. If set to required, an exception is thrown when no value is supplied. If set to Selectively Required, the view criteria item is ignored at run time if no value is supplied and another criteria item at the same level has a value; otherwise, an exception is thrown.
- **mode property of view criteria:** Determines the initial mode, Basic or Advanced

Using the af:quickQuery Component

A quick query:

- Has a selectable list of attributes to search on
- Has a single search criterion input field
- Can have a link to more advanced search capabilities, with switching implemented in a managed bean



Quick query component in horizontal layout

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using the af:quickQuery Component

The quick query component has a single search criteria input field. The user can select which attribute to search by selecting from a drop-down list. The available searchable attributes are drawn from the model or from a managed bean. The user can search against the selected attribute or against all attributes. The search criteria field type will automatically match the attribute type. For example, if the selected search field is of type VARCHAR, an Input Text component is rendered.

Quick query components may be used as the starting point of a more complex transactional search that uses a query component. For example, the user may perform a quick query search on one attribute, and if successful, may want to continue to a more complex search. The quick query component supports this by having a built-in advanced link that is an af:commandLink component. You can create a managed bean to enable the user to switch from a quick query to a query component.

Under the af:quickQuery tag are several attributes that define the quick query properties. The attributes of interest include:

- The id attribute, which uniquely identifies the quick query. This value should be set to match the results table or the component's partialTriggers value.

Using the af:quickQuery Component (continued)

- The Layout attribute, which specifies the quick query layout to be either horizontal or vertical
- The Label attribute, which specifies the name to be displayed at the top of the search panel. It is usually an af:outputText component.

The af:quickQuery tag supports the following three facets:

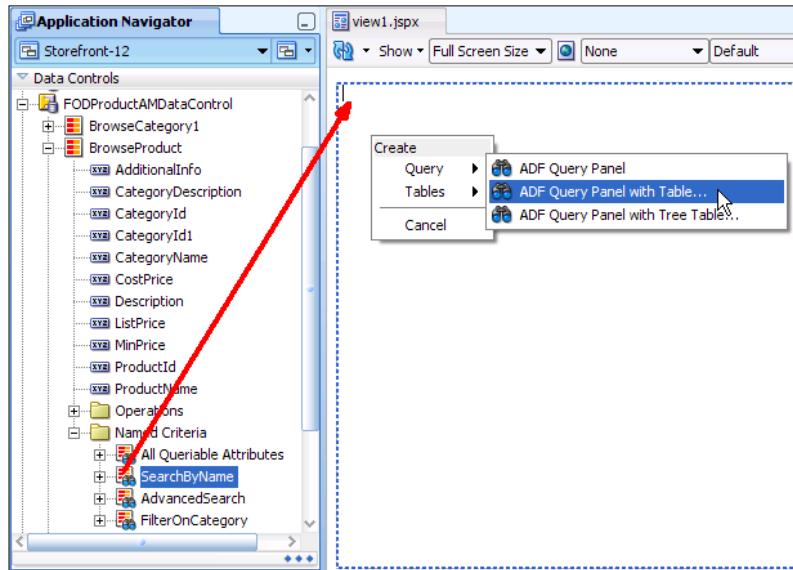
- The criteriaItems facet specifies the component that displays the search attributes of the search object. This component is usually an af:selectOneChoice, but other components can be specified. You are responsible for connecting the quick query component with the model and implementing the queryListener in the model.
- The criterionValue facet specifies the component that displays or accepts the value of the search attribute of the search object. This component is usually an input component, such as af:inputText, but other components can be specified. You are responsible for connecting the quick query component with the model and implementing queryListener in the model.
- The end facet, where you can add a command component to enable users to change from a quick query to a query (advanced mode)

The example shown in the slide has the following source code:

```
<af:quickQuery id="personsViewCriteriaOneQueryId" label="Search"
value="#{bindings.PersonsViewCriteriaOneQuery.queryDescriptor}"
queryListener="#{bindings.PersonsViewCriteriaOneQuery.quickSearch}"
layout="vertical">
    <f:facet name="criteriaItems">
        <af:selectOneChoice label="criteriaItems" simple="true"
            shortDesc="value"
            value="#{bindings.PersonsViewCriteriaOneQuery.selectedItem.value}">
            <f:selectItems
value="#{bindings.PersonsViewCriteriaOneQuery.searchableItems}"/>
        </af:selectOneChoice>
    </f:facet>
    <f:facet name="criterionValue">
        <af:inputText label="criterionValue" simple="true" shortDesc="value"
value="#{bindings.PersonsViewCriteriaOneQuery.selectedItem.criterionValue}"/>
    </f:facet>
    <f:facet name="end">
        <af:commandLink text="Advanced" rendered="true"/>
    </f:facet>
</af:quickQuery>
```

Creating a Query Search Form

Drag a named criteria from the Data Controls panel:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Query Search Form

You create a query search form by dropping a named view criteria item from the Data Control panel onto a page. The named criteria represent the view criteria that have been defined for the data collection in the model. When you drop a named view criteria onto a page, that view criteria is the basis for the initial search form. All other view criteria defined against that data collection appear in the Saved Search drop-down list. Users can then select any of the view criteria search forms, and also any saved searches.

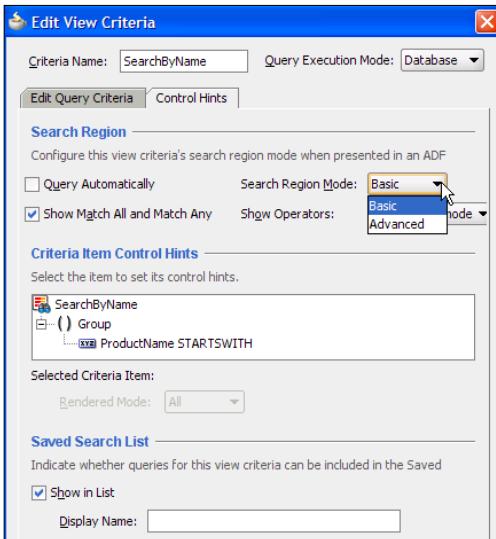
You have a choice of dropping only a search panel, dropping a search with a results table, or dropping a search with a tree table. If you choose to drop the search panel with a table, you can select the filtering option to create a filtered table.

Normally, you would drop a query search panel with the results table or tree. JDeveloper automatically wires up the results table or tree with the query panel. If you drop the query search panel without the results table or tree, you can designate a component that is based on the query's data collection as a results component. In the Property Inspector for the query panel, copy the value of the ResultsComponentId field, and then paste that value into the Id field for the component that you want to designate as a results component.

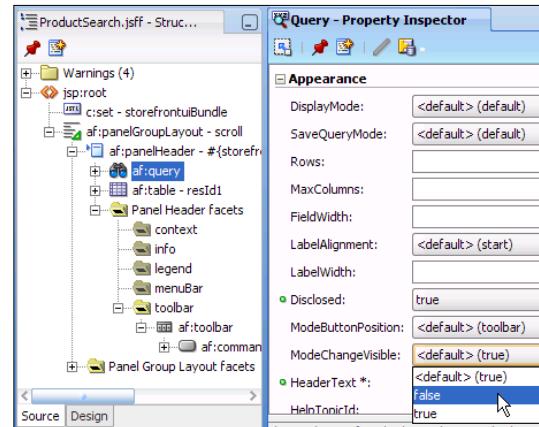
Modifying Query Behavior

You can set properties that affect query behavior:

On the model's view criteria:



On the query in the UI:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Modifying Query Behavior

When you define the view criteria in a view object, you specify the attributes and conjunctions that form the query statement and serve as a basis for the query search form. You can also specify some of the default properties of the search form after it has been dropped onto the page. These properties include setting the basic/advanced mode, automatic query execution, and search criteria field rendering.

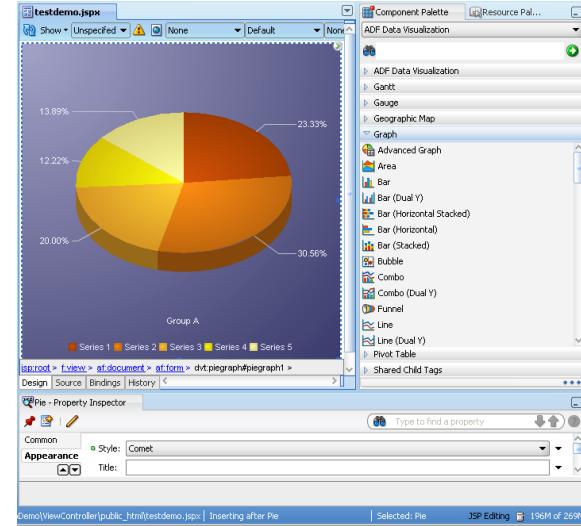
After the query search form is created, you can set other properties such as setting the ID of the results table or component. For example, to hide the Basic/Advanced toggle button, you can set the `ModeChangeVisible` field to `false`.

Additionally, you can add a Cancel button to the footer facet of the query search form and use it to call code in the business or model layer that would cancel a long-running query.

Using ADF Data Visualization Components

Common features:

- Design time creation by using:
 - Data Controls panel
 - JSF visual editor
 - Property Inspector
 - Component Palette
- Live data preview at design time
- Support for data binding



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of ADF Data Visualization Components

ADF Data Visualization components are a set of rich interactive ADF Faces components that provide significant graphical and tabular capabilities for analyzing data. ADF Data Visualization components have the following characteristics in common:

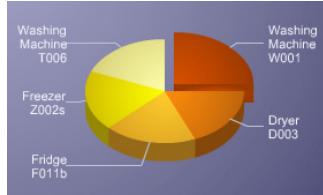
- You create them at design time by using the Data Controls panel, the visual editor, the Property Inspector, and the Component Palette.
- Design time provides a live data preview.
- You can bind data to standard rowset data controls, as well as hierarchical, BAM, and BI data controls.

To see a demo of these components, go to

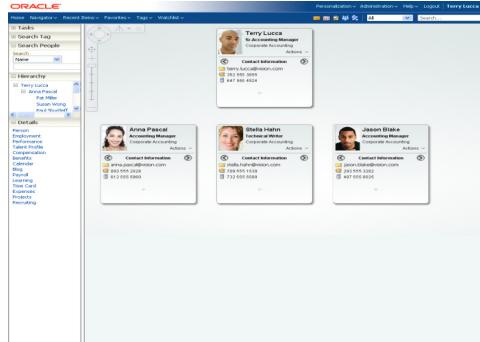
<http://www.oracle.com/technology/products/jdev/viewlets/11/index.html> and select Data Visualization and Graphs for JSF.

Visualizing Data

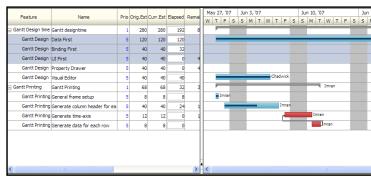
The following types of data visualization components are available:



Graph



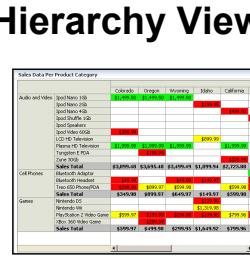
Hierarchy Viewer



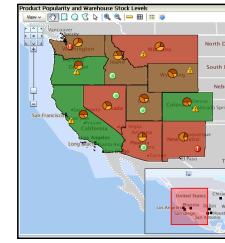
Gantt chart



Gauge



Pivot table



Geographic map

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Visualizing Data

The following list identifies the data visualization components:

- Graph:** Supports more than 50 types of graphs, including bar, pie, line, scatter, and stock graphs, enabling you to evaluate data points on multiple axes in a variety of ways. Part of JDeveloper since 10g, Graph is now a JSF component with many new features added:
 - There are several new graph types, including funnel, floating bar, and fit to curve.
 - Graphs can render in Flash or in Portable Network Graphics (PNG), an XML-based language for drawing two-dimensional graphics.
 - Interactivity features include zooming and scrolling, time selector window, line and legend highlighting and fading, and dynamic reference lines and areas.
 - The JSF graph tag is improved, with simplified tags for 17 commonly used graph types.
- Hierarchy Viewer:** The hierarchy viewer component displays hierarchical data as a set of linked nodes in a diagram. The nodes and links correspond to the elements and relationships to the data. The component supports pan and zoom operations, expanding and collapsing of the nodes, rendering of simple ADF Faces components within the nodes, and search of the hierarchy viewer data. A common use of the hierarchy viewer is to display an organization chart, as shown in the slide.

Visualizing Data (continued)

- **Gauge:** A new data visualization component in JDeveloper 11g that focuses on identification of problems in data. Supported gauge types are:
 - Dial: Standard and Threshold
 - Status Meter: Standard and Threshold
 - LED
- **Geographic map:** A new data visualization component in JDeveloper 11g that provides functionality of Oracle Spatial within the ADF framework. It enables you to represent business data on a geographic map, and supports superimposing multiple layers of information on a single map. Available map types are:
 - Thematic
 - Pie
 - Bar
 - Point
- **Pivot table:** A new data visualization component in JDeveloper 11g that supports multiple layers of data labels on a row or a column edge and automatic calculation of subtotals and totals. It enables you to switch data labels from one edge to another to obtain different views of your data. Supported features include:
 - Horizontal and vertical scrolling
 - Header and cell formatting
 - Drag and drop pivoting
 - Automatic totals and subtotals against ADF BC Data Control
 - Drilling against BI Data Control
- **Gantt chart:** A new data visualization component in JDeveloper 11g that provides the ability to track tasks and resources on a time axis. Its purpose is to assist in project planning. The following Gantt Chart types are supported:
 - Project Gantt: Focuses on project management
 - Scheduling Gantt: Focuses on resource management

Summary

In this lesson, you should have learned how to:

- Internationalize the user interface
- Use component facets
- Enable users to:
 - Select a value from a list
 - Select a date from a calendar
- Display tabular data in tables
- Display hierarchical data in trees
- Display text or media with icons and images
- Define search forms and display results
- Display data graphically



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 13 Overview: Using ADF Faces Components

This practice covers the following topics:

- Creating a Table with Row Selection
- Creating a Search Page
- Creating a Read-Only Form
- Creating a Sortable Table
- Creating a Category Tree
- Creating LOVs



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 13 Overview: Using ADF Faces Components

In the practices for this lesson, you begin to add functionality to pages by using ADF Faces components. You create search and detail pages and another page that displays a hierarchical tree of categories and subcategories. You also create several pages that display input components as a list of values.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Osi S.R.L. use only

14

Implementing Navigation on Pages

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Implement command buttons and links
- Create menus
 - Menu bar
 - Pop-up menu
 - Context menu
- Use an ADF menu model
- Define navigation panes
- Use breadcrumbs
- Define trains



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

ADF Faces contains several components to enhance and implement navigation on pages. This lesson teaches you how to implement such navigation by using ADF Faces navigation components.

Using ADF Faces Navigation Components

The screenshot shows the Oracle JDeveloper interface. On the left, there's a palette titled "Navigation Components" containing four items: "command button", "go button", "command link", and "go link". In the center, there's a list titled "Navigation Components" with the following items:

- [breadCrumbs](#)
- [commandButton](#)
- [commandImageLink](#)
- [commandLink](#)
- [commandMenuItem](#)
- [commandNavigationItem](#)
- [commandToolbarButton](#)
- [goButton](#)
- [goLink](#)
- [goImageLink](#)
- [navigationPane](#)
- [region](#)
- [train](#)
- [trainButtonBar](#)

At the top right, there's a breadcrumb trail: Home > Benefits > Insurance > Health.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

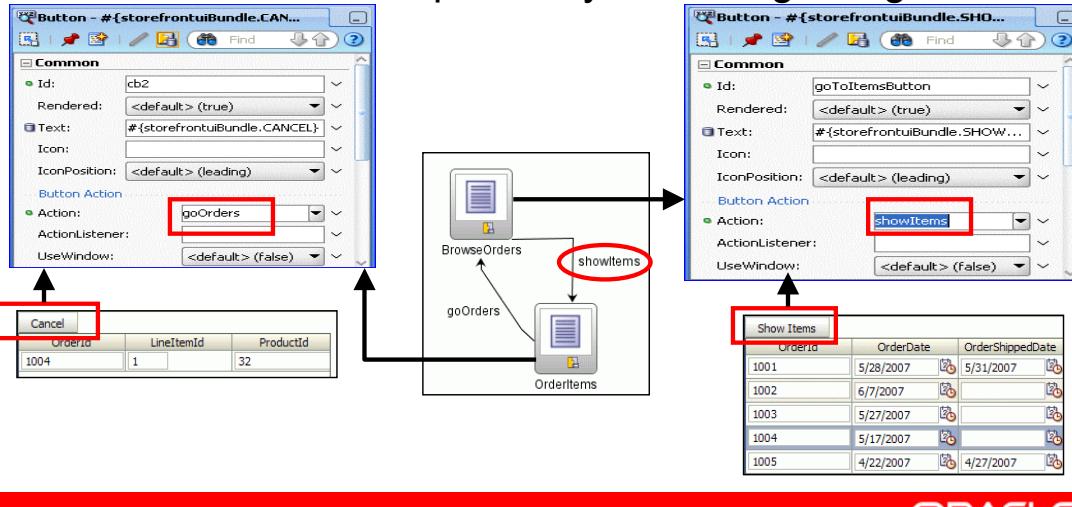
Using ADF Faces Navigation Components

Navigation components in ADF Faces include:

- Button and link components for navigating to another location with or without server-side actions
- Components that render items such as tabs and breadcrumbs for navigating hierarchical pages
- Train components for navigating a multistep process

Performing Navigation

- Control flow rules are defined in the task flow.
- A component fires an action event to generate a String outcome corresponding to the control case from-outcome.
- The event listener responds by executing navigation.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Performing Navigation

Like any JSF application, an application that uses ADF Faces components contains a set of rules for choosing the next page to display when, for example, a button or link is clicked. You define the rules in an ADF Faces application by adding control flow rules and cases to task flows.

JSF uses an outcome String to select the control flow rule to use to perform a page navigation. ADF Faces navigation components generate an action event when users activate the component. The JSF navigation handler and default ActionListener mechanism use the logical outcome String on the activated component to find a match in the set of control flow rules. When JSF locates a match, the corresponding page is selected and rendered.

The example in the slide shows two pages in an ADF task flow:

- The BrowseOrders page has a control-flow-case whose control flow from-outcome is named showItems, which defines that navigation should go to the OrderItems page.
- The OrderItems page has a control-flow-case whose control flow from-outcome is named goOrders, which defines that navigation should return to the BrowseOrders page.

There are two command buttons on the pages:

- The BrowseOrders page has a command button whose action is set to showItems.
- The OrderItems page has a command button whose action is set to goOrders.

Using Buttons and Links

- Command components:
 - Include af:commandButton, af:commandLink, and af:commandImageLink
 - Submit requests and fire action events when activated
 - Are typically used for internal navigation
- Go components:
 - Include af:goButton, af:goLink, and af:goImageLink
 - Navigate directly without server-side actions
 - Are typically used for external navigation
 - Do not use control flow rules
- Command and go components have the same appearance.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Buttons and Links

Buttons and links have an Action property that is a String, or a method that returns a String. This String should correspond to one of the outcomes defined in the task flow, so that it can be used for navigation.

The components commandImageLink and goImageLink render images as links, along with optional text.

Defining Access Keys

- Use for input, command, and go components
- Set component attributes:

- accessKey (input, command, or go component):

```
<af:goLink text="Home" accessKey="h">
```

- textAndAccessKey (command or go components):

```
<af:commandButton textAndAccessKey="&#amp;Home" />
```

- labelAndAccessKey (input components):

```
<af:inputSelectDate value="Choose date"  
labelAndAccessKey="D&#amp;ate" />
```

- valueAndAccessKey (input components):

```
<af:outputLabel for="someid"  
valueAndAccessKey="Select Date&#amp;e" />  
<af:inputText simple="true" id="someid" />
```

- Can define same access key for multiple components



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Access Keys

You can define access keys (also known as mnemonics or shortcut keys) in the Property Inspector for the component by entering the mnemonic character in the AccessKey property.

Use one of the following attributes to specify a keyboard character for an ADF Faces input, command, or go component:

- **accessKey**: Use to set the mnemonic character used to gain quick access to the component. For command buttons, command links, go buttons, and go links, the character specified by this attribute must exist in the text attribute of the component; otherwise, ADF Faces does not display the visual indication that the component has an access key.
- **textAndAccessKey**: Use to simultaneously set the text and the mnemonic character for a component using the ampersand (&) character. In .jspx files, use & for an ampersand. In JSP files and in the Property Inspector, you need to use only the &. The ampersand should precede the character to be used as an access key.
- **labelAndAccessKey**: Use to simultaneously set the label attribute and the accesskey on an input component using conventional ampersand notation
- **valueAndAccessKey**: Use to simultaneously set the value attribute and the access key using conventional ampersand notation. The example in the slide specifies the label as Select Date and sets the access key to e, the letter immediately after the ampersand.

Defining Access Keys (continued)

Reusing Access Keys

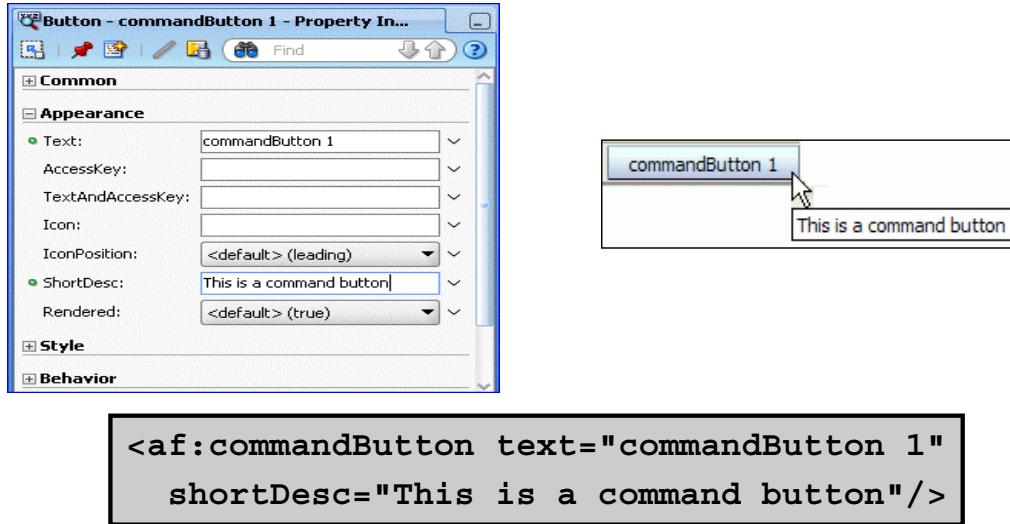
You can bind the same access key to several components. If the same access key appears in multiple locations on the same page, the rendering agent cycles among the components accessed by the same key. That is, each time the access key is pressed, focus moves from component to component. When the last component is reached, focus returns to the first component.

Depending on the browser, if the same access key is assigned to two or more Go components on a page, the browser may activate the first component instead of cycling through the components that are accessed by the same key.

Note: If you assign an access key that is already defined as a shortcut menu in the browser, the ADF Faces component access key takes precedence.

Defining Tool Tips

- Use the shortDesc attribute:



- Can also be defined as a UI Control Hint on EO or VO in the model

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Tool Tips

You can define tool tips in two ways:

- In the component's Property Inspector, you can set the ShortDesc property to define a tool tip for the component.
- In the data model, you can define the tool tip text on the Control Hints tab of the attribute editor that is effective whenever that attribute is displayed in the UI.

Using Toolbars, Toolbar Buttons, and Toolboxes

- Toolbars contain other components.
- Toolbar buttons have special properties. (If you use toolbar buttons, always put them on a toolbar.)
- Toolboxes contain multiple toolbars.
- You can stretch one component on a toolbar.
- Overflow buttons are automatic.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Toolbars, Toolbar Buttons, and Toolboxes

Along with menus, you can create toolbars in your application that contain toolbar buttons used to initiate some operation in the application. The buttons can display text, an icon, or a combination of both. You can use the `group` component to group related toolbar buttons.

Toolbars can also contain other UI components, such as drop-down lists, command buttons, and command links. However, toolbar buttons provide additional functionality:

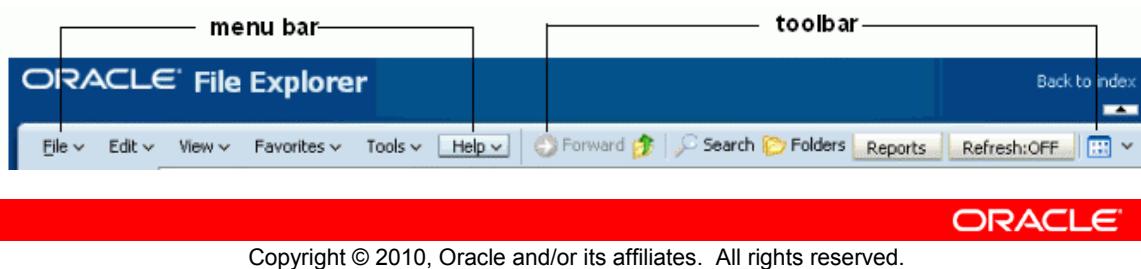
- The `af:commandToolbarButton` component has a `popup` facet that enables you to provide pop-up menus from a toolbar button.
- The `flex` property of a component can be set to an integer that indicates the ratio of this component to other flexible components within the same container.

You can use more than one toolbar component by enclosing them in a toolbox component (in the Layout group of the Component Palette). Doing so stacks the toolbars so that the first toolbar on the page displays on the top, and the last toolbar displays at the bottom. When you use more than one toolbar, you can set the `flex` attribute on the toolbars to determine which toolbar should be the longest. If you want toolbars to be displayed next to each other (rather than stacked), you can enclose them in a group component.

Within a toolbar, you can set one component to stretch so that the toolbar will always equal that of its parent container. When a window is resized such that all the components within the toolbar can no longer be displayed, the toolbar displays an overflow icon.

Using Menus for Navigation

- You can group menu bars and toolbars in a toolbox.
- Menu bars contain menus, which provide the top-level menu.
- Menu items:
 - Provide the vertical (drop-down) selections
 - Like toolbar buttons, can:
 - Display differently when selected
 - Perform navigation



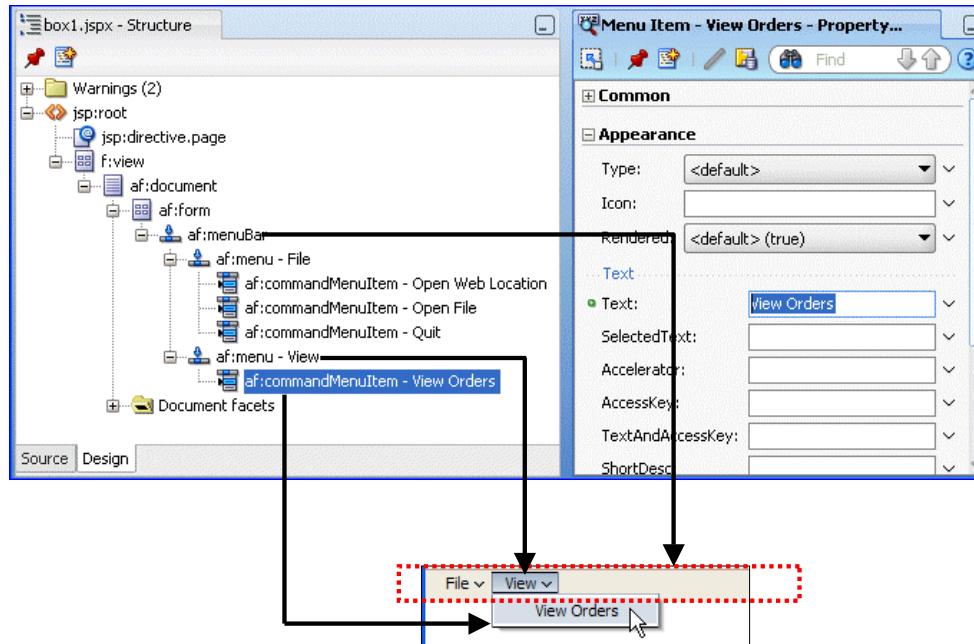
Using Menus for Navigation

ADF Faces provides menu bars and toolbars that you can add to a page. Menu items and toolbar buttons enable users to perform some change on a selected object or to navigate somewhere else in the application. A toolbox can be used to group menus and multiple toolbars together. The example in the slide contains both a menu bar and a toolbar.

Both menu items and toolbar buttons have the capability to display differently when selected. Additionally, they can both navigate to other pages in the application, and perform any logic needed for navigation in the same way that other command components do.

You use the `menuBar` component to render a bar that contains the menu bar items, such as `File` in the example. Each item on a menu bar item is rendered by a `menu` component, which holds a vertical menu. Each vertical menu consists of a list of `commandMenuItem` components that can invoke some operation on the application. You can nest menu components inside menu components to create submenus.

Creating Menus



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Menus

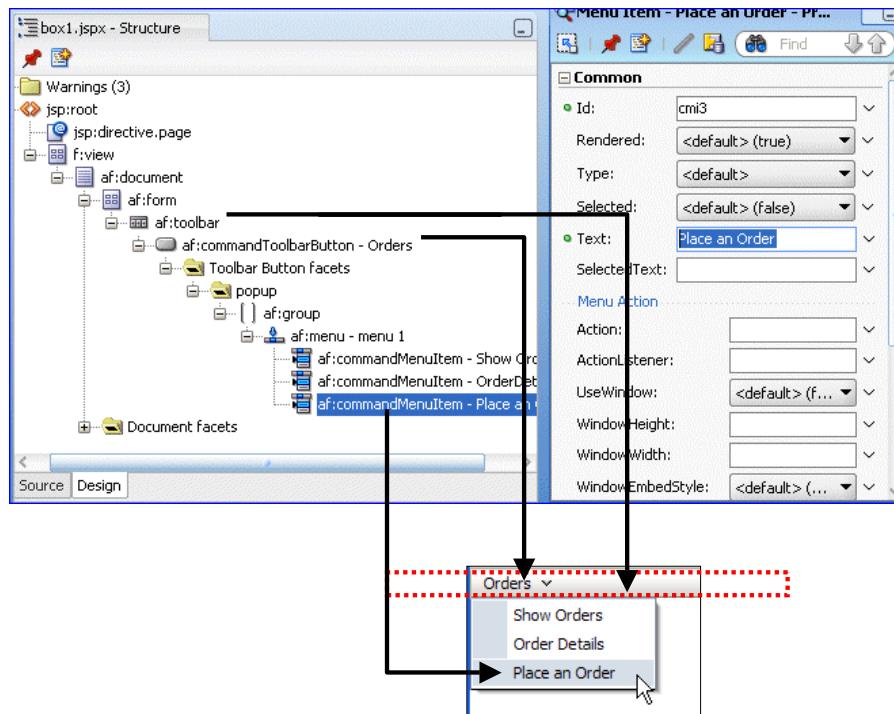
To create and use menus in a menu bar, perform the following steps:

1. Drag a Panel Menu Bar from the Component Palette to the JSF page.
2. Drag the desired number of Menu components to the menu bar.
3. Drag a Menu Item from the Component Palette. You can drag to a menu or directly to the Panel Menu Bar, which renders similar to a button onto the menu bar.
4. Within each menu component, drag Menu Item components from the Component Palette to insert a series of `af:commandMenuItem` components to define the items in the vertical menu.
5. Set properties of menu components in the Property Inspector. In addition to setting text and access key as described previously, you can set the following properties:
 - Accelerator: The keystroke to activate this menu item's command when the item is selected. ADF Faces converts the keystroke and displays a text version of the keystroke (for example, `Ctrl + O`) next to the menu item label.
 - Icon: Enter the URI of the image file you want to display before the menu item label.
 - Disabled: Set this to `true` to disable this menu item.

Creating Menus (continued)

- Type: Specify a type for this menu item. When a menu item type is specified, ADF Faces adds a visual indicator (such as a check mark) and a toggle behavior to the menu item. At run time, when the user selects a menu item with a specified type (other than default), ADF Faces toggles the visual indicator or menu item label. Use one of the following acceptable type values:
 - check: Toggles a check mark next to the menu item label. The check mark is displayed when the menu item is selected.
 - radio: Toggles an option button next to the menu item label. The option button is displayed when the menu item is selected.
 - antonym: Toggles the menu item label. The value set in the `selectedText` attribute is displayed when the menu item is selected, instead of the menu item defined by the value of `text` or `textAndAccessKey` (which is what is displayed when the menu item is not selected). If you select this type, you must set a value for the `selectedText` attribute.
 - default: No type is assigned to this menu item. The menu item is displayed in the same manner whether or not it is selected.
- Selected: Set to `true` to make this menu item selected. By default, a menu item is not selected. The `selected` attribute is supported for check, radio, and antonym type menu items only.
- SelectedText: Set the alternate label to display for this menu item when the menu item is selected. The `type` attribute for the menu item must be set to `antonym`.
- Action: Use an EL expression that evaluates to an action method in an object (such as a managed bean) to be invoked when this menu item is activated by the user. The expression must evaluate to a public method that takes no parameters, and returns a `java.lang.Object`. If you want to cause navigation in response to the action generated by `commandMenuItem`, instead of entering an EL expression, enter a static action outcome value as the value for the `action` attribute. You then need to either set `partialSubmit` to `false`, or use a redirect.
- `actionListener`: Specify the expression that refers to an action listener method that will be notified when this menu item is activated by the user. This method can be used instead of a method bound to the `action` attribute, allowing the `action` attribute to handle navigation only. The expression must evaluate to a public method that takes an `ActionEvent` parameter, with a return type of `void`.

Creating Pop-Up Menus



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

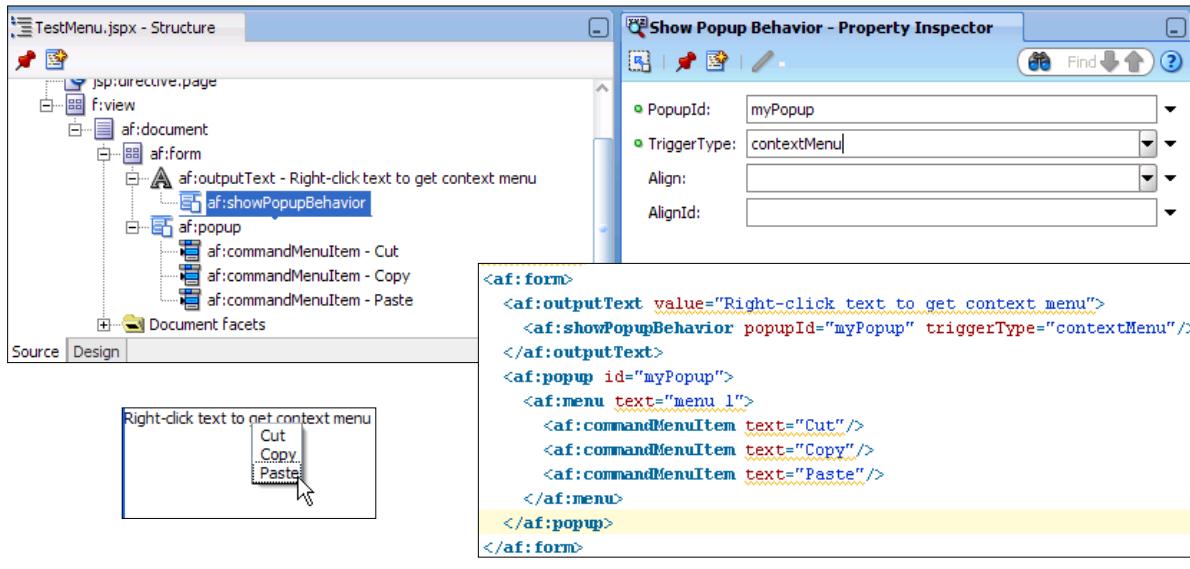
Creating Pop-Up Menus

You can also have the menu launch in a pop-up window that displays vertically. To create a pop-up menu, perform the following steps:

1. Drag the ToolbarButton component to a toolbar on the JSF page and set the text attribute to display the name of the button. Use the icon attribute to set the image to use on the button.
2. In the Structure window, expand the ToolbarButton facets and drag a Menu component to the popup facet.
3. Drag to the Menu component as many Menu Item components as needed to define the items in the vertical menu. Set properties on the Menu Items as needed.

Creating Context Menus

To create a context menu, use the `af : showPopupBehavior` operation.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Context Menus

ADF Faces client behavior tags provide declarative solutions to common client operations that you would otherwise have to write yourself using JavaScript, and register on components as client listeners. In this release, ADF Faces supports the client behavior `af : showPopupBehavior` to use in place of a client listener. You perform the following steps to use `af : showPopupBehavior` to create a context menu for a component:

1. Create a pop-up menu with `af : popup`, `af : menu`, and `af : commandMenuItem` components.
2. Give the pop-up menu an ID.
3. Create a component to use the context menu.
4. Drag the `af : showPopupBehavior` operation to the component and set properties:
 - `popupId`: Specify the ID of the `af : popup` component whose contents you want to display in a pop-up.
 - `alignId`: Specify the ID of the component to align the pop-up contents with.
 - `align`: Specify an alignment position that is relative to the component identified by `alignId`.

Creating Context Menus (continued)

`triggerType`: Specify the event type to use to trigger the pop-up. Default is `action`, because typically you would associate `af:showPopupBehavior` with a command component. When the command component is clicked, an action event is fired, which triggers the pop-up to display. If you associate `af:showPopupBehavior` with some other noncommand component, such as `af:outputText`, set `triggerType` on `af:showPopupBehavior` to `contextMenu`, which displays a pop-up context menu. The slide shows a sample code that displays a pop-up menu when users right-click the text rendered by `af:outputText`. The slide also shows the sample context menu generated at run time.

Note: You can use the same general steps to display a pop-up window that contains any components, not just one that contains a menu.

Using a Navigation Pane

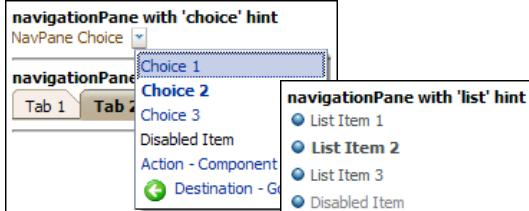
- You can either:
 - Define navigation levels explicitly
Or
 - Bind to XML Menu Model

navigationPane with 'bar' hint
Bar Item 1 | Bar Item 2 | Bar Item 3 | Disabled Item | Component Guide

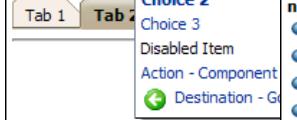
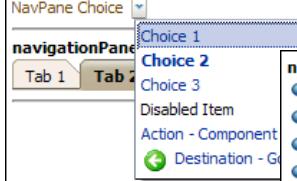
```
<af:navigationPane hint="bar" id="barExample">
  <af:commandNavigationItem text="Bar Item 1" partialSubmit="true"
    actionListener="#{demoCommandNavigationItem.navigationItemAction}"/>
  <af:commandNavigationItem text="Bar Item 2" partialSubmit="true" selected="true"
    actionListener="#{demoCommandNavigationItem.navigationItemAction}"/>
  <af:commandNavigationItem text="Bar Item 3" partialSubmit="true"
    actionListener="#{demoCommandNavigationItem.navigationItemAction}"/>
  <af:commandNavigationItem text="Disabled Item" partialSubmit="true" disabled="true"
    actionListener="#{demoCommandNavigationItem.navigationItemAction}"/>
  <af:commandNavigationItem text="Component Guide" immediate="true" action="guide"/>
</af:navigationPane>
```



navigationPane with 'buttons' hint
Button 1 | Button 2 | Button 3 | Disabled Item | Component Guide



navigationPane with 'choice' hint



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using a Navigation Pane

The `af:navigationPane` component is a layout component that creates a series of navigation items representing one level in a navigation hierarchy. You can create this hierarchy manually, or you can bind the navigation pane to the XML Menu Model. You create one navigation pane for each level of the hierarchy.

To set up a hierarchy manually, you define one global navigation rule that can access each page in the hierarchy. Then create a navigation pane for each level of the hierarchy, setting the Hint attribute as described below. Drag navigation items to each navigation pane for the pages that should display at its level of the hierarchy. For each navigation item, set the Action to a String outcome corresponding to one of the outcomes in the global navigation rule.

Depending on the Hint attribute of the navigation pane, the navigation items can be rendered as:

- **Bar:** Displays the navigation items separated by a bar
- **Buttons**
- **Choice:** Displays navigation items in a pop-up list when the associated icon is clicked (You must include a value for the navigation pane's icon attribute.)
- **List:** Bulleted list
- **Tabs**

You would need to add the navigation panes to each page that should display them, or use a page fragment or page template, discussed in the lesson titled “Ensuring Reusability.”

Using Breadcrumbs

- Breadcrumbs: Use the `af:breadcrumbs` component.
- Links: Use the `af:commandNavigationItem` component.
- You can define:
 - Breadcrumbs explicitly
 - Links using the XML Menu Model

The red bar spans most of the width of the slide, centered horizontally.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Breadcrumbs

The `af:breadcrumbs` component enables you to display a path of links for navigation. You can explicitly define the links by using one `af:commandNavigationItem` component for each. Alternatively, you can define the links dynamically by using the XML Menu Model.

You need to add the breadcrumbs to each page that should display them, or use a page fragment or page template, discussed in the lesson titled “Ensuring Reusability.”

Using Explicitly Defined Breadcrumbs

- Use static values:

```
<af:breadcrumbs>
    <af:commandNavigationItem text="Store"/>
    <af:commandNavigationItem text="Media"/>
    <af:commandNavigationItem text="Music"/>
</af:breadcrumbs>
```

| ProductName | Description | Price |
|-------------------------------|--|-------|
| Essential Bach | It is not possible to fit all of Bach's music onto one CD. | 15.99 |
| Beethoven Symphony No. 5 | These are thoroughl | 18.99 |
| Essential Mozart | This 2CDs set is quite | 15.99 |
| Chopin: Favorite Piano Etudes | In this recording we have | 17.99 |
| Tchaikovsky's Ballet | We haven't had a dis | 16.99 |
| Dvorak's Great Symphonies | CD 1 THE THREE GRE | 16.99 |

- Can also use EL for dynamic values

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Explicitly Defined Breadcrumbs

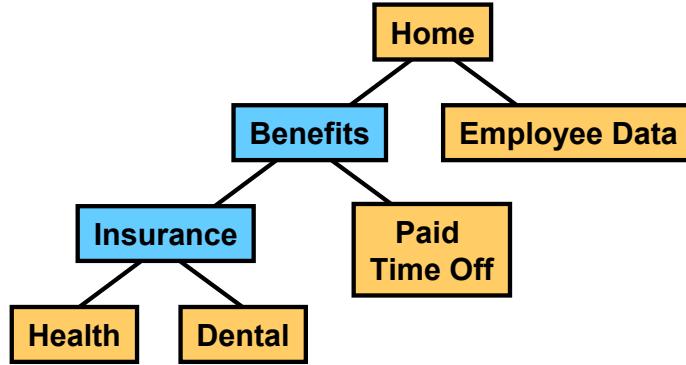
To provide a simple navigation hierarchy where you define the navigation items for the breadcrumbs, you can add the af : breadcrumbs component to each page, with one or more af : commandNavigationItem components that you explicitly set. The value of the af : commandNavigationItem component's attributes may be set to a page or to an EL expression.

For example, in a shopping application, you may want to enable the user to follow the breadcrumbs to display categories and subcategories of products, rather than pages, as shown in the example in the slide. Instead of static values, you can use parameters and action listeners to make these breadcrumbs dynamic, displaying the names of the categories and subcategories by using EL expressions that evaluate to changing parameter values.

Using XML Menu Model for Dynamic Navigation Items

XML Menu Model:

- Represents navigation for a page hierarchy in XML format
- Contains the following elements:
 - menu
 - groupNode
 - itemNode
 - sharedNode



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using XML Menu Model for Dynamic Breadcrumbs

For a very simple page hierarchy, you can manually create navigational components, such as breadcrumbs or navigation panes, and add navigation items for each. However, this becomes very cumbersome when a navigational hierarchy encompasses several pages. An ADF menu model enables you to create complex navigational hierarchies without all the tedious manual coding.

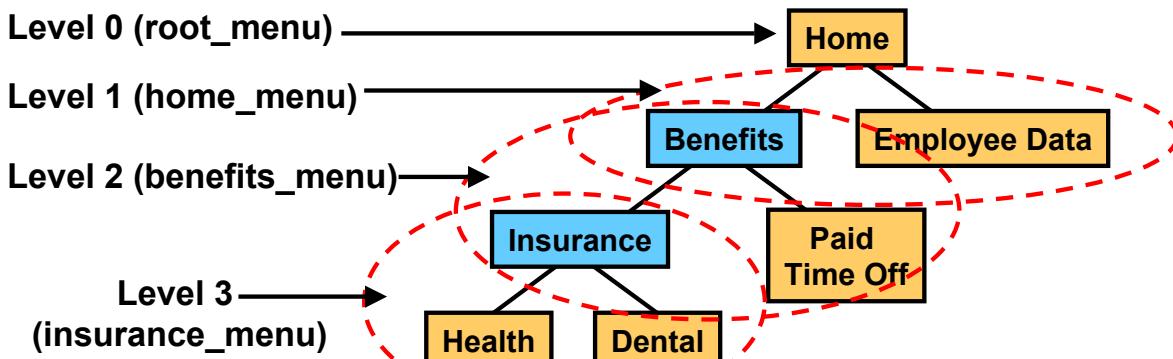
You can organize related JSF pages in a tree-like hierarchy that users can navigate to drill down or up to another level of the hierarchy by using navigation items in navigation panes or breadcrumbs. The XML Menu Model represents navigation for a page hierarchy in XML format. The hierarchy is described within the menu element. The hierarchy is composed of nodes, each of which can correspond to a page. The menu element can contain the following types of nodes:

- groupNode: Contains one or more child nodes to navigate to indirectly, and must reference the ID of at least one of its child nodes, which can be an item node or another group node
- itemNode: A node that performs navigation upon selection
- sharedNode: References another XML menu, and is used to link or include various menus

For more information and examples of creating an XML Menu hierarchy, see the *Web User Interface Developer's Guide for ADF* and the *Fusion Developer's Guide for ADF*.

Creating an ADF Menu Model

- Create the page hierarchy in one or more unbounded task flows.
- In the Application Navigator, right-click each task flow and select Create ADF Menu.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating an ADF Menu Model

You can use the ADF Controller features in conjunction with the XML Menu Model to build the page hierarchy. JDeveloper creates the nodes in the metadata file for your page hierarchy automatically, and also creates the links.

To create a page hierarchy, you first divide the nodes into menus, similar to the example shown in the slide. The example shows the top-level menu as containing the Home page, which is represented as an item node. It contains a navigation pane with two tabs: one that links to Employee Data (an item node), and the other that links to Benefits (a shared node).

The Benefits menu adds two links to Insurance (shared node) and Paid Time Off (item node).

The Insurance menu adds two links to the Health and Dental item nodes.

To create a page hierarchy, perform the following steps:

1. Create an unbounded task flow for each menu in the page hierarchy. You can put all the pages in `adfc-config.xml`, but for reuse you may opt to create multiple menus. The example contains three menus: Home, Benefits, and Insurance. So in the example, the developer would create three unbounded task flows: `adfc-config-home.xml`, `adfc-config-benefits.xml`, and `adfc-config-insurance.xml`. (It is helpful to prefix unbounded task flow names with `adfc` or `adfc-config` to easily identify them as unbounded task flows.) The default task flow, `adfc-config.xml`, serves as the root menu.

Creating an ADF Menu Model (continued)

2. Add view activities to each unbounded task flow that correspond to each menu node for which you want to create a page.

For example, the Benefits menu contains:

A group node, Benefits, for grouping the subnodes

A shared node that points to the Insurance menu

An item node, Paid Time Off, that corresponds to a page

Therefore, in the `adfc-config-benefits` task flow, you would add one view activity (for paid time off). Do not create views for nodes that do not have a page in this task flow.

3. Create the pages for your menu hierarchy: one page for each node in the hierarchy. Alternatively, you could create the pages first and then drag them to the appropriate view activities in the task flows, or drag them to create the view activities.
4. In the Application Navigator, right-click each unbounded task flow and select Create ADF Menu. Create in a bottom-up manner, that is, create the submenus first, and then the parent menus. In the Create ADF Menu Model dialog box, give each menu a descriptive file name. You can use the `root_menu` default file name for the root `adfc-config.xml` file's menu. When you create an ADF menu for a task flow, a global navigation rule is added to the task flow to access any views in that task flow.
5. In the Application Navigator, select the menu XML file that was just created. In the Structure window, you see that all views in the unbounded task flow have corresponding item nodes in the menu XML file. Create the node hierarchy by:
 - Creating group nodes for those groups that do not have a view activity: Right-click the menu node and select “Insert inside menu” > `groupNode`. Give the group node a descriptive name (such as `groupNode_Benefits`), set `idref` to the ID of one of the node's children (such as `itemNode_PaidTimeOff`), and set the label that will display on navigation items (for example, Benefits).
 - Converting to group nodes any nodes that you want to use as a group node, but that do have a view activity and so already exist in the menu. Right-click the node and select Convert > `groupNode`. Change the ID if desired (for example, from `itemNode_<name>` to `groupNode_<name>`), and set `idref` and label as in the step just above.
 - Creating shared nodes to reference other menus to access from this one. Right-click the menu node and select “Insert inside menu” > `sharedNode`. Set the `ref` property to the EL expression for the referenced menu, such as `# {insurance_menu}`.
 - Dragging each child to its parent. In the example of the Benefits menu, you would have created a shared node that references the insurance menu and a group node named `groupNode_Benefits` and labeled Benefits. You would drag the shared node and `itemNode_PaidTimeOff` to the group node.
 - Checking the labels of all group and item nodes—the label is what is displayed in the navigation item on a page

Note that the shared node of one menu may be a group node or an item node in the referenced submenu. For example, in the Home menu, Benefits is represented as a shared node that references `# {benefits_menu}`. In the Benefits menu, Benefits is the label for a group node that groups the `PaidTimeOff` item node and another shared node that references the insurance menu.

Examining the ADF Menu Model

The screenshot illustrates the configuration of an ADF Menu Model. At the top, an XML snippet shows the menu structure:

```

<?xml version="1.0" encoding="windows-1252" ?>
<menu xmlns="http://myfaces.apache.org/trinidad/menu">
    <groupNode id="groupNode_Insurance" idref="itemNode_Dental"
               label="Insurance">
        <itemNode id="itemNode_Dental" label="Dental" action="adfMenu_Dental"
                  focusViewId="/Dental"/>
        <itemNode id="itemNode_Health" label="Health" action="adfMenu_Health"
                  focusViewId="/Health"/>
    </groupNode>
</menu>

```

The 'action' attributes for the 'itemNode_Dental' and 'itemNode_Health' nodes are highlighted with red boxes. Below the XML is a 'Structure' view showing the 'insurance_menu.xml' file structure. To the right are two configuration dialogs:

- adfc-config-insurance.xml** (Control Flows tab):

| From Activity | From Outcome | To Activity |
|---------------|----------------|-------------|
| * | adfMenu_Dental | Dental |
| * | adfMenu_Health | Health |
- adfc-config-insurance.xml** (Managed Beans tab):

| Name * | Class * | Scope * |
|----------------|--|---------|
| insurance_menu | org.apache.myfaces.trinidad.model.XMLMenuModel | request |

A red arrow points from the 'To Activity' column in the first dialog to the 'To Activity' column in the second dialog.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Examining the ADF Menu Model

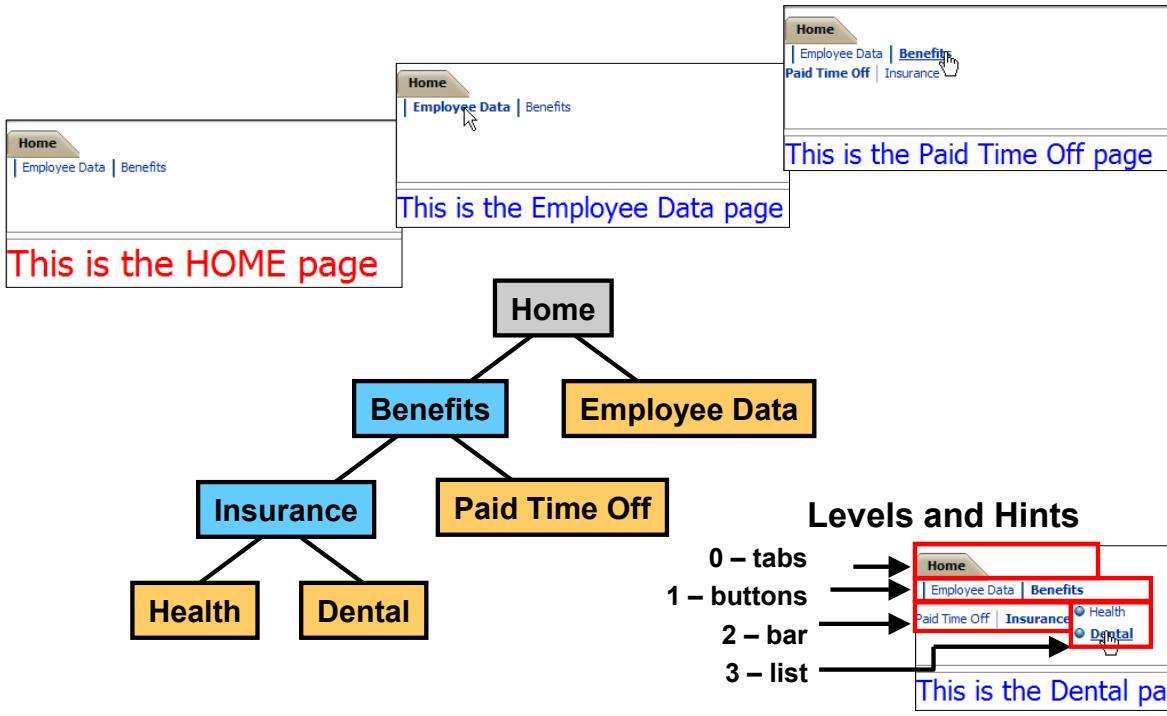
When you create an ADF Menu Model, an XML file is created for each menu. The XML file defines the nodes that you can use in navigational components, such as breadcrumbs.

If you examine the unbounded task flow, you can see that a global control flow rule, containing one or more control flow cases, has been added. The `from-outcome` properties of these control flow cases correspond to the actions in the item nodes named in the XML Menu Model.

Creating the ADF Menu Model also adds a managed bean of the class `org.apache.myfaces.trinidad.model.XMLMenuModel`. One of the managed properties of this bean is `source`, which is set to the XML file for the menu model.

Next you learn to use the menu model in navigational components.

Binding the Navigation Pane to an XML Menu Model



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Binding the Navigation Pane

To display the dynamic navigation controls similar to the one shown on the page at the right of the slide, you need to add a navigation pane for each level of the menu hierarchy. The navigation panes are identical except for the Level and Hint properties, so you can copy and paste them and change only those two properties. Set the following properties on the navigation panes:

- Hint: Typical usage is buttons for level 0, tabs for level 1, and bar for level 2. The example in the slide shows a fourth level, level 3, which has Hint set to list.
- Level: The top level is zero.
- Value: Set to the EL expression for the root menu, such as `#{root_menu}`
- Var (on the Data tab): Set to a name, such as `menuInfo`, that you refer to in the navigation item (added below) to get the needed data from the menu model

Now, add to the `nodeStamp` facet of the navigation pane a single navigation item. At run time, this stamps out the appropriate number of navigation items for the level of menu that is being displayed. The navigation items for all of the navigation panes should be identical, so you can copy and paste them. Set the following properties on the navigation item:

- Rendered: `#{<varvalue>.rendered}` – for example, if the Var property in the navigation pane is set to `menuInfo`, set Rendered to `#{menuInfo.rendered}`.
- Text: `#{menuInfo.label}`
- Icon: `#{menuInfo.icon}`

Binding the Navigation Pane (continued)

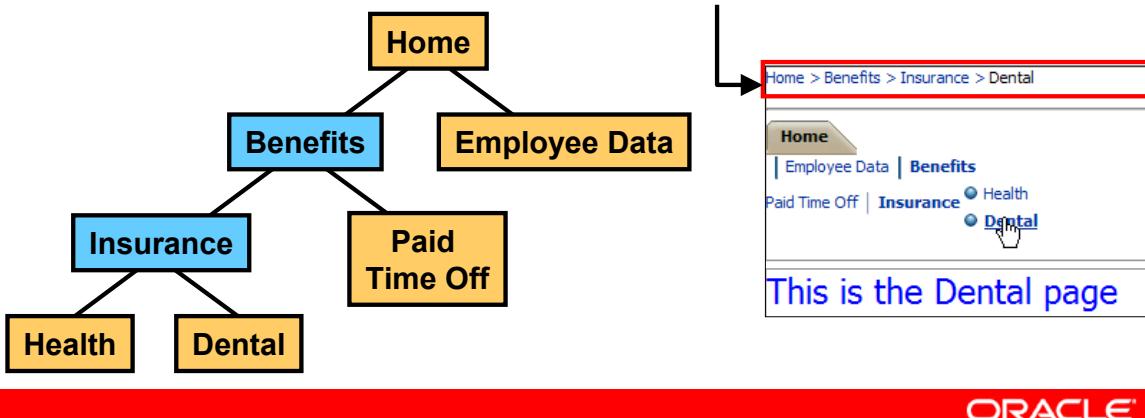
- Destination: # {menuInfo.destination}
- Action: # {menuInfo.doAction}
- Visible: # {menuInfo.visible}

After you have defined the navigation panes on one page, you can copy and paste them to all pages in the hierarchy. Alternatively, you can use a page fragment or a page template, as discussed in the lesson titled “Ensuring Reusability.”

Binding Breadcrumbs to an XML Menu Model

To link breadcrumbs to the menu model:

- Set the Var and Value properties on the breadcrumbs component
- Add a navigation item to the nodeStamp facet and set its properties just as for navigation pane's item
- Copy the breadcrumbs component to each page



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Binding Breadcrumbs to an XML Menu Model

In the example page hierarchy that we have discussed, to view Health information, the user drills from the Home page and selects Benefits, and then selects the Insurance page where two choices are presented, one of which is Health. The path of links starting from Home and ending at Health is known as the focus path in the tree, and can be represented on a page as breadcrumbs.

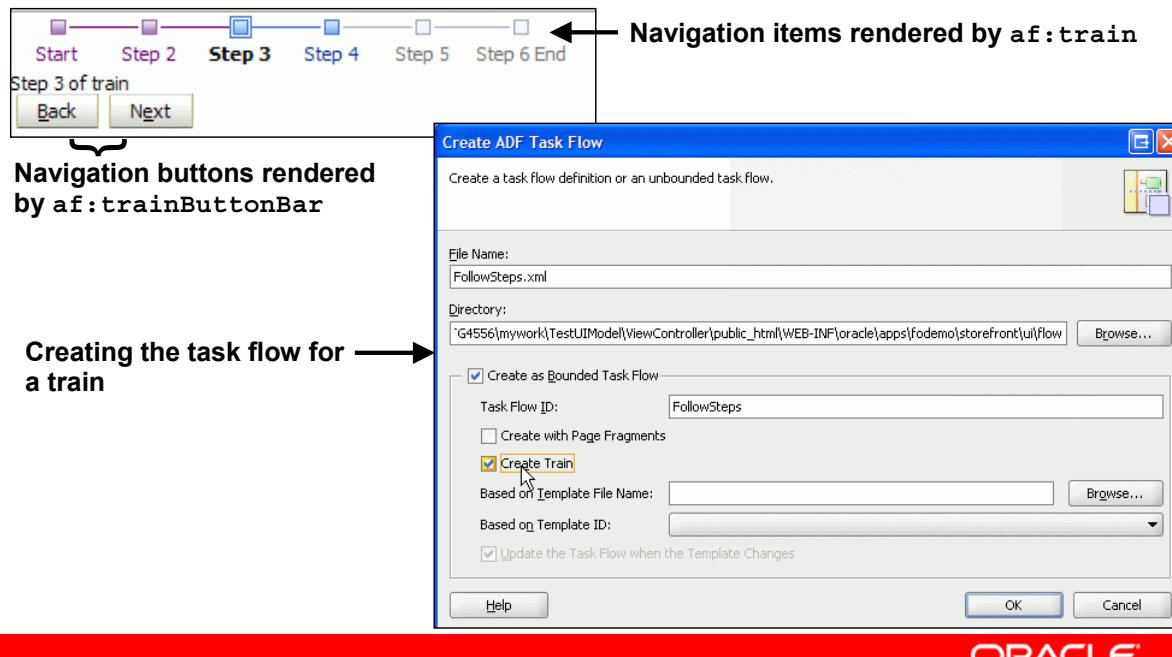
Using an XML Menu Model to display dynamic breadcrumbs is similar to the way you use the navigation pane. On any page, to show the user's current position in relation to the entire page hierarchy, you can use the af:breadcrumbs component with one af:commandNavigationItem component as a nodeStamp, to provide a path of links from the current page back to the root page (that is, the current nodes in the focus path). You can bind the value attribute of the af:breadcrumbs component to an XML Menu Model, and use one af:commandNavigationItem in the nodeStamp facet of af:breadcrumbs to stamp out the items for a page.

Set the Value and Var properties on the breadcrumbs component the same way as for the navigation pane—you can even use the same values for these properties. Then set the navigation item properties as for the navigation item in the navigation pane.

Note: You also can bind a tree to an XML Menu Model and use it on pages for navigation.

Defining a Sequence of Steps

A train is a specialized type of task flow with linear steps:



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Using a Train Component

If you have a set of pages that users should visit in a particular order, consider using the **af : train** component on each page to display a series of navigation items to guide users through the multistep process. Not only does a train display the number of steps in a multistep process, it also indicates the location of the current step in relation to the entire process.

Suppose you have eight steps in a multistep process, such as creating a new item in an inventory system. When configured properly, the **af : train** component would render all eight steps on a page, with each step represented as a train stop, and with all the stops connected by lines. Each train stop has an image (for example, a square block) with a label underneath the image. The slide shows an example of how this train could look on a page.

Each train stop corresponds to one step or one page in your multistep process. Users navigate the train stops by clicking an image or label, which causes a new page to display. Typically train stops must be visited in sequence, that is, a user cannot jump to step 3 if the user has not visited step 2.

As shown in the slide, a train is actually a specialized type of task flow with linear steps.

Note: Internationalization of the text displayed for a train stop is currently not supported.

Using a Train Component (continued)

As shown in the slide, the train component provides at least four styles for train stops:

- The current stop is indicated by a bold font style in the label, and a different image for the stop.
- Stops visited before the current stop are indicated by a different label font color and image color.
- The next stop immediately after the current stop appears enabled.
- Any other stops that have not been visited are disabled.

A train stop can include a subprocess train. That is, you can launch a child multistep process from a parent stop, and then return to the correct parent stop after completing the subprocess. Suppose stop #4 has a subprocess train containing three stops. When the user navigates to the first stop in the subprocess train, ADF Faces displays an icon representing the parent train before and after the subprocess train.

Adding Navigation Buttons

The `af:trainButtonBar` component may be used in conjunction with the `af:train` component to provide additional navigation items for the train, in the form of Back and Next buttons. The third example in the slide shows what a rendered `af:trainButtonBar` component could look like. The Back and Next buttons provided by `af:trainButtonBar` enable users to navigate only to the next or previous train stop from the current stop. You can also use the `af:trainButtonBar` component without `af:train`.

The train and train button bar work by having the `value` attribute bound to a train model of type `org.apache.myfaces.trinidad.model.MenuModel`. The train menu model contains the information needed to:

- Control a specific train behavior (that is, how the train advances users through the train stops to complete the multistep process)
- Dynamically generate the train stops, including the train stop labels, and the status of each stop (that is, whether a stop is currently selected, visited, unvisited, or disabled)

Creating a Train

Create ADF Task Flow

Create a task flow definition or an unbound task flow.

File Name: TrainFlow.xml

Directory: G:\mywork\TestUIModel\ViewController\pub

Create as Bounded Task Flow
Task Flow ID: TrainFlow

Create with Page Fragments

Create Train

Bind train

Select a train model to bind to. This page is a train stop in a taskflow, so the taskflow's train model has been selected for you.

Value: #{controllerContext.currentViewPort.taskFlowContext.trainModel} Bind...

Code Snippet:

```
<af:train value="#{controllerContext.currentViewPort.taskFlowContext.trainModel}" />
<af:trainButtonBar
    value="#{controllerContext.currentViewPort.taskFlowContext.trainModel}" />
<af:outputText value="Train Stop 1"/>
```

Diagram:

```
graph LR
    TS1[Train Stop 1] --> TS2[Train Stop 2]
    TS2 --> TS3[Train Stop 3]
```

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Creating a Train

To create a train, perform the following steps:

1. Create a bounded task flow, being sure to select the Create Train check box.
2. Drag the View components to the task flow—these become train stops. You can add other components to the task flow as needed, but flow between train stops is defined automatically.
3. Double-click each view to create the page, and drag a Train component to the page. By default, it binds to the train model of the task flow.
 - Drag a train button bar to the page, which also, by default, binds to the task flow's train model.
 - Add whatever content you want to appear on the page.

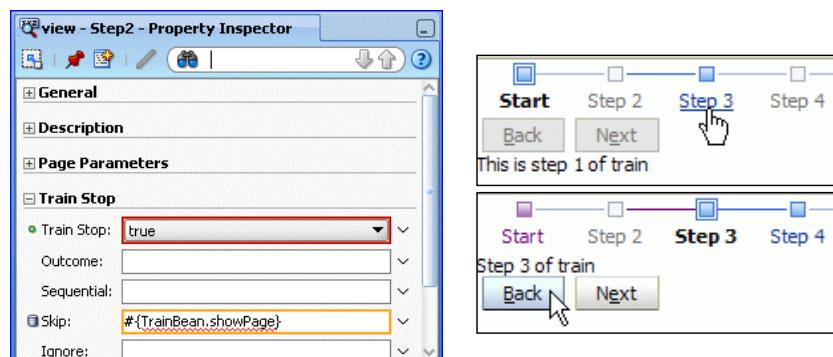
The bottom part of the slide shows a series of pages that were created by following the steps above, adding an output text component to each page that shows the current train stop. The running pages are also shown.

A train component has a nodeStamp facet, which accepts an af : commandNavigationItem component to act as a stamp for the train component, stamping out each train stop in the train model. When you bind the train to the task flow's train model, you do not explicitly define these navigation item components; they are implicit.

Skipping a Train Stop

Controlled by the `skip` property of the train stop:

- If `true`, users cannot navigate to that train stop
- Typically set to an expression that evaluates to `true` or `false`, such as managed bean method or property



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Skippping a Train Stop

You can set a train stop to be skipped by setting the `skip` property to `true`. You would typically set this property to an EL expression that evaluates to `true` or `false` to conditionally skip the train stop.

The example in the slide shows the Property Inspector for a view in the task flow of the train. The `skip` property is set to a managed bean method that is coded to perform some evaluation and conditionally return either `true` or `false`.

At run time, the user cannot navigate to a skipped train stop.

Summary

In this lesson, you should have learned how to:

- Implement command buttons and links
- Create menus
 - Menu bar
 - Pop-up
 - Context
- Use an ADF menu model
- Define navigation panes
- Use breadcrumbs
- Define trains



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 14 Overview: Using ADF Faces Navigation Components

This practice covers the following topics:

- Creating Buttons for Navigation
- Creating Links for Navigation
- Using Breadcrumbs
- Defining a Sequence of Steps
- Simplifying and Enhancing a Task Flow



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 14 Overview: Using ADF Faces Navigation Components

In this set of practices, you implement navigation on pages by creating buttons, links, breadcrumbs, and a train.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Osi S.R.L. use only

Achieving the Required Layout

15

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Use complex layout components
- Explain how to use ADF Faces skins
- Use dynamic page layout



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

One of the goals in Web development is to design attractive and usable pages. This lesson describes how to use complex layout components and styles to achieve a desired appearance. Students also learn to use dynamic page layout techniques.

Using ADF Faces Layout Components

You can use the following components to achieve the desired layout:

- af:spacer
- af:separator
- af:panelSplitter
- af:panelStretchLayout
- af:panelAccordion
- af:panelFormLayout
- af:panelTabbed
- af:showDetail
- af:panelDashboard
- af:panelGroupLayout
- af:panelCollection
- af:panelHeader
- af:showDetailHeader
- af:group
- af:panelList
- af:panelBox
- af:panelBorderLayout
- af:decorativeBox



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using ADF Faces Complex Layout Components

ADF Faces layout components include the following:

- spacer, separator: Add blank space and divider lines.
- panelSplitter, panelStretchLayout: Enable automatic component stretching on your pages.
- panelSplitter, panelAccordion: Create collapsible panes.
- panelFormLayout: Arrange items in columns or grids.
- panelTabbed: Create stacked tabs.
- showDetail: Hide and display groups of content.
- panelDashboard: Arrange child components (usually panelBox components) into a grid of columns and rows.
- panelGroupLayout: Enable automatic scroll bars on your pages, and arrange items horizontally or vertically.
- panelCollection: Add menus, toolbars, and status bars to data aggregation components such as tables, trees, and tree tables.
- panelHeader, showDetailHeader: Create titled sections and subsections.
- group: Group semantically related components.
- panelList, panelBox: Create styled lists and content boxes.

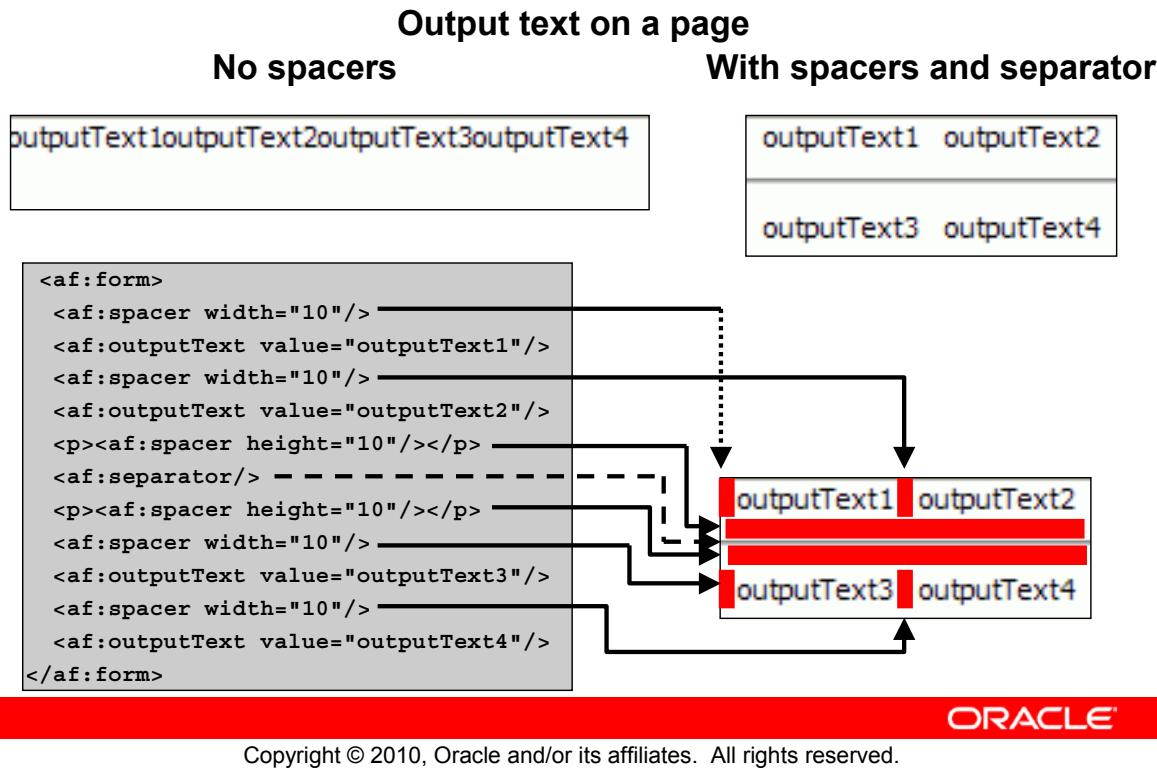
Using ADF Faces Complex Layout Components (continued)

`panelBorderLayout`, `panelStretchLayout`: Place items in fixed, peripheral areas around a central area.

`decorativeBox`: Create a container component whose facets use style themes to apply a bordered look to its children. This is typically used as a container for the `navigationPane` component that is configured to display tabs.

The following slides show how to use these components for various layout tasks. You can find some excellent ADF Faces Rich Client demos at: adfui.us.oracle.com.

Adding Spaces and Lines: Spacer and Separator



Adding Spaces and Lines

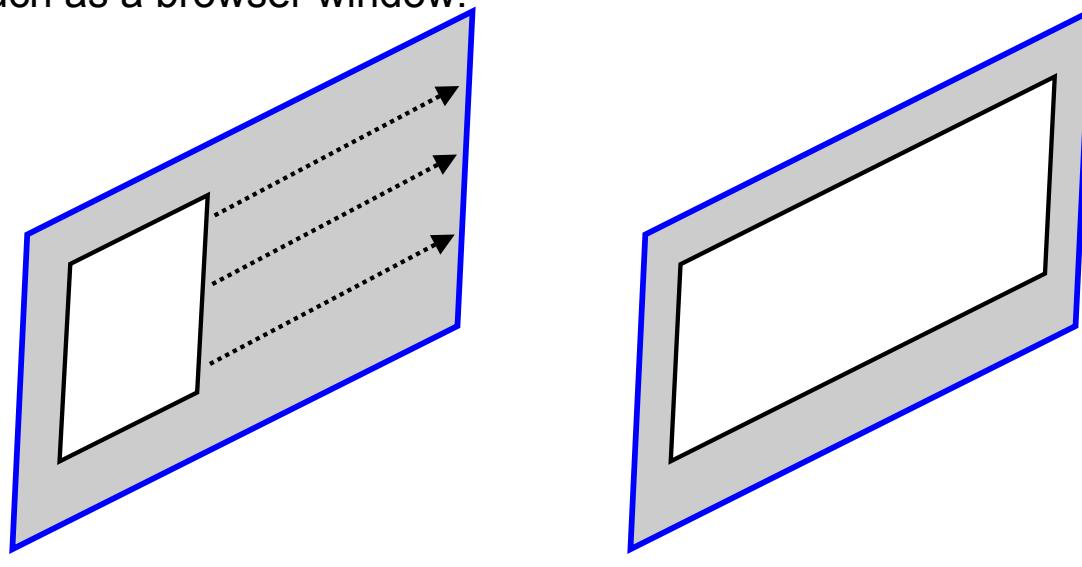
You can use the `af:spacer` component to add blank space on your pages between components, so that the page appears less cluttered than it would if all the components were right next to each other. You can include either or both vertical and horizontal space on a page by using the following attributes:

- `height`: The amount of vertical space to include on the page. Use `height` with `width` not set to create vertical spacer, surrounded with a paragraph tag so that it appears on its own line
- `width`: The amount of horizontal space to include between components. Use `width` with `height` not set to create horizontal spacer

The `af:separator` component creates a horizontal line on the page.

Stretching Components

Enable stretching so that a component fills the designated area, such as a browser window:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Stretching Components

By default, the `maximized` attribute on `af:document` is set to `true`. Upon rendering content, any outer margin, border, and padding are removed, and the body is stretched to fill the entire browser window.

However, this does not mean that setting `maximized` to `true` is sufficient when you want your page contents to fill a browser window. When the user resizes the browser window, the `af:document` component does not reposition and resize its child components accordingly.

If you want a component to fill an area, such as the browser or a designated part of another component, then you must enable stretching for that component. If a stretchable component is a root component (shown at the first level of a component hierarchy in the Application Navigator), then it stretches automatically.

Enabling Automatic Component Stretching: Panel Splitter or Panel Stretch Layout

Table on page:

| ProductId | ProductName | ListPrice |
|-----------|--------------------------|-----------|
| 9 | Nintendo DS | 129.99 |
| 14 | Bluetooth Phone Headset | 49.99 |
| 15 | Ipod Speakers | 89.99 |
| 16 | Creative Zen Vision V | 389.99 |
| 17 | Ipod Video 80Gb | 339.99 |
| 18 | Ipod Shuffle 1Gb | 99.99 |
| 19 | Ipod Video 30Gb | 249.99 |
| 20 | Ipod Video 60Gb | 399.99 |
| 21 | Ipod Nano 1Gb | 149.95 |
| 22 | Ipod Nano 2Gb | 199.95 |
| 10 | Zune 30Gb | 225.99 |
| 11 | RAZR Cellular Phone | 259.99 |
| 12 | Muvo Personal MP3 Player | 99.99 |
| 13 | Bluetooth Adaptor | 19.99 |
| 1 | Plasma HD Television | 1999.99 |
| 2 | PlayStation 2 Video Game | 199.95 |
| 3 | Treo 650 Phone/PDA | 299.99 |

Same table within
the stretched Panel
Stretch Layout
component:

| ProductId | ProductName | ListPrice | CategoryId | Description |
|-----------|--------------------------|-----------|------------|--------------------------------------|
| 9 | Nintendo DS | 129.99 | 7 | Lighter, brighter and more portable |
| 14 | Bluetooth Phone Headset | 49.99 | 6 | Streamlined and sophisticated |
| 15 | Ipod Speakers | 89.99 | 4 | The Portable Audio System |
| 16 | Creative Zen Vision V | 389.99 | 4 | The Creative Zen Vision V |
| 17 | Ipod Video 80Gb | 339.99 | 4 | Apple iPod - Continuity |
| 18 | Ipod Shuffle 1Gb | 99.99 | 4 | The Apple 1 GB Shuffle |
| 19 | Ipod Video 30Gb | 249.99 | 4 | Apple iPod - Continuity |
| 20 | Ipod Video 60Gb | 399.99 | 4 | Apple iPod - Continuity |
| 21 | Ipod Nano 1Gb | 149.95 | 4 | Includes: earbud headphones |
| 22 | Ipod Nano 2Gb | 199.95 | 4 | Includes: earbud headphones |
| 10 | Zune 30Gb | 225.99 | 4 | Zune is here. Design is here. |
| 11 | RAZR Cellular Phone | 259.99 | 6 | Thin is definitely in. A |
| 12 | Muvo Personal MP3 Player | 99.99 | 4 | The Creative Nomad |
| 13 | Bluetooth Adaptor | 19.99 | 6 | SyNNet's TBW-101UB |
| 1 | Plasma HD Television | 1999.99 | 4 | The TH-42PX60U 42-inch |
| 2 | PlayStation 2 Video Game | 199.95 | 7 | Whether you're a die-hard fan or not |
| 3 | Treo 650 Phone/PDA | 299.99 | 6 | The unlocked PalmOr |
| 4 | Treo 700w Phone/PDA | 399.99 | 6 | The PalmTm Treo 700w |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enabling Automatic Component Stretching

Some ADF Faces components, however, do reposition and resize their children when the browser size changes. Such components include `af:panelSplitter` and `af:panelStretchLayout`, which automatically use client-side geometry management to control the stretching of their child components. You do not have to write any code to enable the stretching—simply use either `af:panelSplitter` or `af:panelStretchLayout` to contain your page contents.

Although a geometry management component stretches its children, the component itself does not stretch automatically. So unless the geometry management component is the root component for your page contents, you must set a property to tell it to fill up the section of the window where it is displayed.

To stretch and position your outermost layout panel component, you can set the `maximized` property of `af:document` to `true` (the default). If the root component is stretchable, it then stretches, both vertically and horizontally, to fit the browser viewport.

To horizontally stretch any component that does not stretch automatically, set its `Width` property to `100%`. You should not, however, set the `Height` property to `100%`. If you want to stretch a component vertically, you should place it in a stretched component that automatically stretches its children.

Stretching a Table Column

Table with no stretched columns:

| ProductName | ListPrice | ProductId | CategoryId | Description |
|--------------------------|-----------|-----------|------------|---|
| Plasma HD Television | 1999.99 | 1 | 13 | The TH-42PX60U 42-inch Diagonal Plasma HDTV gives you deep black levels and vibrant colors. |
| PlayStation 2 Video Game | 199.95 | 2 | 7 | Whether you're a die-hard PlayStation 2 fan or a new player, this console has something for everyone. |
| Treo 650 Phone/PDA | 299.99 | 3 | 6 | The unlocked PalmOne Treo 650 boasts all the legendary features of its predecessor, plus a few surprises. |
| Treo 700w Phone/PDA | 399.99 | 4 | 6 | The Palm® Treo 700w smartphone delivers everything you need in a compact, easy-to-use package. |
| Tungsten E PDA | 195.99 | 5 | 4 | The sleek and powerful Palm Tungsten E handheld comes packed with features and a long battery life. |
| Bluetooth Phone Headset | 49.99 | 14 | 6 | Streamlined and sophisticated, the Bluetooth Headset H500 provides clear audio and a comfortable fit. |
| iPod Speakers | 89.99 | 15 | 4 | The Portable Audio System for iPod and iPod Photo allows you to listen to your music on the go. |
| Creative Zen Vision W | 389.99 | 16 | 4 | The Creative Zen Vision W Multimedia Player lets you take all your music with you. |

Table with last column stretched:

| ProductName | ListPrice | ProductId | CategoryId | Description |
|--------------------------|-----------|-----------|------------|---|
| Plasma HD Television | 1999.99 | 1 | 13 | The TH-42PX60U 42-inch Diagonal Plasma HDTV gives you deep black levels and vibrant colors. |
| PlayStation 2 Video Game | 199.95 | 2 | 7 | Whether you're a die-hard PlayStation 2 fan or a new player, this console has something for everyone. |
| Treo 650 Phone/PDA | 299.99 | 3 | 6 | The unlocked PalmOne Treo 650 boasts all the legendary features of its predecessor, plus a few surprises. |
| Treo 700w Phone/PDA | 399.99 | 4 | 6 | The Palm® Treo 700w smartphone delivers everything you need in a compact, easy-to-use package. |
| Tungsten E PDA | 195.99 | 5 | 4 | The sleek and powerful Palm Tungsten E handheld comes packed with features and a long battery life. |
| Bluetooth Phone Headset | 49.99 | 14 | 6 | Streamlined and sophisticated, the Bluetooth Headset H500 provides clear audio and a comfortable fit. |
| iPod Speakers | 89.99 | 15 | 4 | The Portable Audio System for iPod and iPod Photo allows you to listen to your music on the go. |
| Creative Zen Vision W | 389.99 | 16 | 4 | The Creative Zen Vision W Multimedia Player lets you take all your music with you. |



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

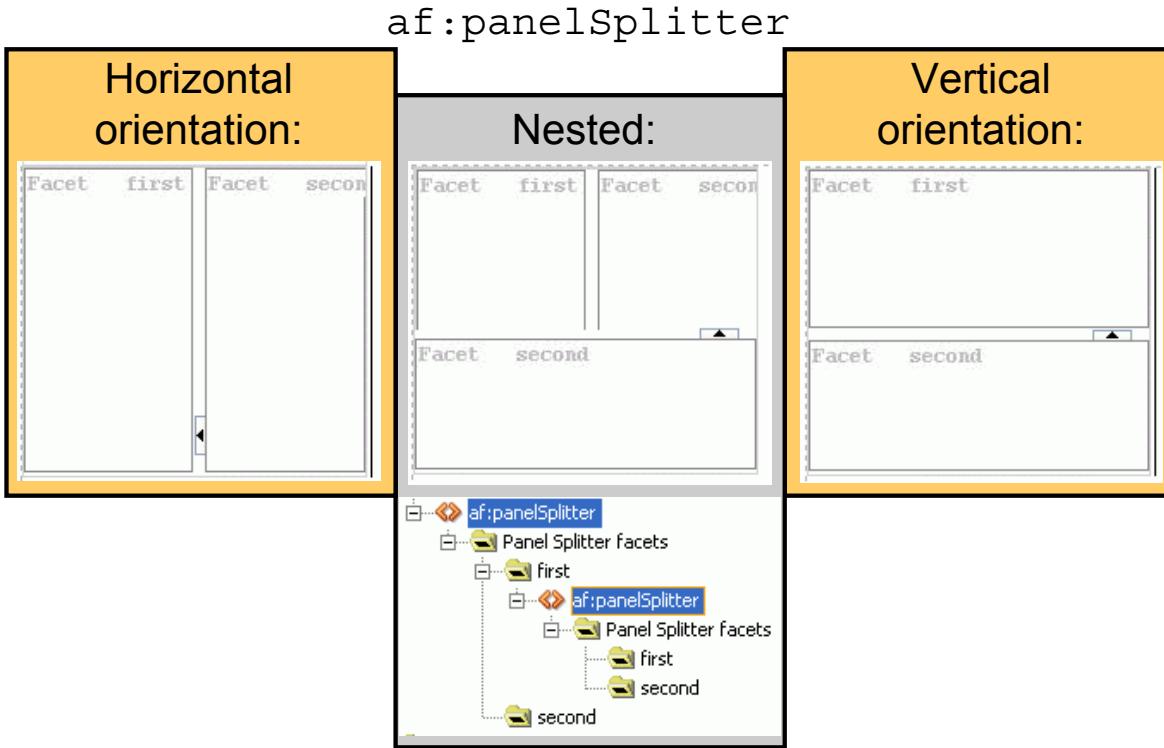
Stretching a Table Column

Certain components have attributes that relate to stretching their subcomponents. One of these is the ADF table.

You can stretch a table column to fill up the available space. On the Appearance tab of the table's Property Inspector, you can set the ColumnStretching property to one of the following values:

- none (default): Gives optimal performance
- blank: Inserts an empty blank column that is stretched. Use this so that the row's background colors will span the entire width of the table.
- last: Stretches the last column to fill up any unused space inside the viewport
- column : <id>: Stretches a specified column. Use the ID of the column that you want to stretch (requires that you explicitly set the column's ID property—the drop-down list for the property displays all columns whose ID has been set).

Creating Resizable Panes: Panel Splitter



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

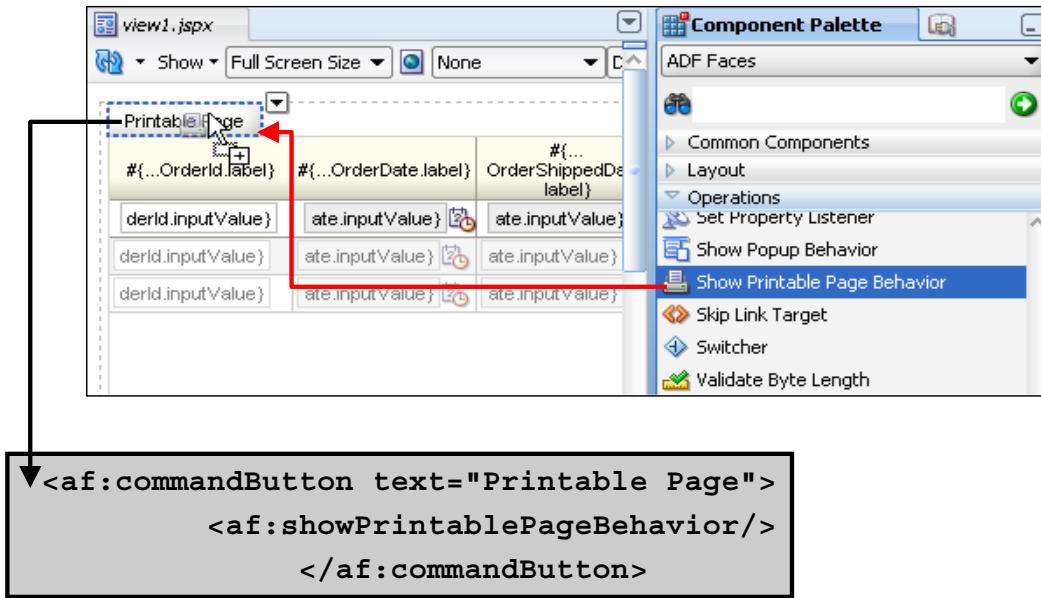
Creating Resizable Panes

When you have groups of unique content to present to users, consider using multiple panes separated by adjustable splitters. When viewing the page in a client browser, users can change the size of panes by dragging a splitter, and also choose to collapse or restore panes. When a pane is collapsed, its contents are hidden; when a pane is restored, the contents are displayed.

The `af:panelSplitter` component enables you to organize contents into two panes separated by an adjustable splitter. Clicking the arrow button on a splitter collapses a pane (and its contents) in the direction of the arrow. The value of the `orientation` attribute determines whether the panes are horizontal (default) or vertical. The pane contents are specified in the facets `first` and `second`. The slide shows a page divided into two horizontal panes (placed left to right), and another page divided into two vertical panes (placed top to bottom). It also shows a nested `panelSplitter`, created by a horizontal `panelSplitter` into the first facet of a vertical `panelSplitter`.

The `af:panelSplitter` component uses geometry management to stretch its child components at run time. This means when the user collapses one pane, the contents in the other pane are explicitly resized to fill up available space. If `af:panelSplitter` is the root component for your page contents, stretching occurs automatically.

Printing Layout Panel Content: Show Printable Page Behavior Operation



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Printing Layout Panel Content

The `af:showPrintablePageBehavior` operation creates a printable page that the user can print by using the browser's Print button. To allow printing, place the `af:showPrintablePageBehavior` component (an Operations component) within the page. The `showPrintablePageBehavior` tag is an operation, so you can drag it from the Operations components section of the Component Palette to a button or any other component that executes an operation.

If the page contains a layout component with panes, such as a panel splitter, place the button that contains the `af:showPrintablePageBehavior` component in the facet whose pane contents you want users to be able to print. For example, in a panel splitter, if the second facet is the main content area of the `af:panelSplitter` component, then insert the button that contains the `af:showPrintablePageBehavior` component into that facet. If you insert `af:showPrintablePageBehavior` anywhere outside of `af:panelSplitter`, the printed result may not contain all of the main content area.

Therefore, it is important to place the print operation within the `af:panelSplitter` facet whose pane contents users would normally want to print. If both facets need printing support, insert one `af:showPrintablePageBehavior` component into each facet. To print contents of both panes, the user then has to execute the print command one pane at a time.

Creating Collapsible Panes: Panel Splitter

Attributes:

| collapsed | positionedFromEnd | Behavior |
|-----------|-------------------|---|
| true | false | First pane hidden; second pane stretches |
| true | true | Second pane hidden; first pane stretches |
| false | true | Both panes displayed, with splitterPosition governing the size of the second pane and the splitter arrow pointed toward the second pane |
| false | false | Both panes displayed, with splitterPosition governing the size of the first pane and the splitter arrow pointed toward the first pane |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

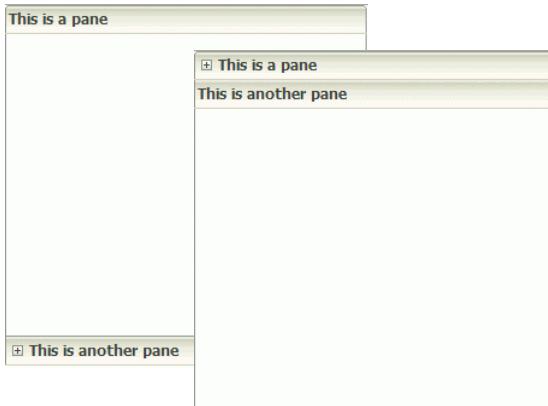
Creating Collapsible Panes with the Panel Splitter Component

Another feature of the panel splitter component is that users can collapse or restore its panes. When a pane is collapsed, the pane contents are hidden; when a pane is restored, the contents are displayed. The `collapsed` attribute on `af:panelSplitter` determines whether the splitter is in a collapsed (hidden) state. By default, the `collapsed` attribute is `false`, which means both panes are displayed. When the user clicks the arrow button on the splitter, `collapsed` is set to `true` and one of the panes is hidden.

ADF Faces uses the `collapsed` and `positionedFromEnd` attributes to determine which pane (that is, the first or second pane) to hide (collapse) when the user clicks the arrow button on the splitter. When `collapsed` is `true` and `positionedFromEnd` is `false`, the first pane is hidden and the second pane stretches to fill up the available space. When `collapsed` is `true` and `positionedFromEnd` is `true`, the second pane is hidden instead. Visually, the user can know which pane will be collapsed by looking at the direction of the arrow on the button. When the user clicks the arrow button on the splitter, the pane collapses in the direction of the arrow. The `splitterPosition` and `collapsed` attributes are persistable; that is, when the user moves the splitter or collapses a pane, ADF Faces can implicitly persist the attribute value changes for the component.

Creating Collapsible Panes: Panel Accordion

With default settings



Characteristics of the Panel Accordion component:

- Does not automatically stretch its children
- Panes cannot be resized at run time—only expand or contract
- Panes defined by showDetailItem

```
<af:panelAccordion>
    <af:showDetailItem text="This is a pane"/>
    <af:showDetailItem text="This is another pane"/>
</af:panelAccordion>
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Collapsible Panes with the Panel Accordion Component

Another component that enables you to create collapsible panes is panel accordion. A series of af : showDetailItem components inside af : panelAccordion make up the accordion panes, with one af : showDetailItem component corresponding to one pane. The pane contents are the child components inside each af : showDetailItem component.

You can use more than one af : panelAccordion component on a page, typically in different areas of the page, or nested. After adding the af : panelAccordion component, insert a series of af : showDetailItem components to provide the panes, using one af : showDetailItem component for one pane. Then insert components into each af : showDetailItem component to provide the pane contents.

Note: The af : showDetailItem component has a StretchChildren property that you can set to stretch the first child of the af : showDetailItem component, if the child is a stretchable component.

Panel Accordion Overflow

Automatic overflow icons to display content out of view

The screenshot shows a complex ADF Faces interface. At the top left, there's a navigation bar with a 'Browse Products' link. Below it is a table showing categories like Media, Electronics, and Office. To the right is a large empty space. Further down is a tree view labeled 'Show Subcategories'. To the right of the tree is a grid of products with columns for Description, SupplierId, ProductName, ListPrice, and ProductId. At the bottom right of the main content area, there's a form for a 'Selected Product' with fields for Description, AdditionalInfo, CostPrice, SupplierId, ProductName, ListPrice, and ProductId. Below the form are buttons for 'Browse Categories' and 'Show Subcategories'. Red arrows and circles highlight the overflow icons at the top-left and bottom-right corners of the main content area.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Panel Accordion Overflow

At run time, when the available browser space is less than the space needed to display expanded pane contents, ADF Faces automatically displays overflow icons that enable users to select and navigate to those panes that are out of view. The slide shows overflow icons at the upper-left (first example) portion and at the lower-right (second example) portion. When the user clicks the overflow icon, ADF Faces displays the overflow pop-up menu, as shown in the slide, for the user to select and navigate to a pane that is out of view.

Setting Panel Accordion Properties

`<af:showDetailItem
text="Browse Products"
flex="1">`

`<af:showDetailItem
text="Selected Product"
flex="2">`

With discloseMany="true" discloseNone="true"

This is a pane
This is another pane
This is a pane
This is another pane

| Description | SupplierId | ProductName | ListPrice | ProductId |
|------------------------------|--------------------------|-------------|-----------|-----------|
| Zune is here. Design 123 | Zune 30Gb | 225.99 | 10 | |
| Thin is definitely in. A 112 | RAZR Cellular Phone | 259.99 | 11 | |
| The Creative Nomad 123 | Muvo Personal MP3 F99.99 | | 12 | |
| SyNET's TBW-101UB 101 | Bluetooth Adaptor | 19.99 | 13 | |
| Includes: earbud head 106 | Ipod Nano 4Gb | 249.95 | 23 | |

Selected Product

* Description: Zune is here. Designed around the principles of sharing, discovery and commu...
 AdditionalInfo: The Digital Media Player reinvented. With the new Microsoft Zune you can wir...
 CostPrice: 100
 * SupplierId: 123
 ProductName: Zune 30Gb
 * ListPrice: 225.99
 * ProductId: 10
 CategoryId: Audio and Video
 * MinPrice: 169.99
 WarrantyPeriodMonths: 3
 * ShippingClassCode: CLASS2

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Setting Panel Accordion Properties

The following properties govern the behavior of the panel accordion at run time:

- `discloseNone`: Allows all panes to be collapsed at once (default is `false`)
- `discloseMany`: Allows multiple panes to be expanded at once (default is `false`)

The following properties of `showDetailItem` affect the behavior of the accordion panes:

- `flex`: Specifies a nonnegative integer that determines how much space is distributed among the `af:showDetailItem` components of one `af:panelAccordion`. By default, `flex` is 0 (zero), that is, the pane contents of each `af:showDetailItem` are inflexible. To enable flexible contents in a pane, specify a `flex` number larger than 0.
- `inflexibleHeight`: Specifies the number of pixels a pane will use. Default is 100 pixels. This means if a pane has a `flex` value of 0 (zero), ADF Faces will use 100 pixels for that pane, and then distribute the remaining space among the nonzero panes. If the contents of a pane cannot fit within the `af:panelAccordion` container given the specified `inflexibleHeight` value, ADF Faces automatically pushes out nearby contents into overflow menus, as described in the previous slide.
- `stretchChildren`: When set to `first`, stretches a single child component. However, the child component must allow stretching.

Setting Panel Accordion Properties (continued)

The `discloseMany` attribute governs how many panes can be visible at once.

- Set to `true` if you want users to be able to expand and see the contents of more than one pane at the same time.
- By default, `discloseMany` is `false`. This means that only one pane can be expanded at any one time. For example, suppose there is one expanded pane A and one collapsed pane B when the page first loads. If the user expands pane B, pane A will be collapsed, as only one pane can be expanded at any time.

The `discloseNone` attribute governs whether users can collapse all panes.

- Set the `discloseNone` attribute to `true` to enable users to collapse all panes.
- By default, `discloseNone` is `false`. This means one pane must remain expanded at any time.

You can add toolbars and toolbar buttons in the pane headers. To add toolbar buttons to a pane, insert the `af:toolbar` component into the `toolbar` facet of the `af:showDetailItem` component that defines that pane. Then insert the desired number of `af:commandToolbarButton` components into the `af:toolbar` component. Although the `toolbar` facet is on `af:showDetailItem`, it is the parent component `af:panelAccordion` that renders the toolbar and its buttons.

To allow users to print the contents of a single pane, place the `af:showPrintablePageBehavior` component (wrapped in `af:commandButton`) within the `af:showDetailItem` whose pane contents you want users to be able to print.

Arranging Items in Columns or Grids: Panel Form Layout

Use `af:panelFormLayout` with labels above or below the text:

Right-Aligned Labels and Left-Aligned Fields in a Form

Enter or select a date

File to upload

Enter password

Check all that apply Coffee
 Tea
 Low-fat milk
 Half-and-half

Pick one only None

Labels Above Fields in a Form

Label for InputText

Check All that Apply Coffee
 Tea
 Milk

Use `rows` and `maxColumns` attributes to arrange items in the form:

Form with `rows` set to 4

| | | |
|--|---|--|
| CostPrice <input type="text" value="100"/> | * ProductId <input type="text" value="10"/> | * ShippingClassCode <input type="text" value="CLASS2"/> |
| * SupplierId <input type="text" value="123"/> | CategoryId <input type="text" value="Audio and Video"/> | CategoryName <input type="text" value="Audio and Video"/> |
| ProductName <input type="text" value="Zune 30Gb"/> | * MinPrice <input type="text" value="169.99"/> | CategoryId <input type="text" value="4"/> |
| * ListPrice <input type="text" value="225.99"/> | WarrantyPeriodMonths <input type="text" value="3"/> | CategoryDescription <input type="text" value="Audio and Video"/> |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Arranging Items in Columns or Grids

The `af:panelFormLayout` component enables you to lay out multiple form input components such as input fields and selection list fields in one or more columns. They can be arranged in one of two ways:

- With the field labels right-aligned and the fields left-aligned, as shown in the first example in the slide
- With the labels above the fields, as shown in the second example. To place the labels above the fields, set the `labelAlignment` attribute on `af:panelFormLayout` to `top`.

When you nest an `af:panelFormLayout` component inside another, by default, the label alignment in the nested layout is `top`.

The following attributes determine the number of rows and columns to display the child components:

- `rows`: The number of rows after which a new column will start. The default is `2,147,483,647 (Integer.MAX_VALUE)`, which means all visible, rendered child components display in a single column. When `rows` is a nondefault value, the number of columns is dependent on the number of rendered children. For example, if `rows` is set to 6 and there are 7 to 12 rendered children, the list is displayed in two columns.

Arranging Items in Columns or Grids (continued)

- `maxColumns`: The maximum number of columns to display the child components. The default is 3 (2 on PDAs). For nested panel form layout components, the inner component's `maxColumns` value is always 1.

The row-column split is based solely on the number of rendered children and not on their size. If the number of rendered children does not fit in the number of rows and columns specified, ADF Faces automatically increases the number of rows to accommodate the number of children.

To add content below the children input components, insert the desired component into the `footer` facet of `af:panelFormLayout`. Facets accept only one child. If you have to insert more than one `footer` child, use `af:panelGroupLayout` or `af:group` to wrap the `footer` children.

Creating Stacked Tabs: Panel Tabbed with Show Detail Item

Use `af:panelTabbed`; use `af:showDetailItem` for each tab:

```
<af:panelTabbed position="both">
    <af:showDetailItem text="Categories">
        <!-- Root categories table here -->
    </af:showDetailItem>
    <af:showDetailItem text="Subcategories">
        <!-- Subcategories table here -->
    </af:showDetailItem>
    <af:showDetailItem text="Products">
        <!-- Products table here -->
    </af:showDetailItem>
</af:panelTabbed>
```

The screenshot shows a user interface with three tabs at the top: 'Categories', 'Subcategories', and 'Products'. The 'Categories' tab is currently selected, displaying a table with three rows: Media (CategoryName), Books, Music, and More (CategoryDescription), and 1 (CategoryId). The 'Subcategories' tab shows a table with one row: Electronics (CategoryName), Consumer Electronics (CategoryDescription), and 3 (CategoryId). The 'Products' tab shows a table with one row: Office (CategoryName), Office Supplies (CategoryDescription), and 2 (CategoryId). Below the tabs, there is a red footer bar with the 'ORACLE' logo.

| Categories | Subcategories | Products |
|-------------|------------------------|----------|
| Media | Books, Music, and More | 1 |
| Electronics | Consumer Electronics | 3 |
| Office | Office Supplies | 2 |

| Categories | Subcategories | Products |
|-------------|------------------------|----------|
| Electronics | Consumer Electronics | 3 |
| Office | Office Supplies | 2 |
| Media | Books, Music, and More | 1 |

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Stacked Tabs

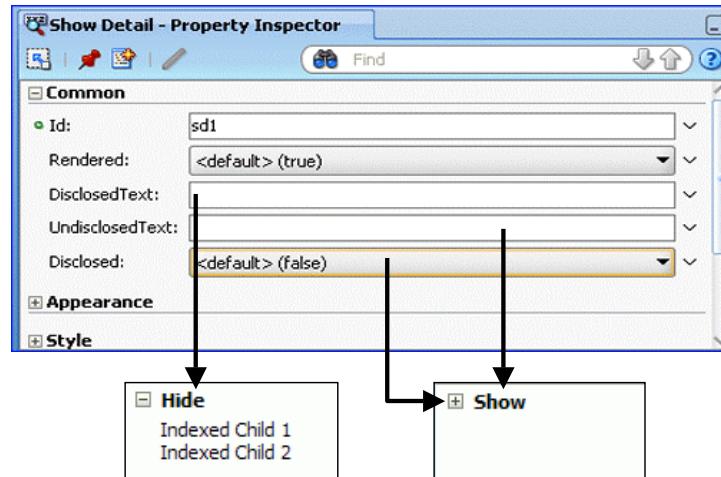
Using `af:panelTabbed` to create tabbed panes is similar to using `af:panelAccordion` to create accordion panes. After adding an `af:panelTabbed` component, insert a series of `af:showDetailItem` components to provide the tabbed pane contents for display.

To display and hide contents using PanelTabbed, perform the following steps:

1. Add the `af:panelTabbed` component to the JSF page.
2. Set the `position` attribute to `below` if you want the tabs to be rendered below the contents in the display area.
By default, `position` is `above`. This means the tabs are rendered above the contents in the display area. The other acceptable value is `both`, where tabs are rendered above and below the display area.
3. To add a tabbed pane, insert the `af:showDetailItem` component inside `af:panelTabbed`. You can add as many tabbed panes as you want.
4. To add contents for display in a pane, insert the desired child components into each `af:showDetailItem` component.

The example in the slide shows three tabs, positioned at both top and bottom. Each tab is defined by a show detail item component, each of which contains a table as content for the tab.

Hiding and Displaying Groups of Content: Show Detail



```
<af:showDetail>
    <af:panelGroupLayout layout="vertical">
        <af:outputText value="Indexed Child 1"/>
        <af:outputText value="Indexed Child 2"/>
    </af:panelGroupLayout>
</af:showDetail>
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Hiding and Displaying Groups of Content

You can use the `af:showDetail` component to display and hide groups of content. Its child components can be displayed or hidden at run time.

To display or hide contents using `af:showDetail`, perform the following steps:

1. Add the `af:showDetail` component to the JSF page and set its properties:
 - `disclosed`: Set to `true` if you want the component to show its children.
 - `disclosedText`: Set to the label you want to display next to the toggle icon when the contents are disclosed (shown). By default, the label is “Hide.”
 - `undisclosedText`: Set to the label you want to display next to the toggle icon when the contents are undisclosed (hidden). The default label is “Show.”

Note: If you specify a value for either `disclosedText` or for `undisclosedText`, but not for both, then that value is used for both the disclosed state and undisclosed state. Instead of using text specified in `disclosedText` and `undisclosedText`, you could use the prompt facet to add a component that will render next to the toggle icon.
2. Insert the child components for display into `af:showDetail`. You can wrap the children in, for example, `af:panelGroupLayout`, to achieve the desired layout.

Hiding and Displaying Groups of Content (continued)

An `org.apache.myfaces.trinidad.event.DiscloserEvent` is raised when the user clicks the toggle icon. You can perform special handling of this event by binding the component's `disclosureListener` attribute to a `DisclosureListener` method in a backing bean. The `DisclosureListener` method must be a public method with a single `DisclosureEvent` event object and a `void` return type, as shown in the following example:

```
public void some_disclosureListener(DisclosureEvent  
disclosureEvent) {  
    // Add event handling code here  
}
```

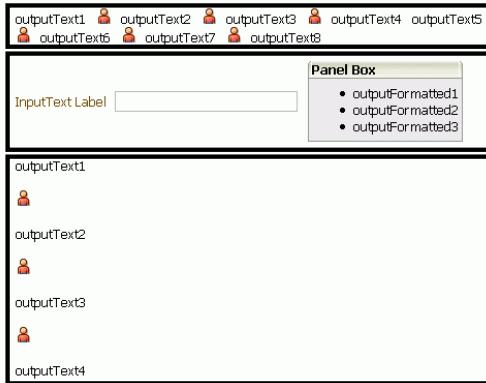
By default, `DisclosureEvent` events are usually delivered in the Invoke Application phase, unless the component's `immediate` attribute is set to `true`. When `immediate` is `true`, the event is delivered in the earliest possible phase, usually the Apply Request Values phase.

On the client-side component, the `AdfDisclosureEvent` is fired. The event root for the client `AdfDisclosureEvent` is set to the event source component; only the event for the pane whose `disclosed` attribute is `true` is sent to the server.

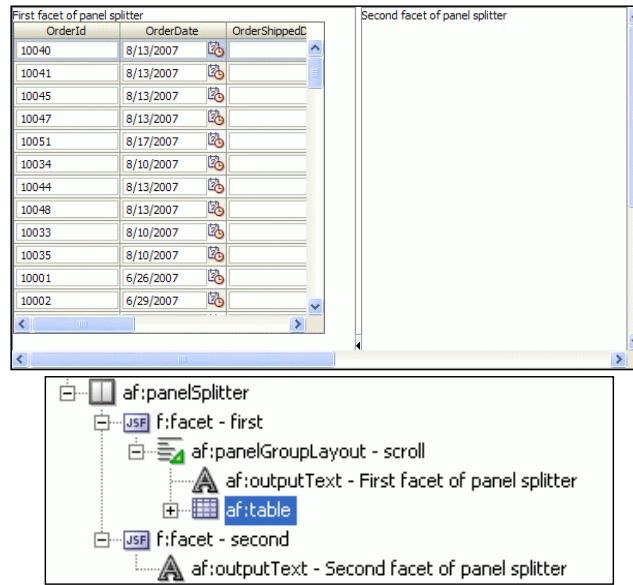
You learn more about events in the lesson titled “Responding to Application Events.”

Arranging Items Horizontally or Vertically, with Scrollbars: Panel Group Layout

Panel group layout arrangements



Panel group layout inside stretched panel splitter does not stretch its children



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Arranging Items Horizontally or Vertically, with Scrollbars

The `af:panelGroupLayout` component enables you to arrange a series of child components vertically or horizontally without wrapping, or consecutively with wrapping, as shown in the first example in the slide. The `layout` attribute value determines the arrangement of the children.

In all arrangements, each pair of adjacent child components can be separated by a line or white space using the `separator` facet on `af:panelGroupLayout`.

When using the horizontal layout, the children can also be vertically or horizontally aligned. For example, you can make a short component beside a tall component align at the middle, as shown in the second layout in the slide.

Unlike `af:panelSplitter` or `af:panelStretchLayout`, the `af:panelGroupLayout` component does not stretch its children. You can use `af:panelGroupLayout` in situations where you are already using `af:panelSplitter` or `af:panelStretchLayout` as the root component for the page, and you have a large number of child components to flow normally, but are not to be stretched, as in the second example in the slide.

Arranging Items Horizontally or Vertically, with Scrollbars (continued)

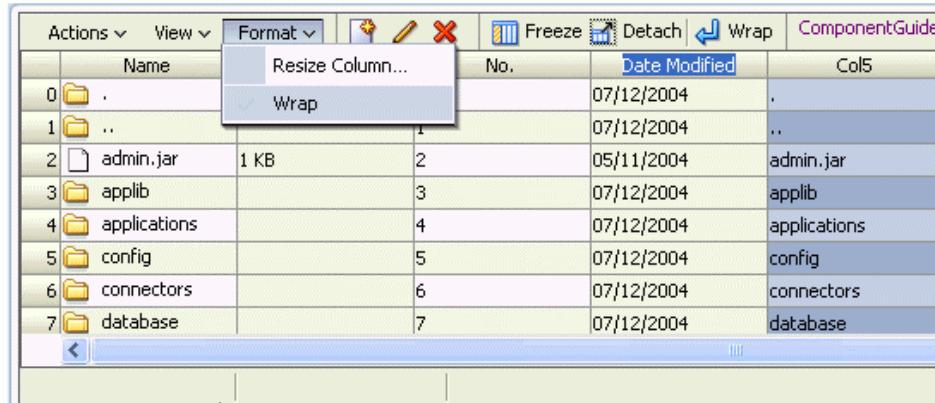
To provide scrollbars when flowing the child components, wrap the children in the `af:panelGroupLayout` component with its `layout` attribute set to `scroll`, and then place the `af:panelGroupLayout` component inside the `af:panelSplitter` or `af:panelStretchLayout` facet. The second example in the slide shows a panel splitter that is stretched. It contains a panel group layout with its `layout` attribute set to `scroll`. Although the child table is not stretched, it does display scrollbars to enable users to see all of its content.

When `layout` is set to `scroll` on `af:panelGroupLayout`, ADF Faces automatically provides a scrollbar at run time when the contents contained by the `af:panelGroupLayout` component are larger than the `af:panelGroupLayout` itself. You do not have to write any code to enable the scrollbars, or set any inline styles to control the overflow.

For example, when you use layout components such as `af:showDetail` that enable users to expand and collapse children contents, you do not have to write code to show the scrollbars when the contents expand, and to hide the scrollbars when the contents collapse. Simply wrap the `af:showDetail` component inside an `af:panelGroupLayout` component, and set the `layout` attribute to `scroll`. To get the layout you desire for the `af:showDetail` children, you can use nested `af:panelGroupLayout` components.

Displaying Table Menus, Toolbars, and Status Bars: Panel Collection

Use panelCollection for table with menus and toolbars:



The screenshot shows a table component within an Oracle ADF application. The table has columns for Name, No., Date Modified, and Col5. The first row's Name column contains a folder icon. A context menu is open over this icon, with 'Wrap' highlighted. The menu bar above the table includes 'Actions', 'View', 'Format', and various toolbar icons. The status bar at the bottom right displays the Oracle logo.

| Name | No. | Date Modified | Col5 |
|----------------|-----|---------------|--------------|
| 0 . | 1 | 07/12/2004 | . |
| 1 .. | 2 | 07/12/2004 | .. |
| 2 admin.jar | 3 | 05/11/2004 | admin.jar |
| 3 applib | 4 | 07/12/2004 | applib |
| 4 applications | 5 | 07/12/2004 | applications |
| 5 config | 6 | 07/12/2004 | config |
| 6 connectors | 7 | 07/12/2004 | connectors |
| 7 database | | | database |

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Displaying Table Menus, Toolbars, and Status Bars

You can use the af:panelCollection component to add menus, toolbars, and status bars to data aggregation components such as tables, trees, and tree tables. You would create the af:panelCollection component, and then add the table, tree, or tree table component as a direct child of panelCollection.

The af:panelCollection tag contains a menu facet for af:menu, a toolbar facet for af:toolbar, a secondaryToolbar facet for another af:toolbar, and a statusbar facet. The table, tree, or tree table is added after the last facet.

The default top-level menu and toolbar items depends on the child component being used:

- Tables and trees: The default top-level menu is View.
- Tables and tree table: The default toolbar menu is Detach.
- Tree and tree table (when the pathStamp is used): The toolbar buttons Go Up, Go To Top, and Show as Top also appear.

Displaying Table Menus, Toolbars, and Status Bars (continued)

The example shown in the slide has the following source code:

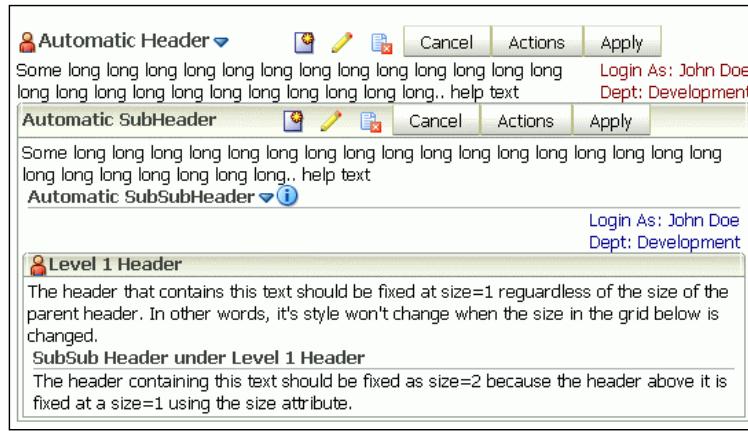
```
<af:panelCollection binding="#{editor.component}">
  <f:facet name="menus">
    <af:menu text="Actions">
      <af:commandMenuItem text="Add..." />
      <af:commandMenuItem text="Create..." />
      <af:commandMenuItem text="Update..." disabled="true"/>
      <af:commandMenuItem text="Copy"/>
      <af:commandMenuItem text="Delete"/>
      <af:commandMenuItem text="Remove" accelerator="control A"/>
      <af:commandMenuItem text="Preferences"/>
    </af:menu>
  </f:facet>
  <f:facet name="toolbar">
    <af:toolbar>
      <af:commandToolbarButton shortDesc="Create" icon="/new_ena.png">
      </af:commandToolbarButton>
      <af:commandToolbarButton shortDesc="Update" icon="/update_ena.png">
      </af:commandToolbarButton>
      <af:commandToolbarButton shortDesc="Delete" icon="/delete_ena.png">
      </af:commandToolbarButton>
    </af:toolbar>
  </f:facet>
  <f:facet name="secondaryToolbar"> </f:facet>
  <f:facet name="statusbar">
    <af:toolbar>
      <af:outputText id="statusText" ... value="Custom Statusbar Message"/>
    </af:toolbar>
  </f:facet>
  <af:table rowSelection="multiple" columnSelection="multiple" ...
    <af:column ...
    </af:column>
```

The panelCollection component is a naming container. If you want to add its contained component, such as a table, as partialTriggers (described later in this lesson), you must use its fully qualified name. For example, if you have a panelCollection component with id=entriesCollection and a contained table with id=deptTable, and you want to update a panelForm on the selection change of the table, you would declare the panelForm's partialTriggers as:

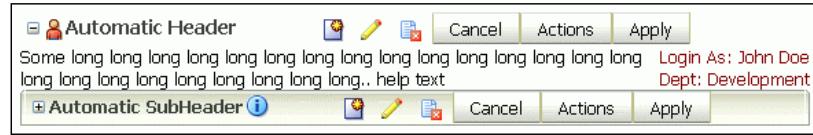
```
<panelForm partialTriggers="entriesCollection:deptTable">
```

Creating Titled Sections and Subsections: Panel Header

Panel header component with sections and subsections:
af : panelHeader



Show detail header component with sections that expand or collapse: **af : showDetailHeader**



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Titled Sections and Subsections

You may want to divide a page into sections and subsections, starting each section or subsection with a header. The **af : panelHeader** component enables you to create sections and subsections with a label and an icon at the top of each section or subsection header, as shown in the first example in the slide.

To enable users to toggle the display of contents under each section or subsection header, use the **af : showDetailHeader** component instead. This is similar to the **af : panelHeader** component, except that it renders a toggle icon that enables users to hide or display the section or subsection contents. The second example in the slide shows a top section with its contents displayed, and a subsection with its contents hidden. Partial page rendering (PPR), which you learn about later in this lesson, is automatically used to refresh a section of the page when the user selects to hide or show contents under the header.

The **disclosed** attribute on **af : showDetailHeader** specifies whether to show or hide the contents under its header. The default is **false**, which means that the contents are hidden.

Grouping Related Components: Group

Use af:group to:

- Add multiple components to a facet
- Group related child components

Grouped Components in PanelFormLayout

The screenshot shows a web form with the following structure:

- A date picker labeled "Pick a date".
- A section labeled "Select all that apply" containing five checkboxes: Coffee, Cream, Low-fat Milk, Sugar, and Sweetener.
- A text area labeled "Special instructions".
- A file upload section labeled "File to upload" with a "Browse..." button.
- A password input field labeled "Enter passcode".
- A text area labeled "Comments".

```
<af:panelFormLayout>
  <af:inputDate label="Pick a date"/>
  <!-- first group -->
  <af:group>
    <af:selectManyCheckbox label=
      "Select all that apply">
      <af:selectItem label="Coffee" value="1"/>
      //other select items
    </af:selectManyCheckbox>
    <af:inputText label="Special instructions"
      rows="3"/>
  </af:group>
  <!-- Second group -->
  <af:group>
    <af:inputFile label="File to upload"/>
    <af:inputText label="Enter passcode"/>
  </af:group>
  <af:inputText label="Comments" rows="3"/>
  <af:spacer width="10" height="15"/>
  <f:facet name="footer"/>
</af:panelFormLayout>
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Grouping Semantically Related Components

The af:group component aggregates or groups together child components that are related semantically. The af:group component does not provide any layout for its children. When used on its own, the af:group component does not render anything; only the child components inside af:group render at run time.

You can use any number of af:group components to group related components together. For example, you might want to group some of the input fields in a form layout created by af:panelFormLayout, as shown in the example in the slide that groups two sets of child components inside af:panelFormLayout.

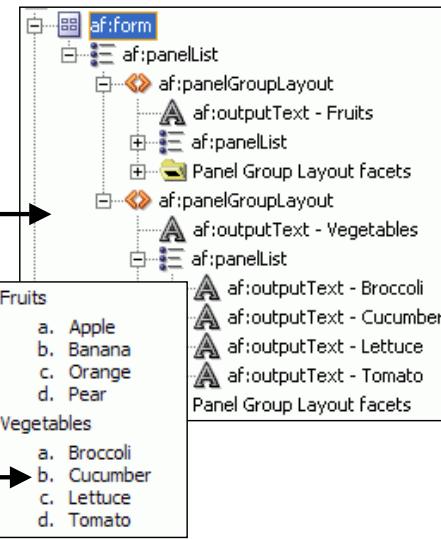
When the af:group component is used as a child in a parent component that does provide special rendering for af:group children, then visible separators, such as bars or dotted lines, display around the children of each af:group. For example, af:panelFormLayout and af:toolbar support special rendering for af:group children. The example in the slide shows how at run time the af:panelFormLayout component renders dotted separator lines before and after the first and second af:group of child components. Children inside af:group are never split across a column on a form.

The af:group component is especially useful when you need to group components under a facet, because a facet may have only one child.

Displaying a Bulleted List: Panel List

- Use `af:panelList` to display a bulleted list in one or more columns.
 - To create columns, nest inside `af:panelFormLayout` and set the `rows` property:
- | | | |
|--------------|---------|----------|
| • Apple | • Date | • Orange |
| • Banana | • Guava | • Pear |
| • Cantaloupe | • Kiwi | |
| • Cherry | • Lemon | |
- Create a list hierarchy with nested `af:panelList` components:
 - Specify the bullet style by setting `list-style-type` to:

| | |
|----------|---------------|
| – disc | – decimal |
| – square | – lower-alpha |
| – circle | – upper-alpha |



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Displaying a Bulleted List

You can use `af:panelList` to display a bulleted list in one or more columns. Use one `af:panelList` component to create each list of items. By default, all rendered child components of `af:panelList` are displayed in a single column. You can surround the child components in a Panel Form Layout component to split the list into two or more columns.

To create a styled list of items, perform the following steps:

1. Add the Panel List component to the JSF page.
2. Insert the desired number of child components (to display as bulleted items).
3. To style the child components, set the `ListStyle` attribute to a valid CSS 2.1 list style value, such as one of the following:

| | |
|---------------------------------------|--|
| - <code>list-style-type:disc</code> | - <code>list-style-type:decimal</code> |
| - <code>list-style-type:square</code> | - <code>list-style-type:lower-alpha</code> |
| - <code>list-style-type:circle</code> | - <code>list-style-type:upper-alpha</code> |

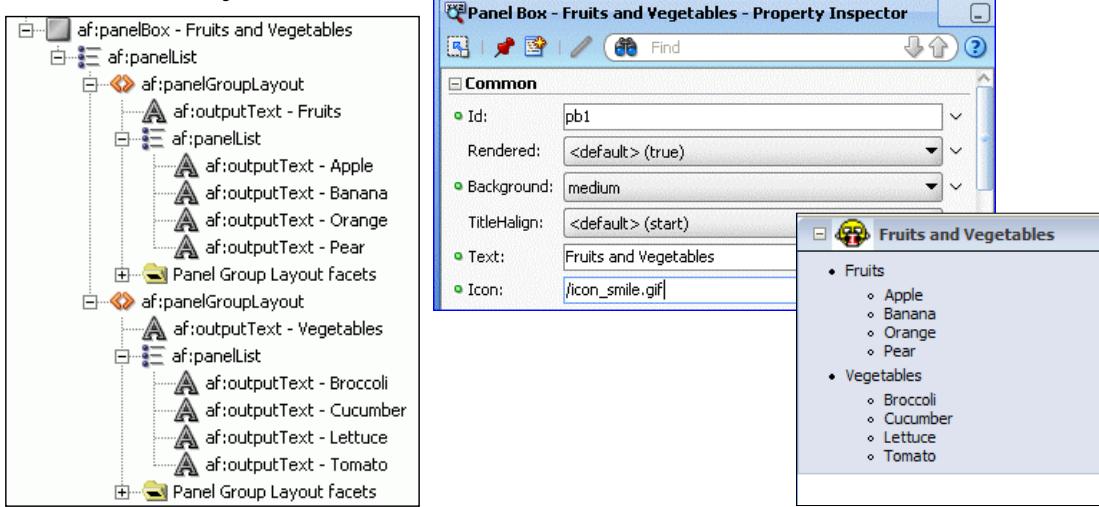
For example, entering `list-style-type:disc` into the `ListStyle` property corresponds to a disc bullet.

You can nest `af:panelList` components to create a list hierarchy that has outer items and inner items, where the inner items belonging to an outer item are indented under the outer item. You can use an `af:panelGroupLayout` component to wrap the components that make up each group of outer item and its inner items.

Displaying Items in a Content Container

Offset by Color: Panel Box

- Use `af:panelBox` to display items in a content container offset by color.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Displaying Items in a Content Container Offset by Color

You can display content in a container that can have a colored background. The `af:panelBox` component enables you to create a container that has a header, which can contain a title with an optional icon. Under the header is the box for your contents, as shown in the slide.

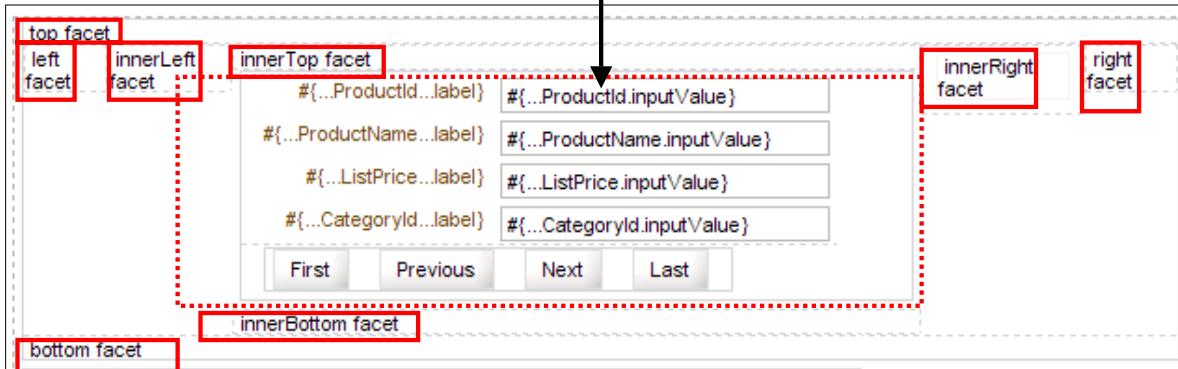
To use a colored content container for your contents, perform the following steps:

- Add the Panel Box component to the JSF page.
- Set attributes as needed:
 - `Background`: Set to light (default), medium, dark, or transparent.
 - `Text`: Set to the text string to display as the title in the header of the container.
 - `Icon`: Set to the URI of the icon image you want to display before the header text.
 - Note:** If both the `Text` and `Icon` attributes are not set, ADF Faces does not display the header portion of the content container.
 - `TitleAlign`: Set the horizontal alignment of the title to one of the following values: center, start, end, left, or right.
 - `InlineStyle`: Set the width of the container box to the exact pixel size or a percentage.
 - `ContentStyle`: Change style of contents inside the container; for example, set to `background-color:Lime` to change the color of the box to lime green.

Note: Color should never be used to convey information because of accessibility issues.

Arranging Content Around a Central Area: Panel Border Layout

Panel Border Layout: Predefined named areas around a central area where direct child content is displayed.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Arranging Items Around a Central Area

The `af:panelBorderLayout` component is a layout element for arranging components in predefined, named areas around a central area. The central area is where direct child components of `af:panelBorderLayout` render consecutively. In addition to the central area, the `af:panelBorderLayout` component supports the following named areas, each of which is defined by a facet bearing the name for the area:

- **top**: Renders children above the central area
- **bottom**: Renders children below the central area
- **left**: Renders children on the left of the central area between the `top` and `bottom` facet children
- **innerLeft**: Renders children to the right of the `left` facet
- **innerTop**: Renders children to the right of the `innerLeft` facet and below the `top`
- **right**: Renders children on the right of the central area between the `top` and `bottom` facets
- **innerRight**: Renders children to the left of the `right` and below the `top` facet
- **bottom**: Renders children below the central area
- **innerBottom**: Renders children between the `bottom` facet and the central area

Arranging Items Around a Central Area (continued)

- **right:** Renders children on the right of the center area between the top and bottom facet children. If the reading direction is left to right, right has precedence over end if both the right and end facets are used. If the reading direction is right to left, right also has precedence over start if both the right and start facets are used.
- **innerTop:** Renders children above the center area, but below the top facet children
- **innerBottom:** Renders children below the center area, but above the bottom facet children
- **innerLeft:** Similar to left, but renders between innerTop and innerBottom, and between left and the center area
- **innerRight:** Similar to right, but renders between innerTop and innerBottom, and between right and the center area
- **innerStart:** Similar to innerLeft, if the reading direction is left to right and similar to innerRight, if the reading direction is right to left
- **innerEnd:** Similar to innerRight, if the reading direction is left-to-right and similar to innerLeft, if the reading direction is right-to-left

Note: For BiDi support (right to left layout), Oracle Applications developers should not use the left and right areas of af : panelBorderLayout. Start and end should be used instead of left and right.

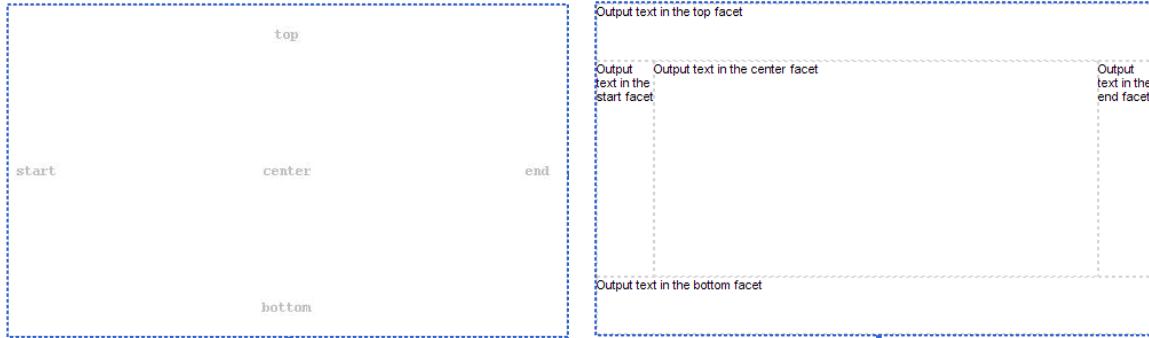
To add items in predefined areas using PanelBorderLayout, perform the following steps:

1. Add the Panel Border Layout component to the JSF page.
2. To place the content in the center area, insert the desired components inside af : panelBorderLayout.
3. The child components are displayed consecutively in the order in which you inserted the desired components. If you want some other type of layout for the child components, wrap the components inside af : panelGroupLayout.
4. To place content in one of the named areas around the center area, do one of the following:
 - If the facet is visible (for example, start or end), insert the desired components into the facet, grouping multiple components because a facet can take only one child.
 - If the facet is not visible, right-click af : panelBorderLayout, select Facets - Panel Border Layout from the context menu, and select a facet name from the list. Visible facets are indicated by a check mark in front of the facet name. Insert the desired components into the visible facet.

Arranging Content Around a Central Area: Panel Stretch Layout

The panel stretch layout component:

- Can arrange content around a central area
- Does not render anything by itself
- Has fewer facets than panel border, with no direct children



The panel stretch layout component at design time

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Arranging Content Around a Central Area: Panel Stretch Layout

As mentioned earlier, the panel stretch layout component can automatically stretch its children when it is stretched. However, it does not have any elements that render by themselves. It does have several facets to which you can add content to arrange such content around a central area. The center facet is the central area, and you can display content above, below, to the left, or to the right of it in the top, bottom, start, and end facets.

Although the panel stretch layout component is similar to the panel border layout, it differs in that:

- Fewer facets restrict you to a less flexible arrangement of content.
- You cannot add direct children. All content must be added to a facet.

Using ADF Faces Skins

- ADF Faces skins:
 - Provide a global style sheet for an application
 - Use a CSS file to set styles
 - Use a resource bundle for text
- Included skins:
 - blafplus-medium
 - blafplus-rich
 - fusion
 - fusion-projector
 - simple



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using ADF Faces Skins

You can change the appearance or look and feel of an application without having to rewrite the user interface. A skin in ADF Faces is a global style sheet that needs to be set only in one place for the entire application. Every component automatically uses the styles as described by the skin. Any changes to the skin are picked up at run time, so no change to code is needed. Skins are based on the Cascading Style Sheet specification, using CSS 3.0 syntax.

In addition to using a CSS file to determine the styles, skins also use a resource bundle to determine the text. For example, the words “Previous” and “Next” in the navigation bar of the ADF Faces table component are determined by using the skin’s resource bundle. All the included skins use the same resource bundle.

You set an element in the `trinidad-config.xml` file that determines which skin to use, and if necessary, under what conditions. In the `trinidad-config.xml` file, replace the `<skin-family>` value with the family name for the skin you want to use. For example:

```
<trinidad-config  
xmlns="http://myfaces.apache.org/trinidad/config">  
  <skin-family>fusion</skin-family>  
</trinidad-config>
```

ADF Faces Skins

simple skin

Update Supplier

| | | | |
|------------------|----------------------------|------|--------|
| * SupplierName | Stuffz | Save | Cancel |
| * SupplierStatus | ACTIVE | ^ | |
| PhoneNumber | 402.555.0158 | | |
| Email | contact@stuffz.example.com | | |

fusion skin

Update Supplier

| | | | |
|------------------|----------------------------|------|--------|
| * SupplierName | Stuffz | Save | Cancel |
| * SupplierStatus | ACTIVE | ^ | |
| PhoneNumber | 402.555.0158 | | |
| Email | contact@stuffz.example.com | | |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ADF Faces Skins

ADF Faces provides the following skins for you to use in your applications:

- **blafplus-medium**: This skin provides a modest amount of styling. This style extends the **simple** skin.
- **blafplus-rich**: This skin extends the **blafplus-medium** skin. It provides more styling than the **blafplus-medium** skin. For example, graphics in the **blafplus-rich** skin have rounded corners.
- **fusion**: This defines the default styles for ADF Faces components. This skin provides a significant amount of styling.
- **fusion-projector**: This defines the styles for an application that you want to demonstrate to an audience using a projector. This skin modifies a number of elements in the **fusion** skin so that an application renders appropriately when displayed using tabletop projectors (particularly older models of projector). This skin is useful if the audience is present at the same location as the projector. This skin may not be appropriate for an audience that views an application online through a Web conference.
- **simple**: This skin contains only minimal formatting.

You can learn more about skinning in the *Oracle Fusion Middleware 11g: Build Applications with ADF II* course.

Using Dynamic Page Layout

Dynamic page layout is made possible by using:

- Expression Language (EL)
- Partial page rendering (PPR)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Expression Language to Conditionally Display Components

- You can use EL for any of a component's attributes.
- Example of setting attributes with EL:

```
<af:selectOneChoice  
    value="#{bindings.CardTypeCode.inputValue}"  
    label="#{bindings.CardTypeCode.label}"  
    partialTriggers="PaymentType"  
    rendered="#{bindings.PaymentTypeCode.inputValue == 'CC'}">  
    <f:selectItems value="#{bindings.CardTypeCode.items}" />  
</af:selectOneChoice>
```



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Expression Language (EL)

In JSF, you use a simple Expression Language (EL) to work with the information you want to present. At run time, a generic expression evaluator returns the value of expressions, automating access to the individual objects and their properties without requiring code. Because any attribute of a component can be assigned a value by using an EL expression, it is easy to build dynamic, data-driven user interfaces.

Following are some examples of how to use EL:

- For the component's value: At run time, the value of a JSF UI component is determined by its `value` attribute. Although a component can have static text as its value, typically the `value` attribute contains an EL expression that the run-time infrastructure evaluates to determine what data to display.
- To hide a component when a set of objects you need to display is empty: Use a Boolean-valued expression such as `#{not empty ProductList.selectedProducts}` in the UI component's `rendered` attribute. If the list of selected products in the object named `ProductList` is empty, the `rendered` attribute evaluates to `false` and the component disappears from the page.

For more information about Expression Language, see:

<http://java.sun.com/j2ee/1.4/docs/tutorial/doc/JSPIntro7.html>

http://www.oracle.com/technology/sample_code/tutorials/jsp20/simpleel.html

Characteristics of Partial Page Rendering (PPR)

PPR:

- Is enabled by ADF Faces
- Enables redrawing only a portion of a page
- Requires server round-trip:
 - Rerenders only a portion of the server-side component tree
 - Downloads only the appropriate fragment of HTML
- Implements certain ADF Faces patterns
 - Single component refresh
 - Cross-component refresh
- Can be enabled declaratively or programmatically



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Partial Page Rendering

Unlike standard JSF events, ADF Faces events support AJAX-style partial postbacks to enable partial page rendering, which enables refreshing portions of page content that have minimal processing without the need to redraw the entire page. In ADF Faces components, this is implemented by a hidden `IFrame`, which is automatically added to a Web page when one of the following ADF Faces elements is used: `af:document`, `afh:body`, or `af:panelPartialRoot`.

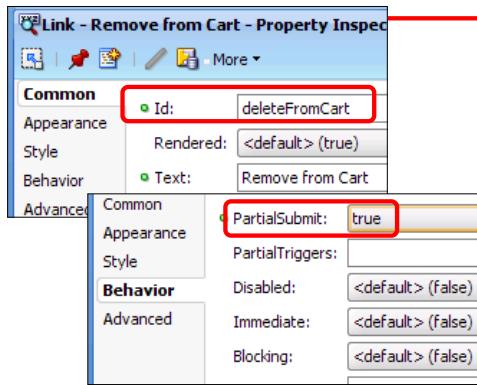
Two main AJAX patterns are implemented with PPR:

- **Single component refresh:** Implemented natively. For example, the ADF Faces table component comes with a built-in functionality that enables you to scroll through the table, sort the table by clicking a column header, mark a line or several lines for selection, and expand specific rows in the table—all through declarative property settings with no coding needed
- **Cross-component refresh:** Implemented declaratively or programmatically by the application developer by defining ADF Faces UI components to act either as a trigger for a partial update or as a partial listener to be updated

Enabling PPR Declaratively

Triggering component: **Remove from Cart**

(must have a unique ID and cause a Submit)

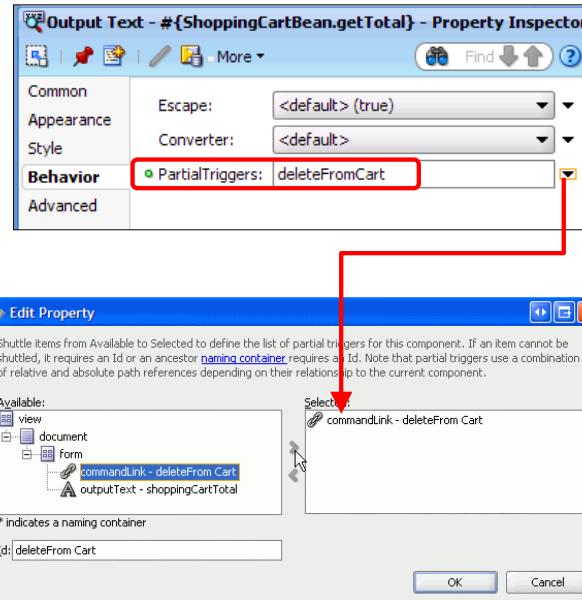


Target component:

Cart Total 500

(must specify triggering component)

Remove from Cart



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enabling PPR Declaratively

For a component to trigger another component to refresh, the trigger component must have a unique ID, which is a valid XML name, and must cause a submit when an appropriate action takes place. For a component to be refreshed as triggered by another component, it must declare which other components are the triggers.

Following are the three main component attributes used in PPR:

- **autoSubmit**: When set to `true`, and an appropriate action takes place (such as a value change), the component automatically submits the enclosing form. For PPR, you can use this in conjunction with a `listener` attribute bound to a method that performs some logic when an event that is based on the submit is launched.
- **partialSubmit**: When set to `true`, the page partially submits when the button or link is clicked. You can use this in conjunction with an `actionListener` method that performs some logic when the button or link is clicked.
- **partialTriggers**: Use this attribute to list the IDs of components whose change events are to trigger this component to be refreshed. Use a space between multiple IDs. When any of those triggering components is submitted or updated (for example, through an `autoSubmit`), this component is also updated. All rendered components support the `partialTriggers` attribute.

Enabling PPR Declaratively (continued)

To enable a component to partially refresh another component, perform the following steps:

1. On the trigger component:
 - Set the `id` attribute to a unique value.
 - If it is an input component in a form, set the `autoSubmit` attribute of the component to `true`. Otherwise, set the `partialSubmit` attribute of the component to `true`.
2. On the target component that you want to partially refresh when the trigger command component is activated, set the `partialTriggers` attribute to the ID of the trigger component. If the component refresh is to be triggered by more than one other component, list their IDs separated by spaces. It is recommended that you use the partial trigger editor for choosing the ID values. If the target component is inside a *container* component, the `partialTrigger` value might contain that *container* ID as well—which the partial trigger editor would automatically pick up.

At run time, when the triggering component fires a partial event, the output component, which is listening for partial events from the triggering component, refreshes its values using PPR.

Example:

```
<af:commandButton text="Toggle" id="commandButton1" partialSubmit="true"
actionListener="#{backing_page.commandButton1_actionListener}"/>
<af:outputText value="Hello World" binding="#{backing_page.outputText1}"
id="outputText1" partialTriggers="commandButton1"/>
```

Setting `partialSubmit` to `true` (the default value is `false`) on `af:commandButton` causes ADF Faces to perform the action through a partial page submit when the button is clicked. (Action listeners and events, as used in the example, are discussed the lesson titled “Responding to Application Events.”) The `partialTriggers` attribute on `af:outputText` tells the component to listen for any event triggered by `af:commandButton`. When an event is triggered, the `af:outputText` component automatically refreshes itself. ADF Faces does not rerender the entire page, but only those components (for example, `af:outputText`) that are listening for events fired by trigger components (for example, `af:commandButton`) are rerendered.

Note that only the following ADF Faces command components use full page rendering for their action events by default (that is, `partialSubmit` is `false` by default):

- `af:commandButton`
- `af:commandLink`
- `af:commandNavigationItem`

In contrast, the following command components always use PPR for their action events (that is, `partialSubmit` is `true` by default), unless explicitly set otherwise:

- `af:commandMenuItem`
- `af:commandToolbarButton`

Like action events, value change events in ADF Faces components can also use PPR. ADF Faces input and select components (such as `af:inputText` and `af:selectOneChoice`) automatically trigger partial page requests when their `autoSubmit` attribute is set to `true`.

Native PPR: Example

| ProductId | ProductName | ListPrice | Category |
|-----------|-----------------------------|-----------|----------|
| 38 | Beginning EJB Application | 29.99 | 8 |
| 39 | Pro EJB 3: Java Persistence | 35.99 | 8 |

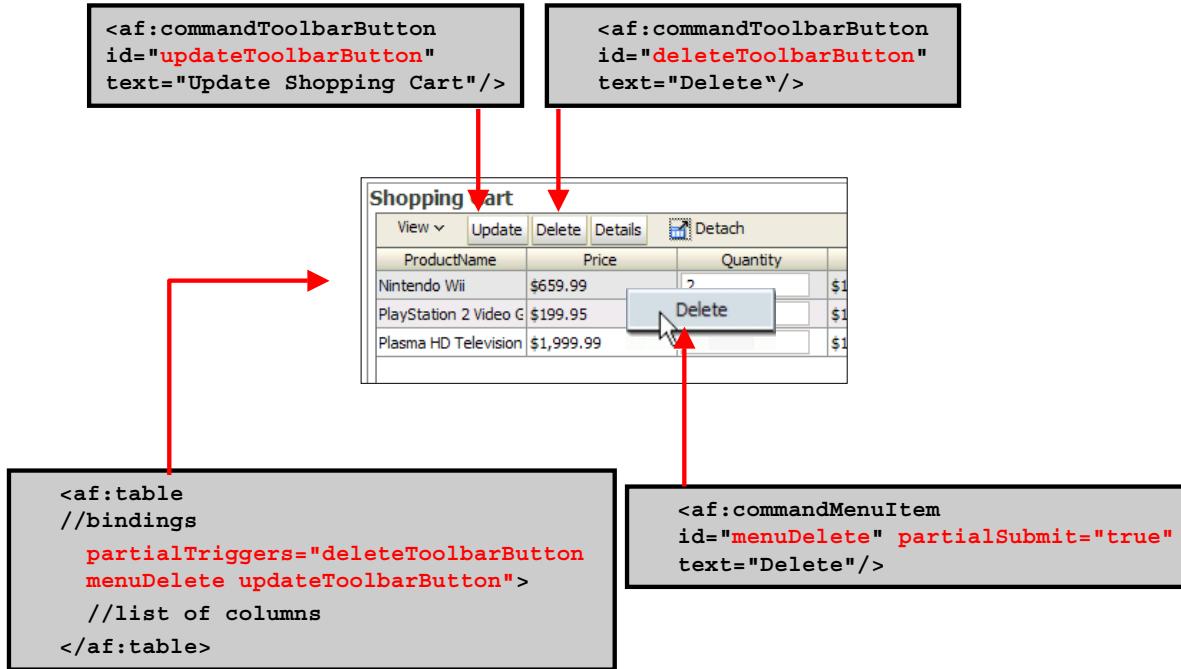
ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Native PPR: Example

As an example, the slide shows a page with two `af:showDetailItem` components in a tabbed panel near the bottom of the display, one for Product Details and one for Stock Levels. When the user selects one of the tabs, the page renders only that portion of the page, not the whole page. This behavior is built into the components; you do not need to add any code.

Declarative PPR: Example



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Declarative PPR: Example

The example in the slide is the View Cart page of the course application, which contains the following code:

```

<af:panelCollection id="tableCollection">
    <f:facet name="toolbar">
        <af:toolbar>
            <af:commandToolbarButton
                actionListener="#{FODShoppingCartBean.onDeleteRow}"
                text="Delete"
                disabled="#{!bindings.Delete.enabled}"
                id="deleteToolbarButton">
                <af:setActionListener from="#{'Modified'}"
                    to="#{requestScope.cartStatus}" />
            </af:commandToolbarButton>
            <af:commandToolbarButton id="updateToolbarButton" text="Update Shopping
Cart"/>
        </af:toolbar>
    </f:facet>

```

Declarative PPR: Example (continued)

```
<af:table value="#{bindings.ShoppingCartItem1.collectionModel}"  
    var="row" rows="#{bindings.ShoppingCartItem1.rangeSize}"  
    first="#{bindings.ShoppingCartItem1.rangeStart}"  
    emptyText="#{bindings.ShoppingCartItem1.viewable ? 'No rows yet.' :  
'Access Denied.'}"  
    fetchSize="#{bindings.ShoppingCartItem1.rangeSize}"  
    selectedRowKeys=  
        "#{bindings.ShoppingCartItem1.collectionModel.selectedRow}"  
    selectionListener=  
        "#{bindings.ShoppingCartItem1.collectionModel.makeCurrent}"  
    rowSelection="single" id="table1" inlineStyle="width:100%"  
    contextMenuItemId="tablePopup"  
    partialTriggers="deleteToolbarButton menuDelete updateToolbarButton">  
    //list of columns  
    //...  
    //...  
</af:table>  
<af:popup id="tablePopup">  
    <af:menu text="menu 1" id="tablePopupMenu">  
        <af:commandMenuItem text="Delete"  
            actionListener="#{FODShoppingCartBean.onDeleteRow}"  
            id="menuDelete" partialSubmit="true">  
            <af:setActionListener from="#{'Modified'}"  
                to="#{requestScope.cartStatus}"/>  
        </af:commandMenuItem>  
    </af:menu>  
</af:popup>  
</af:panelCollection>
```

This example illustrates setting `partialTriggers` on a table to the IDs of three components. You can see that it is not necessary to explicitly set `partialSubmit` on the toolbar buttons because it is set to `true`, by default, on this type of component. However, `partialSubmit` has been explicitly set to `true` on the command menu item, even though it was not necessary to explicitly do so because it is also set to `true` by default on this type of component.

Enabling PPR Programmatically

Why?

- Need logic to determine whether a component should refresh
- Refresh should occur on only one of several events that a triggering component may fire (for example, on row selection in table, use a selection listener)

How? Use the `addPartialTarget` method.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enabling PPR Programmatically

If you need to execute some logic to determine whether a component should be refreshed, you cannot handle the refresh using partial triggers, but need to use a different technique.

Also, if you need to refresh a component only when a certain event occurs (such as when the selection changes), and not when other events are triggered, using partial triggers may not be the most efficient method to use because when you use partial triggers, the component is refreshed for all events that are fired by those trigger components. For example, a table supports multiple events, such as sorting and selecting events. Similarly, for components that display the results of complicated queries, if you want to refresh the component only when the selection changes, a more efficient way than using partial triggers is to use a selection listener.

The `addPartialTarget()` method enables you to add a component as a partial target for an event, so that when that event is triggered, the partial target component is refreshed. With this technique, you can associate the component you want to have refreshed with the event that is to trigger the refresh.

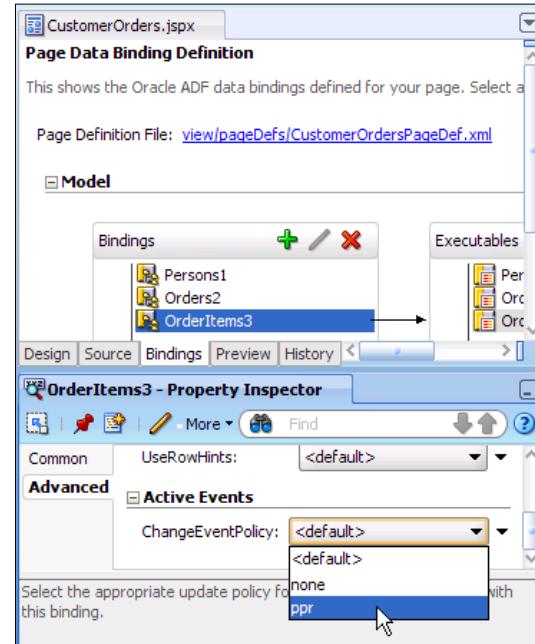
The following example, part of a managed bean, adds a UI component as a partial target :

```
public void doPartialRefresh(UIComponent uc) {  
    AdfFacesContext adfctx = AdfFacesContext.getCurrentInstance();  
    adfctx.addPartialTarget(uc);  
}
```

Enabling Automatic PPR

To enable automatic PPR:

- Select a binding in the page definition file
- Set ChangeEventPolicy to ppr



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enabling Automatic PPR

Configuring PPR manually can be tedious and error-prone, especially when back-end business logic is added to the mix. The complexity rises further when page developers use back-end components that are developed by another team because they may not be aware of how the back-end logic changes values.

To resolve this difficulty, you can enable automatic partial page refresh on the bindings of any page. This causes components whose values change as a result of back-end business logic to be automatically repainted. At run time, the binding layer is notified of value changes, and appropriate components are added to the PPR targets list.

This feature enables you to avoid much of the manual PPR configuration, so that you can focus on building the UI or business logic.

Conforming to PPR Guidelines

- Purposes of PPR:
 - Improve application performance
 - Improve the user experience
- Guidelines for using PPR:
<http://www.oracle.com/technology/tech/blaf/specs/ppr.html>
- PPR should not be used:
 - When navigating to another page
 - When response times may be long
 - When multiple sections of the page need to be redrawn
- PPR may cause accessibility issues.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Conforming to PPR Guidelines

When PPR is implemented correctly, it improves application performance as follows:

- Rendering performance is improved by generating only a subset of the page.
- Network traffic is reduced by sending only a subset of the page's contents to the browser.
- User perception of performance is improved because of not spending time looking at a blank page.

When performance improvement is not possible with PPR, it should not be implemented, thus avoiding unnecessary code bloat.

PPR should not be invoked in the following contexts:

- When navigating to another page, because some page elements, such as page titles, do not change during PPR
- When response times may be long (user is blocked during a partial page submit), such as:
 - Database queries or database maintenance operations
 - Processes that demand significant middle-tier processing
- When multiple sections of the page need to be redrawn, such as:
 - Action or choices that affect more than half the content of the page
 - Inline messaging, which features a message box at the top of the page, and may insert inline messages below multiple fields

Summary

In this lesson, you should have learned to:

- Use complex layout components
- Explain how to use ADF Faces skins
- Use dynamic page layout



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 15 Overview: Using ADF Faces Layout Components

This practice covers the following topics:

- Stretching Tables and Columns
- Adjusting JDeveloper's Default Layout Components
- Adding Layout Components to Existing Pages
- Creating New Pages with Required Layout
- Setting Conditional Display
- Implementing PPR to Coordinate Products Display with Selected Subcategory



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 15 Overview: Using ADF Faces Layout Components

In this set of practices, you use layout components and techniques to enhance the appearance of the application, and you implement partial page rendering to coordinate products with a selected subcategory.

16

Ensuring Reusability

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Explain the benefits of reusing components
- Create and use a resource catalog
- Create and use ADF libraries
- Define and use a task flow template
- Create and use a page template
- Create a declarative component and use it on a page
- Create a page fragment and use it in a bounded task flow
- Use a bounded task flow as a region



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

This lesson teaches you to create and use components that are reusable: page templates, page fragments, declarative components, and bounded task flows in regions. It also introduces the concepts of the Resource Catalog and ADF libraries to package components and make them available for reuse.

Benefits of Reusability

Designing code to be reused has the following benefits:

- Increased developer productivity
- Fewer bugs—debug once
- Consistency:
 - In functionality
 - In look and feel
- Easier maintainability
- Rapid adaptability



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Benefits of Reusability

Duplicating code or content is one of the worst practices. It can be an easy solution sometimes, but changing the application becomes a lot more difficult, which means that in the long term, you cannot adapt to new user requirements or fix problems quickly. Ideally, an application should not contain multiple versions of the same code and the Web content should not be copied and pasted.

Code reuse can increase developer productivity and application maintainability. Instead of reinventing the wheel, you should always look for well-designed frameworks and customizable components. Application-specific code can also be reused across modules and even across related projects. This latter type of reusability allows you to make rapid changes, exploit new features globally, and spend less time testing and debugging.

These things might sound like good advice for programmers, but Web developers should pay attention to them too. Oracle ADF Faces provides built-in components along with support for creating additional reusable components. Very often, however, the bits of HTML and JSP markup are copied from one Web page and pasted into other pages, which means that multiplied markup has to be modified everywhere when the Web content changes. In addition, the look of an application becomes inconsistent if some of the pages are not updated. This does not happen if the UI parts are reused across pages so that any change requires editing in exactly one place.

Designing for Reuse

- Guidelines:
 - Use naming conventions to:
 - Avoid naming conflicts
 - Facilitate identifying the component functionality
 - Standardize storage by agreeing on:
 - Type of repository needed
 - Storage and organization
 - Access methods
- Make components available for reuse by using:
 - Resource catalog
 - ADF library



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Designing for Reuse

Creating and consuming reusable components should be included in the early design and architectural phases of software projects. You and your development team should consider which components are candidates for reuse, not only in the current applications but also for future applications and for applications being developed in other departments.

When you create reusable components, you should try to create unique and relevant names for the application, project, application module, task flow, or any other relevant file or component. Do not accept the JDeveloper Wizard's default names, but try to have unique names to avoid name conflicts with other projects or components in the application. You should also consider creating standardized naming conventions so both creators and consumers of ADF Library JARs can readily identify the component functionality.

You and your team should decide on the type of repository needed to store the library JARs, where to store them, and how to access them. You should consider how to organize and group the library JARs in a structure that fits your organizational needs.

Using a Resource Catalog

A resource catalog:

- Enables application developers to find and incorporate shared resources, such as libraries containing ADF BC Model Objects, validators, message bundles, and so on, that were created by other members of the team
- Provides a federated view of the resources from one or more otherwise unrelated repositories in a unified search and browse UI
- Enables you to store libraries comprising a common repository that can be used by the developers building any client application
- Ensures that all application developers use the same business model without any discrepancies



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

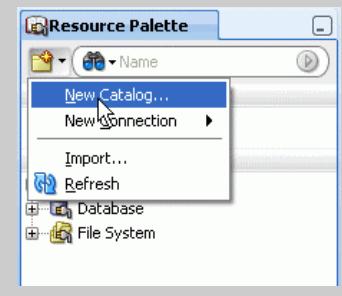
Using a Resource Catalog

The Resource Catalog is your window into the many disparate resources that your application may need to consume. It enables you to define connections to the resource providers, and to drag those resources into your application as needed. The resource catalog provides a search tool to search all the defined repositories in a single action. Connections can be created for the following types of resource repositories:

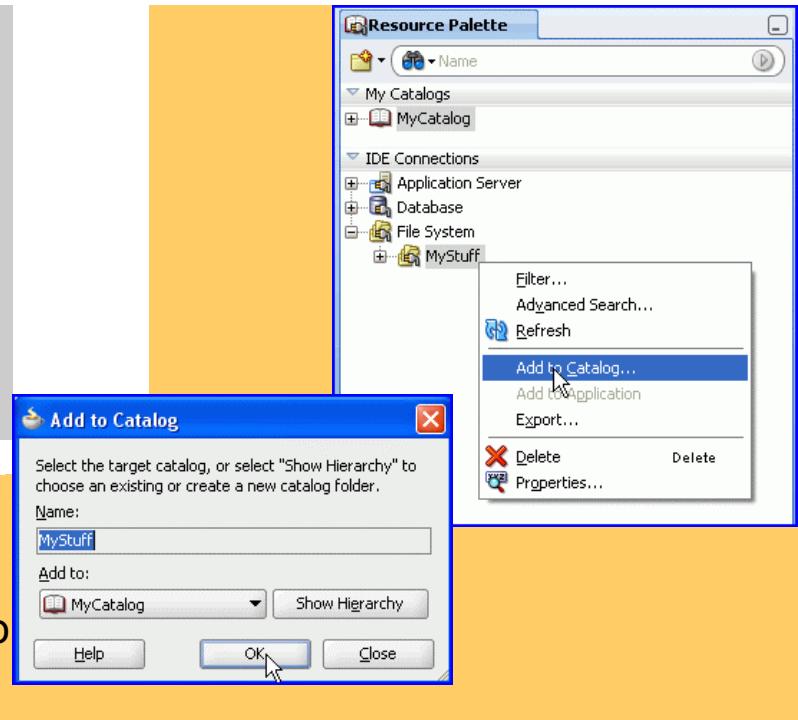
- Application server
- Database
- File system
- UDDI Registry
- URL adapter
- WebDAV
- BAM/Fabric
- BI Presentation server
- BPEL
- MDS connection
- Portlet producers
- Virtual Content Repository (VCR)
- WSDL adapter

Creating a Resource Catalog

Creating a catalog:



Adding resources to a catalog:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Resource Catalog

The Resource Palette is the mechanism for managing resource catalogs and connections. You can open the Resource Palette by selecting View > Resource Palette from the menu. From the Resource Palette, you can create new catalogs or connections or import the existing ones from an archive. You can also search for a resource.

You can create a new folder in a catalog by right-clicking the catalog and selecting New Folder. You also can apply a filter to selectively show or hide groups of resources, or you can export the catalog or connection to an archive. When you create a catalog, only you have the permission to view or edit it. You can share the catalog by granting access to other users.

To add resources to a catalog, you first create a connection to those resources in the Connections panel of the Resource Palette. Then you can right-click the resource and select “Add to Catalog.” In the “Add to Catalog” dialog box, you can accept the offered catalog by clicking OK, or click Show Hierarchy to get a list of available catalogs. You can then select one of the offered catalogs, or click New Catalog to create a new one.

Reusing Components

ADF Library enables you to reuse components:

- Package them into ADF Library JAR files.
- Add them to a resource catalog.
- Add the library to a project to use its components.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

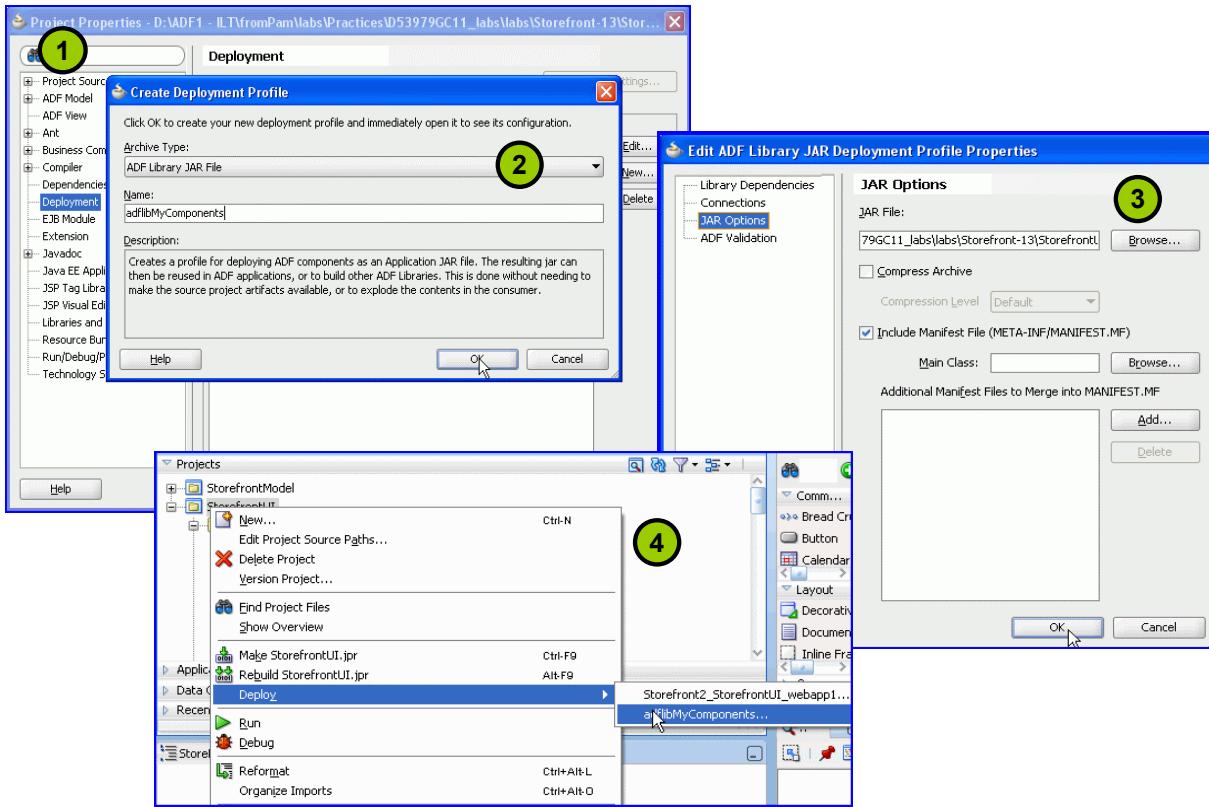
Reusing Components

In the course of application development, certain components are often used more than once. Whether the reuse happens within the same application, or across different applications, it is often advantageous to package these reusable components into a library that can be shared between different developers, across different teams, and even across departments within an organization.

ADF Library provides a convenient and practical way to create, deploy, and reuse high-level components. You should design your application with component reusability in mind. If you created components that can be reused, you can package them into JAR files and add them to a resource catalog. If you need a component, you may look into the resource catalog for that component, and then add the library into your project or application.

For example, you can create an application module for a domain and package it up to be used as the model project in several different applications. Or, if your application consumes components, you may be able to load a page template component from a repository of ADF Library JARs to create common look-and-feel pages. Then you can put your page flow together by stringing several task flow components pulled from the library.

Creating an ADF Library



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

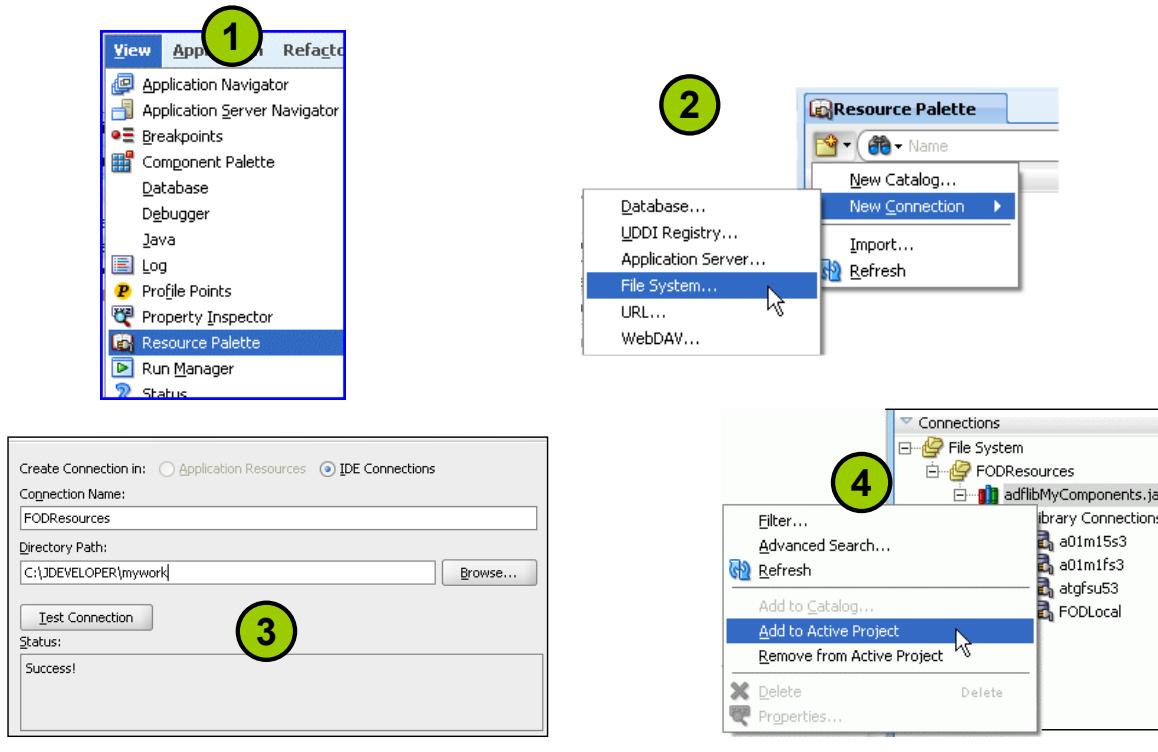
Creating an ADF Library

A project corresponds to one ADF Library JAR. If you create multiple projects and want to reuse components from each of the projects, you may need to create an ADF Library JAR for each project. In other situations, you may be able to involve multiple components under one project to create a single ADF Library JAR. For example, you may be able to create business components, application modules, task flows, and page templates all under one project and create one ADF Library JAR.

To package and deploy a project into the ADF Library JAR, perform the following steps:

1. In the left pane of the Project Properties window for the project that contains the component that you want to make reusable, select Deployment, and then click New.
2. In the Create Deployment Profile dialog box, select ADF Library JAR file from the Archive Type drop-down list and enter a name for the deployment profile. Click OK.
3. Edit the profile that you just created. In the ADF Library JAR Deployment Profile Properties dialog box, verify the default directory path or enter a new path to store your ADF Library JAR file. Click OK, and then click OK again.
4. In the Application Navigator, right-click the project and select the following from the context menu: Deploy > *deployment* > to ADF Library, where *deployment* is the name of the deployment profile.

Adding an ADF Library to a Project by Using the Resource Palette



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Adding an ADF Library to a Project by Using the Resource Palette

Using the JDeveloper Resource Palette is the easiest and most efficient way to add library resources to a project. To add the component to the project using the Resource Palette, perform the following steps:

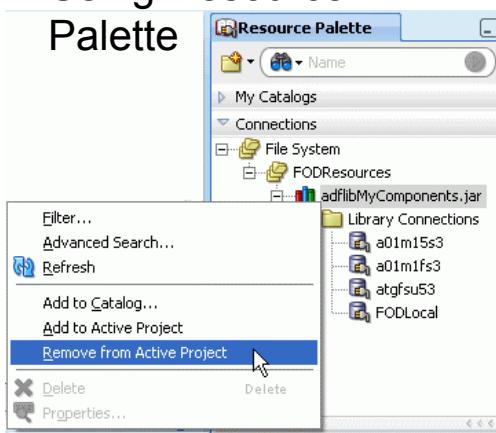
1. Select Resource Palette from the View menu.
2. In the Resource Palette window, click New Connection > File System.
3. In the Create File System Connection dialog box, enter a name for the Connection ID, and then enter the path of the JAR. Click Test Connection, and then click OK if the connection is successful.
4. The new ADF Library JAR appears under the connection name in the Resource Palette. Right-click the JAR or any component within the JAR subdirectory and select “Add to Active Project.” The JAR is added to the class path and all its supported components are added to the current project.

You cannot choose to add only one component in a multicomponent ADF Library JAR, but for application modules and data controls, you have the option to drag the application module or data control from the Resource Palette to the Data Controls Panel.

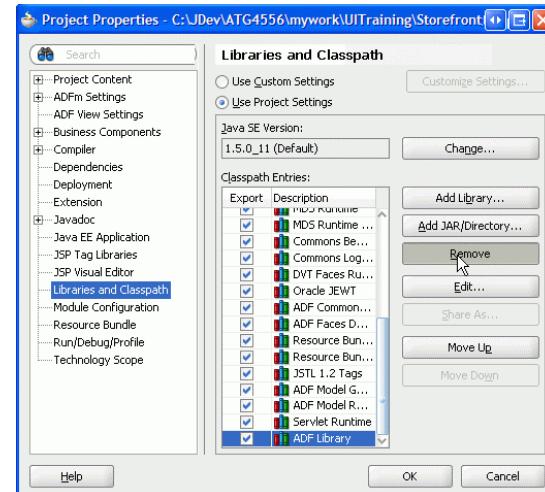
Removing an ADF Library from a Project

Remove by:

- Using Resource Palette



- Using Project Properties



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Removing an ADF Library from a Project

You can remove an ADF Library JAR only if there are no dependencies from any components to the ADF Library components. When you remove a JAR, it is no longer in the project class path and all its components are no longer available for use.

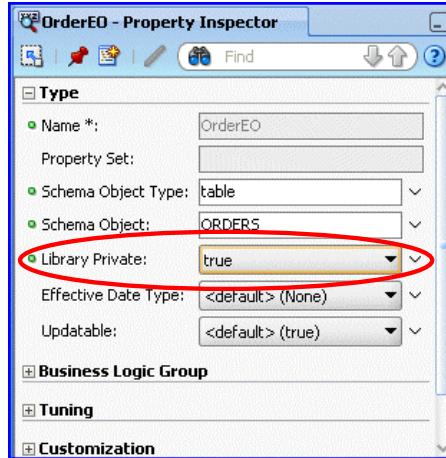
There are two ways to remove an ADF Library from a project:

- By using the Resource Palette: Right-click the JAR in the Resource Palette and select “Remove from Active Project.”
- Manually remove the JAR using the Project Properties window:
 - In the Application Navigator, double-click the project.
 - In the Project Properties dialog box, select “Libraries and Classpath” in the left pane.
 - In the “Libraries and Classpath” list, select ADF Library. Click Edit.
 - In the Edit Library Definition window, select the ADF Library JAR that you want to remove under the Class Path node. Click Remove.
 - Click OK to accept the deletion, and click OK again to exit the Project Properties window.

After you have deleted all ADF Library JAR files from the project, an ADF Library placeholder icon may still be present in the Project Properties Libraries window, as shown in the slide. You do not need to remove this icon.

Restricting BC Visibility in Libraries

Set Library Private property for a business component to `true` if you do not want consumers to see the object in the library JAR.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Restricting BC Visibility in Libraries

When deploying a library for reuse, there may be some business components that are not intended for reuse, but must be included in the library because other components depend upon them. For example:

- The library developer wants consuming projects to access components in the library only through services exposed on the application module. All components in the library are private and may not be reused.
- The library developer wants consuming projects to access an entity object only by using or extending view objects in the library, and the entity object contains attributes that should not be exposed in any view object. In this case, the entity object is private and the view objects are public.

You can mark an ADF BC object as public (the default) or private. Setting Library Private to `true` keeps the object from appearing to the consumer who is browsing the library JAR.

To mark a BC object as private, perform the following steps:

1. Open the object in the editor.
2. In the Property Inspector, click the Type tab and set Library Private to `true`.

When an object is marked as private, it is not visible in the Resource Catalog or in the Application Navigator of the consuming project.

Types of Reusable Components

ADF supports reusing the following types of components:

- Data controls
- Application modules
- Business components
- Task flows
- Task flow templates
- Page templates
- Declarative components



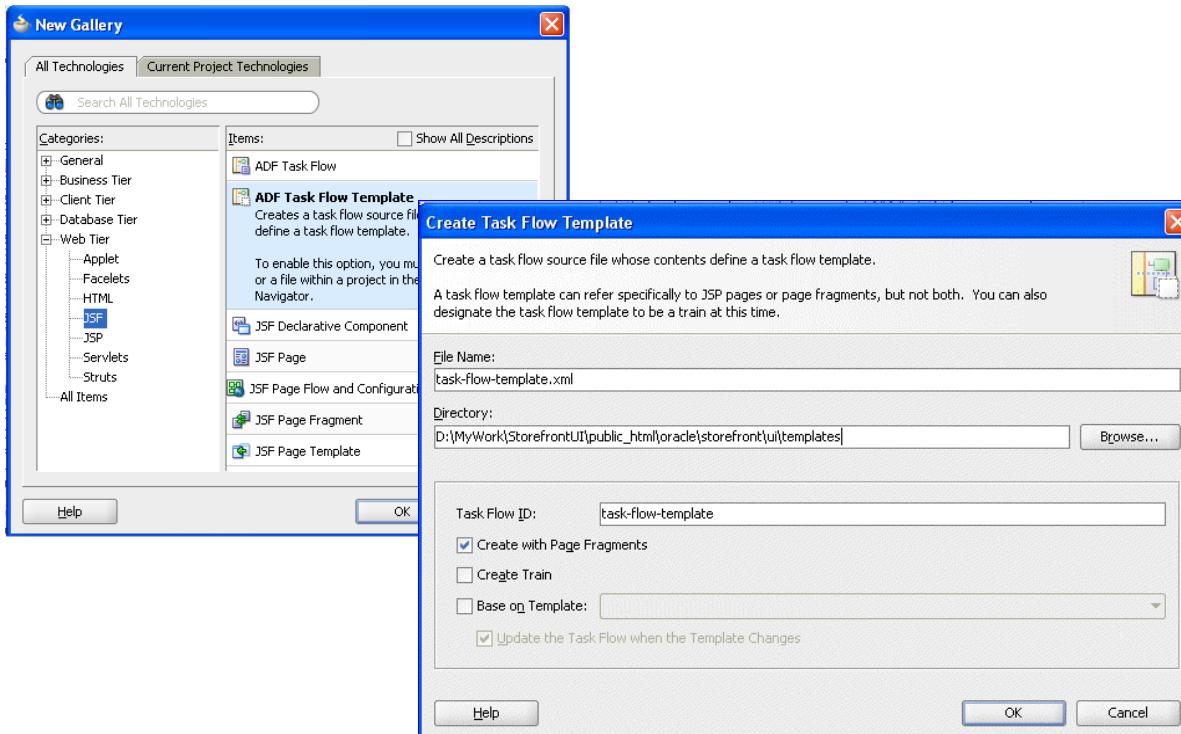
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Types of Reusable Components

The following reusable components are supported by ADF:

- **Data Control:** Any data control can be packaged into an ADF Library JAR.
- **Application Module:** When you package an application module data control, you also package the business components associated with that application module.
- **Business Components:** Business components are the entity objects, view objects, and associations used in the business layer; you can package them by themselves or together with an application module.
- **Task Flows and Task Flow templates:** Task flows can be packaged into an ADF Library JAR for reuse. ADF bounded task flows built using pages can be dropped onto pages, creating a link to call the bounded task flow. If an ADF task flow template was created in the same project as the task flow, the ADF task flow template is included in the ADF Library JAR and is reusable.
- **Page Templates:** You can package a page template and its artifacts into an ADF Library JAR. If the template uses image files and they are included in a directory within your project, these files are also available for the template during reuse.
- **Declarative Components:** You can create declarative components and package them for reuse. The tag libraries associated with the component are included and loaded into the consuming project.

Using Task Flow Templates



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Task Flow Templates

Task flow templates provide the ability to capture common task flows and behavior for reuse across many different bounded task flows. Two reuse mechanisms are provided: by copy and by reference. Exception handling is an area where developers can make use of task flow templates.

Task flow templates define reusable skeletons for task flow definitions to extend and, therefore, cannot be run on their own. To run a task flow template within JDeveloper, you must create a task flow definition that extends it and run that task flow instead.

You create a task flow template from the New Gallery by selecting Web Tier > JSF > ADF Task Flow Template.

Characteristics of Page Templates

Page templates:

- Are reusable
- Enable consistent look across pages or page fragments
- Are built from standard ADF components
- Cannot be nested
- Use partial page refresh when navigating between pages that use the same template
- Use three types of files:
 - Page-specific definition file (.jspx)
 - Definition file for all templates: pagetemplate-metadata.xml
 - Application or library file (.cpx)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of Page Templates

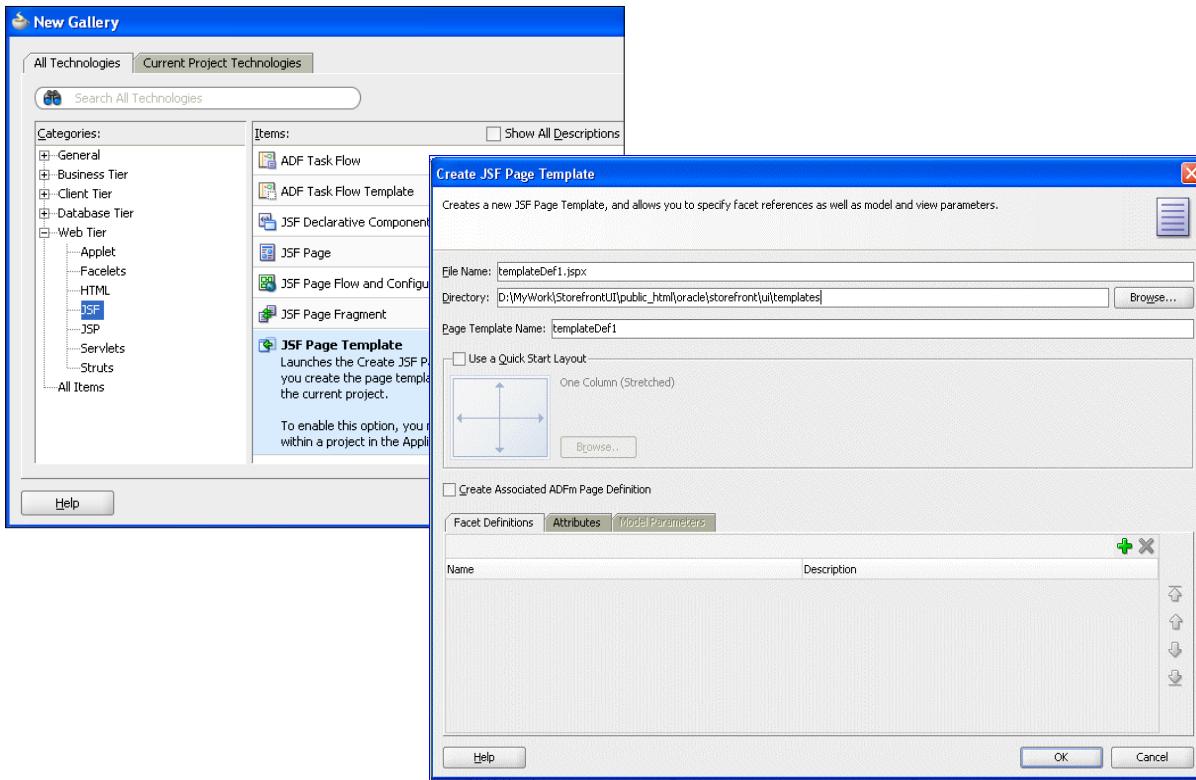
A page template provides a reusable page definition that enforces consistent branding and page layout across the application. It enables developers to change the branding and page layout for all the pages or page fragments that use the page template by changing only the page template definition. Templates can contain attributes and ADF bindings.

Page templates use standard ADF components; therefore, all skins are automatically applied. They cannot be nested, or based on another page template. Pages that use the same page template take advantage of partial page refresh—just the content that is specific to a page changes without the need to refresh the template.

There are two sets of page template definition files:

- For each page template there is:
 - A JSP XML file (.jspx), which defines the rendered UI in terms of other JSF components, facet references, and EL expressions resolved against the page template's UI attributes and model
 - An optional model-specific file—for example, an ADF page definition (.pdm) file
- Across all page templates there is:
 - A pagetemplate-metadata.xml file, which contains the paths of all the page template definitions in the project, located in the META-INF directory of the project
 - An application or library .cpx file, which contains any application and library-wide model information

Creating a Page Template



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

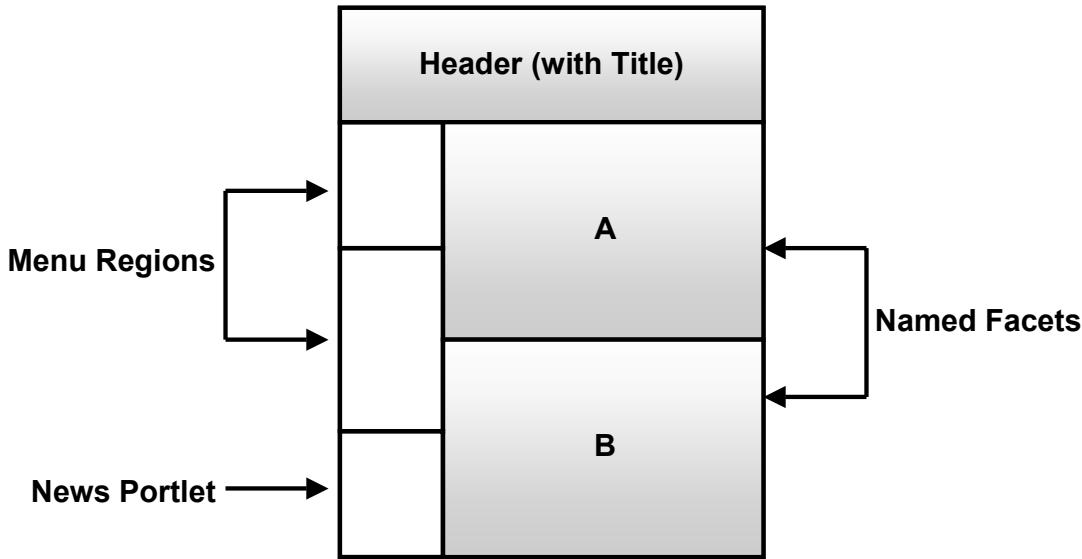
Creating a Page Template

You can create a JSF page template from the New Gallery by selecting Web Tier > JSF > JSF Page Template. You give it a file name, and optionally a different page template name. You can optionally select to create an associated page definition file as well.

The Create JSF Page Template dialog box also enables you to create:

- A quick start layout for the template
- Facet definitions: Placeholders for content to be provided by page developers
- Attributes: Attributes that can be referenced from the template; abstraction between data provided on the page and that used by the page
- Model parameters: Parameters in the page definition file that are passed differently from attributes

Creating a Page Template



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Page Template (continued)

After you close the Create JSF Page Template dialog box, the page template opens in the visual editor. You can add components by dragging them from the Component Palette, and you use the Property Inspector to set properties on components to affect appearance and behavior. You can bind components to data. You can also add regions or portlets.

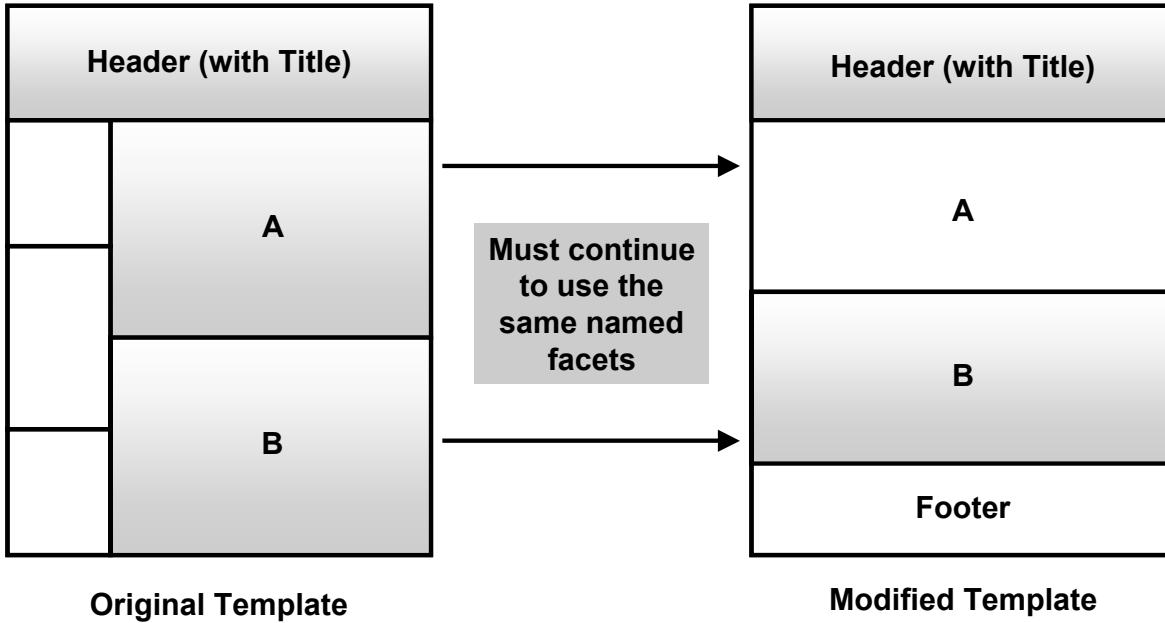
In addition, you can add attributes to the page definition that are accessible via Expression Language (EL). For example, the page definition may define a title attribute and place the title in a specific place on the page. The user of this template would only need to define the text for the title attribute, which would then appear in the correct place on the page.

Page templates typically have static areas that cannot be changed and dynamic areas where developers can place content specific to the page they are building. The example in the slide shows a header, menu regions, and a news portlet that appear on all pages that use this template.

The template developer can also place on the page a `FacetRef`, which is a placeholder for content on the page where this template is used. The sections labeled A and B in the example are facets, or placeholders, where the page developer who uses this template can drop content. This content could be any JSF component, region, or portlet.

By using a template to create a page, you save time and duplicated effort because you do not have to rebuild areas that are common across multiple pages, and you also ensure consistency across the application.

Editing a Page Template



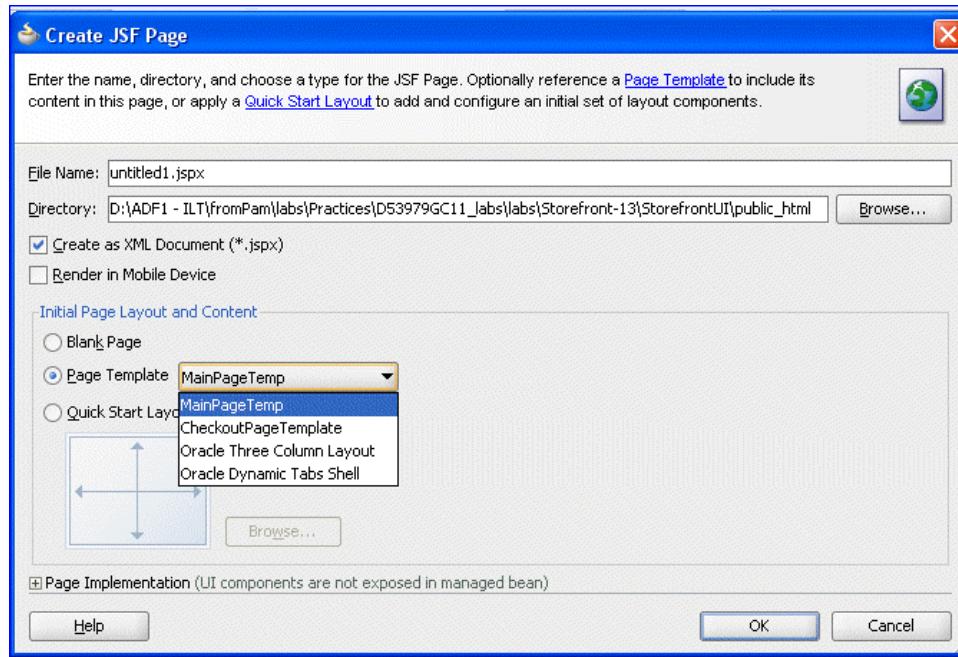
ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Editing a Page Template

A big advantage of page templates is that when you edit the template, any page that uses the template picks up the changes automatically. However, it is required that you maintain the named `FacetRefs` in the template to avoid breaking pages that use the template. The modified template shown in the slide continues to use facets A and B after modification, so pages built using that template would still work.

Applying a Page Template to a Page



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Applying a Page Template to a Page

When you create a JSF JSP in JDeveloper, the wizard gives you the choice to select an available page template from the drop-down list on which to base the creation of the skeleton JSF page. The template defines the entire page layout as defined in the template definition file.

Pages that are created by using page templates are accessed, packaged, versioned, bookmarked, and consumed as is any other stand-alone page.

You can also apply a page template to a page fragment. Page fragments are discussed later in this lesson.

Note: A page template does not necessarily need to be used to construct a single page or page fragment. It can also be used as reusable sections of the page. However, there is no design-time support for this, and doing so is outside the scope of this course.

Characteristics of Declarative Components

Declarative components:

- Are component definitions that are reusable across applications
- When changed, affect all pages that use them
- Are made up of any number of other components
- Do not have data binding in their definition
- Can be made available for reuse in the Component Palette
- Are defined only at design time in:
 - One `.jspx` file for each component
 - One `declarativecomp-metadata.xml` file for all declarative components in a project



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of Declarative Components

Declarative components are reusable, composite UI components that are made up of other existing ADF Faces components. Suppose you are reusing the same components consistently in multiple circumstances. Instead of copying and pasting the commonly used UI elements repeatedly, consider defining a declarative component that contains those components, and then reusing that composite declarative component in multiple places or pages.

Any changes to the declarative component affect all pages that use the declarative component. Declarative components do not support ADF Model parameters. They are not databound and contain no business logic. This means that you cannot drag an item from the Data Controls panel when defining the component layout section of a declarative component.

However, when consuming a declarative component on a page, you can drop databound content into the facets defined on the declarative component, and the declarative component's attributes can have data binding. For example, if you want to reuse a databound address object that comprises four input fields on a JSF page, you could create a declarative component with four input components and four parameters, and assign the parameters to the `value` attributes of the input components. At design time, the page author can bind the input component values (via the declarative component's attributes in the Property Inspector) to different data sources, including ADF Model data binding.

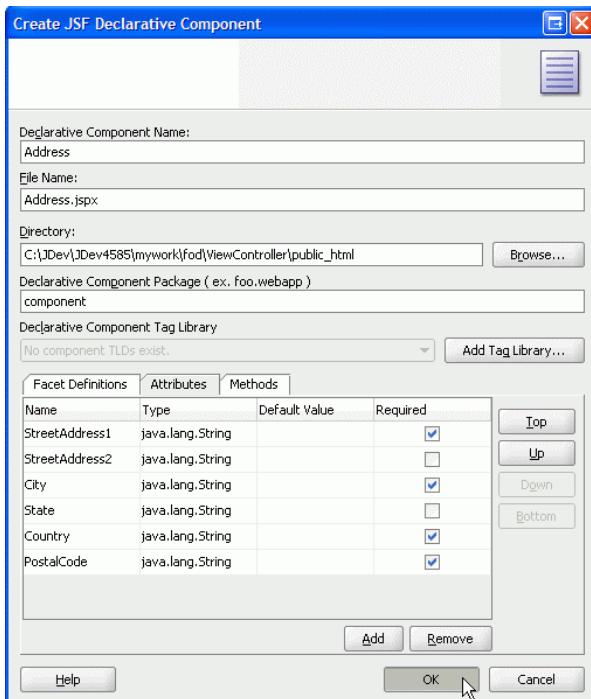
Characteristics of Declarative Components (continued)

Declarative components are composed of other components. For example, you could design a declarative component to include a facet (placeholder) for the product image component and facets for two text components to display a product description and product cost. You can also hide attributes, such as labels, of a declarative component, so that they cannot be changed by other developers who use the component.

A declarative component cannot be changed or customized at run time. It must be defined at design time, and uses the following definition files:

- A JSP xml file (.jspx) that defines the rendered UI in terms of other JSF components, Facet references, and EL expressions resolved against the declarative component's UI attributes. There is one such file for each declarative component
- A `declarativecomp-metadata.xml` file that contains the declarative components' file paths in the project. This file, located in the `META-INF` directory of the project, is shared across all declarative component definitions in an application or library

Creating a Declarative Component



The JSF Declarative Component dialog box:

- Is invoked with New > Web Tier > JSF > JSF Declarative Component
- Creates the `<aaf:componentDef>` tag in .jspx file
- Creates metadata

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Declarative Component

When you invoke the Create JSF Declarative Component dialog box, you can specify the name of the component, its file name and path, and the package in which to place it. You must add a tag library, and you can define facets, attributes, and methods for the component. When you click OK, the .jspx file opens in the editor. You can drag components to the editor to create the component.

JDeveloper creates an `<aaf:componentDef>` tag in the declarative component definition JSPX fragment. This tag acts as the root of the tag hierarchy. This tag describes the entire declarative component in two sections:

- `aaf:xmlContent` section: Defines the declarative component metadata in `facet` and `attribute` elements
- Component layout section (anything outside of `aaf:xmlContent`):
 - Defines the actual UI components that make up the composite declarative component
 - Requires that only one child is defined for the layout, so you can use one of the panel layouts to contain multiple elements
 - Uses `aaf:facetRef` tags to reference named facets, and EL expressions containing named attributes to enable page authors to set their own property values for the declarative component

Using a Declarative Component on a Page

To use a declarative component on a page:

1. Deploy the component's project as an ADF library JAR file.
2. Add the ADF library to your current project.
3. Select the ADF library from the Component Palette drop-down list.
4. Select the declarative component and drag it to your page, providing values for any attributes.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using a Declarative Component on a Page

To use a declarative component on a page, you first have to ensure that the component appears in the Component Palette. You then can drag the component to a page just as you would any other component.

To use a declarative component in another project, you first must deploy the components' project as an ADF library JAR file, and then add it to your current project as described previously—but add it as a JSP tag library. After it is added to your project, the library appears in the drop-down list in the Component Palette, and you can use any declarative components defined in that library by dragging them to a JSF JSP.

You can use the Property Palette to provide values for the attributes that you defined. If you marked any of the attributes as required, when you drag the component to the page, a dialog box appears to enable you to enter attribute values.

Characteristics of Page Fragments

Page fragments:

- Are built like regular pages
- Have page definition files like regular pages do
- Are not defined as a full Web page
- Can be displayed within a page that contains other content
- Cannot contain `af:document` or `f:view` tags
- Cannot be run on their own



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of Page Fragments

As you build Web pages for an application, some pages may quickly become large and unmanageable. One possible way to simplify the process of building and maintaining complex pages is to use page fragments.

Large, complex pages broken down into several smaller page fragments are easier to maintain. Depending on how you design a page, the page fragments created for an entire page may also be reused on other pages. For example, suppose different parts of several pages use the same components, then you might find it beneficial to create page fragments containing those components, and reuse those page fragments on several pages or in several places on the same page. Deciding on how many page fragments to create for one or more complex pages depends on your application, the degree to which you want to reuse portions of a page between multiple pages, and the need to simplify complex pages.

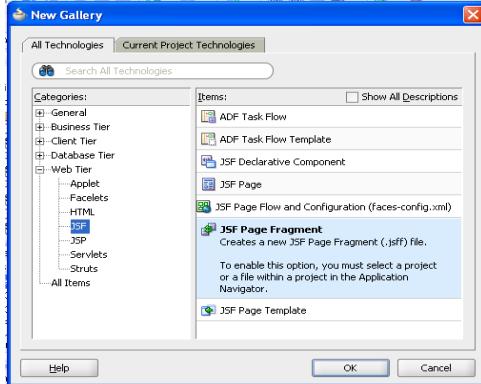
Page fragments are created and used much like pages, except that page fragments are not defined as a full Web page. This enables page fragments to be used as a smaller part of a larger page. To utilize the capability of page fragments to be used as part of a page, it is likely that most pages for Fusion applications will be built as page fragments.

A page fragment does not contain `f:view` or `af:document`; its contents are simply enclosed within `jsp:root`. You cannot run page fragments like you can run complete JSF pages.

Creating a Page Fragment

You can create a page fragment by:

Using the New Gallery OR double-clicking a view in a bounded task flow with page fragments



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Page Fragment

To create a page fragment, perform the following steps:

1. In the Application Navigator, select the project where you want to create and store page fragments. Then use the JSF Page Fragment item in New Gallery to open the Create JSF Page Fragment Wizard.
2. Enter a name for the page fragment file. By default, JDeveloper uses `.jsff` for the source file extension. You cannot overwrite the file extension in the wizard.
3. Accept the default directory for the page fragment or choose a new location. By default, JDeveloper saves page fragments in the project's `/public_html` directory in the file system. For example, you could change the default directory to `/public_html/fragments`.
4. If you want to create a page fragment based on a page template, select a template name from the drop-down list.

Another way to create a page fragment is to drag a view to a bounded task flow that uses page fragments, and then double-click that view to create the page fragment.

When finished, JDeveloper displays the page fragment file in the visual editor. You can drag components from the Component Palette onto the page fragment. You can use any ADF Faces or standard JSF component, but must not enclose the contents in `f:view` or `af:document`.

Using a Page Fragment on a Page

- You can use a page fragment on a page by:
 - Inserting the `jsp:include` tag:
 - The included page fragment uses the binding context of the consuming page.
 - Page definition file for page fragment is not loaded (does not use bindings).
 - Inserting a bounded task flow with page fragments as a region on your page
 - The page fragment can have its own binding context.
- Modifying the page fragment affects all pages that use it (but check the overall layout of consuming pages).



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using a Page Fragment on a Page

When you build a JSF page using page fragments, the page can use one or more page fragments that define different portions of the page. The same page fragment can be used more than once on a page, and on multiple pages.

To consume a page fragment on a JSF page, at the part of the page that will use the page fragment contents, insert the `jsp:include` tag to include the desired page fragment file, as in the following example:

```
<jsp:include page="/fragment_page.jsff"/>
```

Another way to use a page fragment is as part of a bounded task flow that you add to a page as a region; this is described shortly.

When you modify a page fragment, the pages that consume the page fragment automatically display the modifications. With pages built from page fragments, when you make layout changes, you should check that the overall layout of pages that consume the page fragments still meet requirements.

If the consuming page uses Oracle ADF Model data binding, a page fragment used with `jsp:include` uses the binding container of the consuming page, so its page definition file is not loaded. Only page fragments created as part of ADF bounded task flows can have their own binding context.

Characteristics of Regions

An ADF region:

- Represents a task flow as part of a page
- Is similar to a portlet, but for local functionality
- Can share information and transaction boundaries with other page content



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of Regions

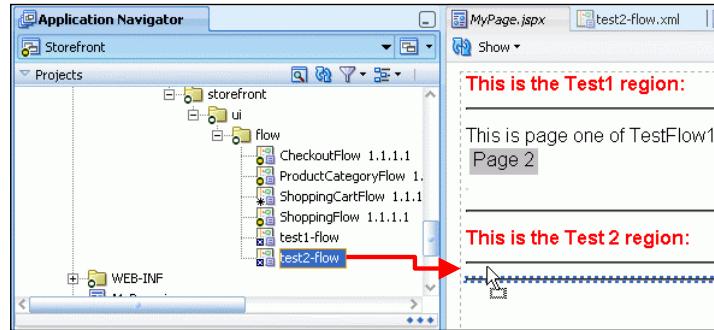
A region is a powerful new feature in Fusion that can be used within a page or page fragment. A region is a JSF component that represents a task flow as a smaller part of a larger page. You can use a region to wrap a task flow for display on a page that has other content.

A region is similar to a portlet in functionality and user behavior, but regions are used when the functionality is in a local context, whereas portlets are used when exposing the functionality for external usage. Because a region runs locally in the local context, it can share information and transaction boundaries with the rest of the items on the page, and it does not have the overhead that a remote portlet does. Any task flow can also be wrapped as a portlet for external usage, but should be used as a region for internal use cases.

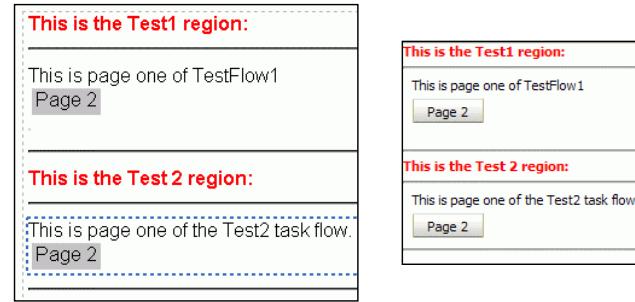
Building well-defined, parameterized task flows that can be used as regions on pages is a good way to provide reusable and highly maintainable segments of functionality across your application. You learn about passing parameter between regions in the lesson titled “Passing Values Between UI Elements.”

Wrapping a Task Flow as a Region

Drag a bounded task flow to the page as a region:



The default activity is displayed at design time and is the initial activity that is displayed at run time:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

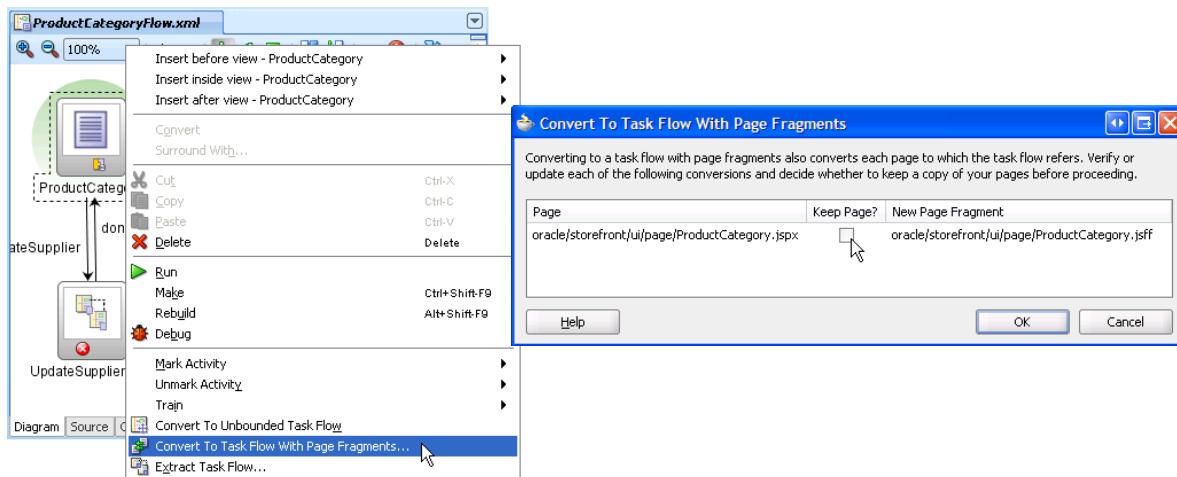
Wrapping a Task Flow as a Region

You can drag to your page a bounded task flow that includes page fragments and create it as a region on the page. The default activity of the task flow appears in the editor and on the page when it is first run.

At run time, you can use page navigation (described in the lesson titled “Implementing Navigation on Pages”) to navigate through the task flow. As you do this, the region refreshes without redrawing the entire page (partial page refresh).

Converting a Bounded Task Flow to Use Page Fragments

To use a bounded task flow containing pages as a region, convert it to a task flow with page fragments.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Converting a Bounded Task Flow to Use Page Fragments

When you are initially developing a bounded task flow, you may use view activities that correspond to pages (.jspx). However, to use a task flow as a region, the view activities cannot be full pages, so you can convert the task flow to one that uses page fragments. You have to only right-click a blank space on the task flow diagram and select Convert To Task Flow With Page Fragments.

This converts all pages associated with view activities in the bounded task flow to page fragments. You can optionally save old pages by selecting the Keep Page? check box. New page fragment names default to the names of the old pages. For example, MyPage.jsp becomes MyPage.jsff.

Note: It is best practice to use page fragments on other pages only through bounded task flows, even if you have a page flow with only that page fragment.

Deciding Which to Use

- You can use a page template when you want:
 - Similar page layout to be used on multiple pages
 - Similar look on multiple pages
- You can use a declarative component when you want to:
 - Use a similar grouping of components on multiple pages or applications
 - Select the component from the Component Palette
- You can use a bounded task flow when you want to:
 - Reuse a set of activities and control flows
 - Use multiple page fragments in a region on a page
 - Create a portlet from a set of activities and control flows



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Deciding Which to Use

At first you may be confused about the differences between page templates, declarative components, and bounded task flows and when you would want to use each.

You could use a page template when multiple pages have similar layout. For example, you may want to display several data elements with a master at top and detail at the bottom in a panel splitter. You may also want several pages to have a similar look and feel, such as branding, header, footer, or toolbars.

A declarative component is a composite of existing components that you define and reuse from the Component Palette. If you often find yourself constructing groupings of components that are the same, you can define a declarative component, so that all you need to do is drag it from the Component Palette to reuse it in multiple pages or applications. For example, you could define an address component that could be used whenever you display an address of any type in any application. That declarative component can be bound to different data, just like any component.

Bounded task flows are useful any time you have a set of activities and control flows that needs to be reused, such as a shopping cart or checkout in a shopping application, or creating an account in a banking application. You can call the bounded task flow. When constructed with page fragments, a bounded task flow can be used in a region on a page, so that the activities are displayed in the region without refreshing the entire page. You can create a portlet from a task flow as well.

Summary

In this lesson, you should have learned how to:

- Explain the benefits of reusing components
- Create and use a resource catalog
- Create and use ADF libraries
- Define and use a task flow template
- Create and use a page template
- Create a declarative component and use it on a page
- Create a page fragment and use it in a bounded task flow
- Use a bounded task flow as a region



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 16 Overview: Implementing Reusability

This practice covers the following topics:

- Converting Bounded Task Flows to Use Page Fragments
- Creating a Page Template and Applying It to Existing Pages
- Creating a Page Template and Applying It to a New Page
- Using Bounded Task Flows as a Region on a Page



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 16 Overview: Implementing Reusability

In this set of practices, you first convert all of the bounded task flows to use page fragments. You create page templates and apply them to the existing checkout pages and to a new main page. You then use the bounded task flows as regions on the main page.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Osi S.R.L. use only

Passing Values Between UI Elements

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Objectives

After completing this lesson, you should be able to do the following:

- Define the data model to reduce the need to pass values
- Use a managed bean to hold values
- Store values in memory-scoped attributes
- Use parameters to pass values



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

This lesson explains how to pass parameters and values between UI elements such as pages, regions, and bounded task flows.

Holding Values in the Data Model (Business Components)

- Define transient attributes to hold values.
- Row concurrency can coordinate multiple pages without passing a parameter.
- Defining view links in the data model:
 - Coordinates master and detail pages
 - Reexecutes queries as needed
 - Reduces the need to pass parameters between UI pages and regions

The red bar spans most of the width of the slide, centered horizontally.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Holding Values in the Data Model

Properly constructing a data model in ADF Business Components eliminates some of the need to pass parameters between pages and regions. For example:

- You can define transient attributes in business components to retain values.
- ADF BC maintains the state of the currently selected row, so a query form that displays results on a different page does not have to pass a parameter to that page, but relies on row concurrency.
- When you define view links between related view objects, you can rely on row concurrency to coordinate pages that contain master and detail results, without explicitly coding parameter passing or reexecuting of queries.

Development of the user interface is, therefore, made much easier if you take time to define the data model to consider related data that may need to be displayed.

Holding Values in Managed Beans

- Managed beans are optional and can be used to:
 - Hold values
 - Store state
- How?
 - Managed properties
 - Memory-scoped attributes



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Holding Values in Managed Beans

Managed beans enable you to hold values, either in the properties of the managed bean or in the scoped managed bean attributes. You learn more about configuring and coding managed beans in the lesson titled “Responding to Application Events.”

You can pass values between pages by using managed beans to hold the values. The JSF run time manages instantiating these beans on demand when any EL expression references them for the first time. When displaying a value, the JSF run time evaluates the EL expression and pulls the value from the managed bean to populate the component with data when the page is displayed. If the user updates data in the UI component, the JSF run time pushes the value back into the corresponding managed bean based on the same EL expression.

Using Managed Properties

- Managed bean attributes that are exposed through the getter and setter methods
- Configured in the .xml file of the task flow
- Possible values:
 - null
 - Literal string
 - Lists and Maps
 - Value binding expression (EL)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Managed Properties

Managed properties are bean attributes that are exposed through the setter and getter methods for write and read access. All attributes that are not explicitly set as managed properties retain the default values upon bean instantiation.

The <managed-property> element represents an individual property of a managed bean. A managed property calls the equivalent setter method of the bean on bean initialization.

The <managed-property> element is nested within the <managed-bean> element.

Managed properties must have a name and a value. Optionally, they can have a property class, a description, a display name, and an icon. The last three elements are for tools to display the property and have no meaning for the JSF application at run time.

The property class does not need to be mentioned for simple types because JSF will figure this out.

Although null is among the possible values for a managed property, it is allowed only if the property is an object. This means that null cannot be used for types of int, boolean, and so on.

Managed Properties: Examples

Literal string:

```
<managed-bean>
<managed-property>
  <property-name>label</property-name>
  <value>Hello World</value>
</managed-property>
```

EL (accessing a managed bean):

```
<managed-bean>
<managed-property>
  <property-name>label</property-name>
  <value>#{Userbean['firstname']}</value>
</managed-property>
```



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Managed Properties: Examples

Expression Language can be used within `adf-config.xml` to initiate the value of a managed property. The EL can reference any object that is in the scope, including previously defined managed beans.

The first example in the slide shows using a literal string for a `label` managed bean property. This example illustrates how to use a literal string, but in practice, labels would come from a resource bundle.

One reason to use a managed property in the `adf-config.xml` file is when you want to have an expression always evaluated and ready for use. For example, you could use a managed property to evaluate `#{bindings}`, giving you handy access to `DCBindingContainer`.

Using Memory-Scoped Attributes

- JSF scopes:
 - application
 - session
 - request
 - none
- Why use scopes?
 - Save state of model
 - Enable passing values to other pages or phases in the same scope
- pageFlow scope:
 - Retains values for current process (page flow) only
 - Is independent of other processes, unless using LaunchEvent
 - Can be accessed via EL or Java
 - Can be explicitly cleared



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Memory-Scoped Attributes

Scopes in JSF enable you to save the state of the model for holding and passing information to another page or to another life-cycle phase. You can set a scope on managed beans, and you also can store values in memory-scoped attributes without explicitly defining a managed bean.

JSF memory scopes include:

- application: The object is available for the duration of the Web application. This is helpful for global beans such as Lightweight Directory Access Protocol (LDAP) directories.
- session: The object is available to the client throughout the client's session.
- request: The object is available from the time it is instantiated until a response is sent back to the client. This is usually the life of the current page.
- none: The object is instantiated each time it is referenced. The object is usable only within EL references in the task flow's .xml file itself.

ADF Faces provides additional scopes:

- view: Stores objects used by a single page and retains the objects as long as the user continues to interact with the page, and then automatically releases them when the user leaves the page
- pageFlow: Makes it easier to pass values from one page to another within a task flow. This scope means that the bean is available for a set of pages in the current task flow.

Using Memory-Scope Attributes (continued)

- **backingBean:** Is used for managed beans for page fragments and declarative components only. The object is available for the duration between an HTTP request until a response is sent back to the client. This is needed because there may be more than one page fragment or declarative component on a page and to avoid collisions, any values must be kept in separate scope instances. Therefore, any managed bean for a page fragment or declarative component must use the `backingBean` scope.

Using Page Flow Scope

For efficient memory usage, you should store values in the scope with the shortest duration that fits your needs. For example, to make a value accessible anywhere within a task flow, you would use page flow scope. Each ADF task flow can specify a page flow scope that is independent of the memory scope for all other task flows.

When one task flow calls another, the calling task flow cannot access the called task flow's page flow scope, so you can use the page flow scope to pass data values only between activities within a task flow. Application and session scopes are also allowed within task flow definition files, but in most cases are not recommended because they may keep objects in memory longer than needed.

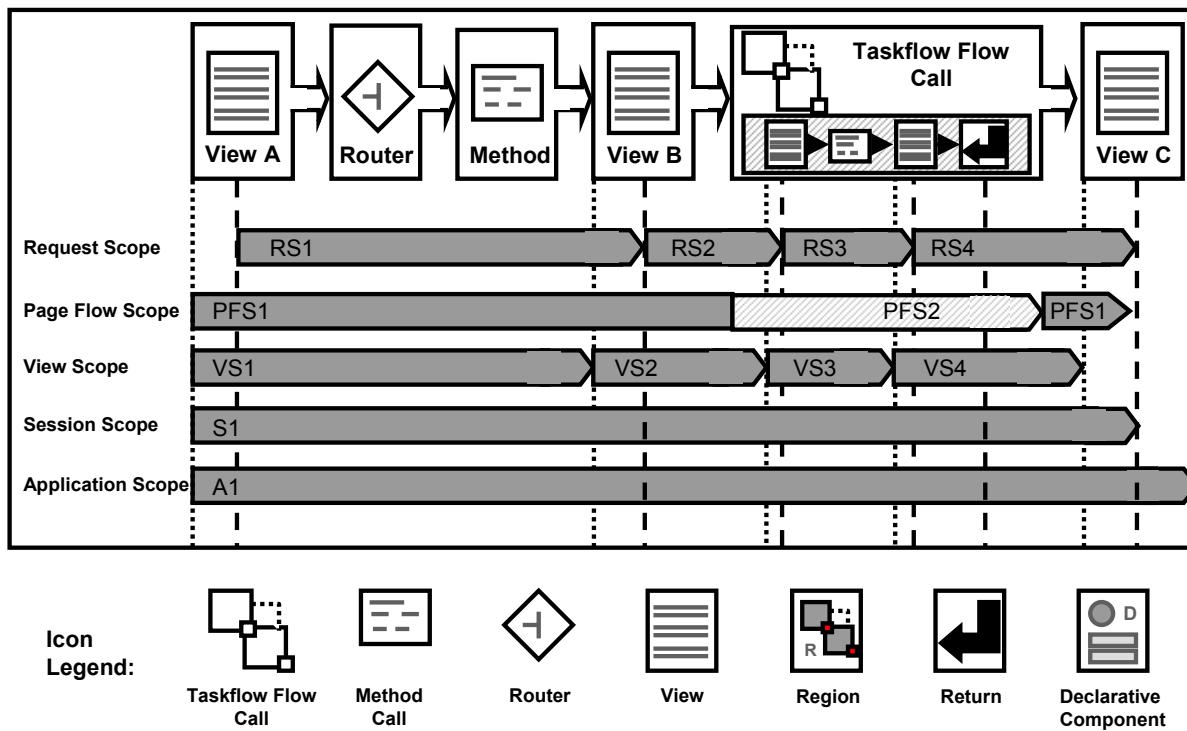
You can bind values to the `pageFlowScope` managed bean properties before control is passed to a called ADF bounded task flow. Values added as `pageFlowScope` continue to be available as the user navigates from one page to another within a task flow. This is true even if you use `<redirect />`. These values are visible only in the current page flow or process. If the user opens a new window and starts navigating, that series of windows has its own process; values stored in each window remain independent. Clicking the browser's Back button resets `pageFlowScope` to its original state.

Clearing `pageFlowScope`

This scope never clears itself. You can manually force `pageFlowScope` to clear, as shown in this example:

```
RequestContext afContext = RequestContext.getCurrentInstance();  
afContext.getPageFlowScope().clear();
```

Memory Scope Duration with a Called Task Flow



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Memory Scope Duration with a Called Task Flow

As stated previously, you should store values in the scope with the shortest duration that fits your needs, so it is important to know the duration of memory scopes in an application.

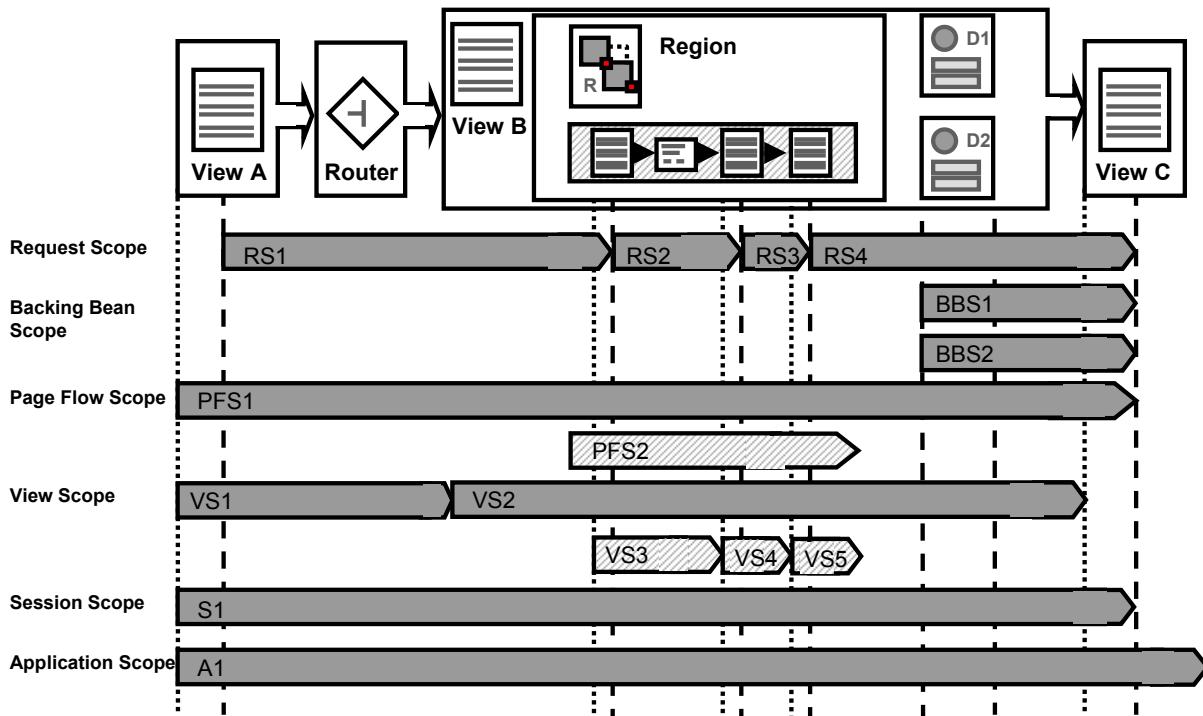
The parent task flow depicted in the slide contains three view activities, a router, a method call, and a task flow call. The called bounded task flow has two view activities: a method call and a task flow return.

The **request scope** line shows that the duration of a request scope is from the time a request is issued in a view activity until another view activity creates a request. Routers and method calls do not have a request scope of their own, so values set in RS1 above are available to the router and the method call. If the called task flow had no view activities, RS1 would span the task flow also. However, each of the called task flow's two view activities creates a request, ending the previous request scope and starting a new one. **View scope** is similar, except that it begins when a view is rendered and ends when a new view is rendered.

As mentioned earlier, the **page flow scope** attributes are accessible anywhere within a task flow. The called task flow has its own page flow scope, PFS2, that is separate from the parent's PFS1, and the called task flow cannot access PFS1, so parameters, which are discussed in this lesson, are needed to pass information from the parent to the called task flow.

Session scope lasts throughout the user session, while **application scope** lasts beyond the user session and can share values among all instances of an application.

Memory Scope Duration with a Region



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Memory Scope Duration with a Region

This example is similar to the previous one, except that the bounded task flow is contained in a region on the page that is represented as a view activity within the main task flow. It shows the following differences from the previous slide:

- The backing bean scope is available for the declarative components, and lasts as long as the component is in focus. Each component notifies the controller when it starts and ends being in focus. There is a separate backing bean scope for each declarative component.
- The page flow scope works differently in this example, because the second task flow is in a region on a view within the main task flow. So the duration of the page flow scope of the main task flow is throughout its entire flow, including the region that contains the other task flow. The bounded task flow represented as a region has its own page flow scope as well. Each task flow's scope is independent of and cannot access objects in the other task flow's scope.
- The view scope for the page fragments in the bounded task flow (in the region on the page represented by the view activity B) is separate from the view scope for the containing page. The views in the region cannot access the view scope of the containing page, and the containing page cannot access the view scopes of the page fragments in the region.

Accessing Memory-Scoped Attribute Values

You can access the pageFlowScope attributes by using:

- EL:

```
<af:commandButton  
text="#{pageFlowScope.buttonBean.label}"  
action="#{pageFlowScope.buttonBean.action}" />
```

- Java:

```
import java.util.Map;  
import  
org.apache.myfaces.trinidad.context.RequestContext;  
...  
Map pageFlowScope =  
RequestContext.getCurrentInstance().getPageFlowScope()  
;  
Object myObject = pageFlowScope.get("myObjectName");
```



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Accessing Memory-Scoped Attribute Values

You can access memory-scoped attributes in one of the following ways:

- **By using EL:** You can access objects stored in memory scopes through EL expressions. The object names must use the name of the scope as a prefix, because EL expressions do not automatically look into any memory scopes.
The first example in the slide shows how to have a button's label provided by a pageFlowScope managed bean, and to have a method on the bean called when the button is selected. (This example is just for illustrative purposes; in practice, you would use resource bundles for the text of labels.)
- **By using Java:** pageFlowScope is a java.util.Map that can be accessed from Java code. To get a reference to pageFlowScope, use the getPageFlowScope() method provided by Trinidad in org.apache.myfaces.trinidad.context.RequestContext. The second example in the slide retrieves an object from pageFlowScope.

Using Memory-Scoped Attributes Without Writing Java Code

```

<af:table
value="#{myManagedBean.allEmployees}"
var="emp" rowSelection="single">
<af:column headerText="Name">
<af:outputText value="#{emp.name}" />
</af:column>
<af:column
headerText="Department Number">
<af:outputText
value="#{emp.deptno}" />
</af:column>
<af:column headertext="Select">
<af:commandButton
text="Show more details"
action="showEmpDetail">
<af:setActionListener
from="#{emp}"
to=
"#{
pageFlowScope.empDetail}" />
</af:commandButton>
</af:column>
</af:table>

```

The Master page stores an employee row in the `empDetail` property of the `pageFlowScope` attribute.

```

<h:panelGrid columns="2">
<af:outputText value="Firstname:"/>
<af:inputText
value="#{
pageFlowScope.empDetail.name}" />
<af:outputText value="Email:"/>
<af:inputText
value="#{
pageFlowScope.empDetail.email}" />
<af:outputText value="Hiredate:"/>
<af:inputText
value="#{
pageFlowScope.empDetail.hiredate}" />
<af:outputText value="Salary:"/>
<af:inputText
value="#{
pageFlowScope.empDetail.salary}" />
</h:panelGrid>

```

The Detail page retrieves employee data from the `empDetail` property of the `pageFlowScope` attribute.

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Using Memory-Scoped Attributes Without Writing Java Code

The `af:setActionListener` tag provides a declarative way to cause an action source (for example, `af:commandButton`) to set a value before navigation. You can use memory-scoped attributes with `af:setActionListener` to pass values from one page to another, without writing any Java code in a backing bean.

Suppose you have a master page with a single-selection table showing employees, as shown in the first example in the slide:

1. The EL variable name `emp` is used to represent one row (employee) in the table.
 2. The `action` attribute value of `af:commandButton` is a static string outcome `showEmpDetail`.
 3. The `af:setActionListener` tag, which has two attributes `from` and `to`, takes the `from` value and stores it with the `to` value.
- When the user clicks the command button on a row, the action listener executes, and the value of `#{emp}` is retrieved. The retrieved row object is stored as the `empDetail` property of `pageFlowScope` with the `#{pageFlowScope.empDetail}` EL expression—you do not need to write any Java code in a backing bean. Then the `action` event executes with the static outcome, and the user is navigated to a detail page.
4. On the detail page, you refer to the `pageFlowScope.empDetail` objects to display more data for the current employee, as shown in the second example in the slide.

Overview of Parameters

- You can use:
 - Page parameters
 - Region parameters
 - Task flow parameters
 - View activity parameters
 - Task flow call activity parameters
- You set the parameter values by using expressions (not by just entering a value).
- The goal is increased reusability of pages and task flows.

Stored in the binding container; defined in the page definition file

Stored in the task flow .xml file; defined in Property Inspector

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Parameters

There are several types of parameters that you can use to pass information to different parts of the application. Subsequent slides discuss page parameters, region parameters, task flow parameters, view activity parameters, and task flow call activity parameters.

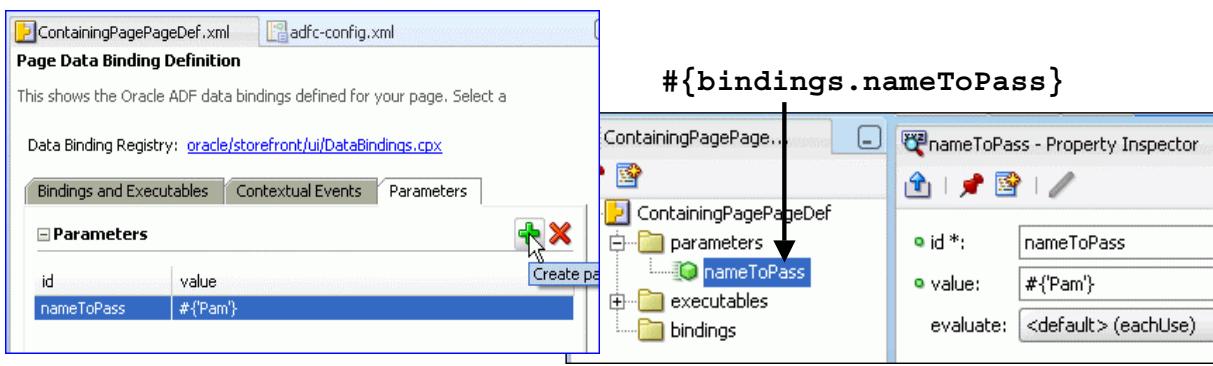
The examples discussed in the next few slides show how to use the first four types of parameters to pass a value from a containing page to a task flow represented as a region on that page. You actually can accomplish this task without using all four types of parameters. However, using more types of parameters enables you to make your pages and task flows more reusable, as shown in the examples.

After presenting this example, the remainder of the lesson discusses how to pass values to a called task flow and return values to the calling task flow.

Using Page Parameters

Page parameters are:

- Output parameters to pass a value *from* a page
- Stored in the binding container for the page
- Defined in the Structure window or Property Inspector for a page definition file:



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Using Page Parameters

You define page parameters in the page definition of the page. The example in the slide shows setting a page parameter to a String value by using Expression Language, because the value of the parameter must be an expression.

You could also set the page parameter to a value contained in a managed bean, in the binding container, or in a memory-scoped attribute. For example:

```
<parameters>
  <parameter id="myDeptId" value="#{DepartmentBean.deptId}" />
  <parameter id="fname" value="#{bindings.FirstName.InputValue}" />
  <parameter id="lname" value="#{pageFlowScope.lastName}" />
</parameters>
```

You can access the parameter value in EL as `#{bindings.<param name>}`.

You can use page parameters to, for example, pass values into a page template, enabling the use of the same template for a variety of different pages. Another example of using a page parameter is to set a value in a containing page to pass to a region (task flow) on the page.

Job of the Page Parameter

Receives a value and stores it in a parameter in the binding container:

| | Parameter type | Parameter name | Parameter value | Parameter function |
|----------------------------|----------------|------------------------|-----------------|--|
| Containing page parameter | output | nameToPass | #{'Pam'} | Receive a value and store it in the binding container. |
| | | | | |
| | | | | |
| | | | | |
| Page fragment in task flow | | #{pageFlowScope.pname} | | |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Job of the Page Parameter

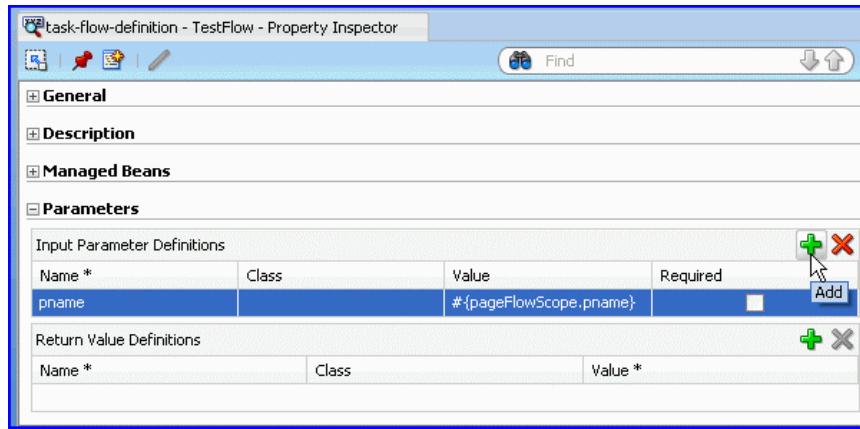
It is often necessary to pass values between pages. In this example, there is a bounded task flow that is represented as a region on a page. The task flow contains a view activity that represents a page fragment, and the requirement is to pass a value from the containing page to that page fragment.

As depicted in the table in the slide, the parameter of a containing page takes a value that is assigned to it and stores it in a parameter in the binding container. Subsequent slides build this table to show the functions of different types of parameters.

Using Task Flow Parameters

Task flow parameters are:

- Input parameters to pass a value *into* a task flow
- Defined in the .xml file for the task flow (You can use the Property Inspector for the task flow.)



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Using Task Flow Parameters

Building well-defined, parameterized task flows that can be used as regions on pages is a good way to provide reusable and highly maintainable segments of functionality across your application.

Parameters and page events are how different regions and portlets on a single page can communicate with each other and pass context to each other without being hard-wired together, or even knowing if the other one is present on the page. For example, a region that shows a graph of types of goods sold for a particular country could take a parameter into the region that specifies the country. That same region could also take in a parameter that specifies whether or not it should display the results as a graph or as a pie chart. Using parameters enables that same region to be reused in a variety of different situations.

You define task flow parameters in the task flow's .xml file. You can define a task flow parameter declaratively, as shown in the slide. The slide shows an example of an input parameter with the name `pname`, and value `#{pageFlowScope.pname}`. The parameter works as follows:

1. When the parameter `pname` is passed into the task flow, its value is stored in the page flow-scoped attribute `pname`.
2. Inside the task flow, activities and pages can get the value of the page flow-scoped attribute by using the EL expression `#{pageFlowScope.pname}`.

Using Task Flow Parameters (continued)

In the lesson titled “Responding to Application Events,” you learn how to set up contextual events to communicate between regions. You can use page parameters and task flow parameters in conjunction with events to coordinate regions.

Job of the Task Flow Parameter

Stores input value in a page-flow-scoped attribute:

| | Parameter type | Parameter name | Parameter value | Parameter function |
|-----------------------------|----------------|----------------|------------------------|--|
| Containing page parameter | output | nameToPass | #{'Pam'} | Receive a value and store it in the binding container. |
| Bounded task flow parameter | input | pname | #{pageFlowScope.pname} | Store parameter value in the memory-scoped attribute. |
| Page fragment in task flow | | | #{pageFlowScope.pname} | |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Job of the Task Flow Parameter

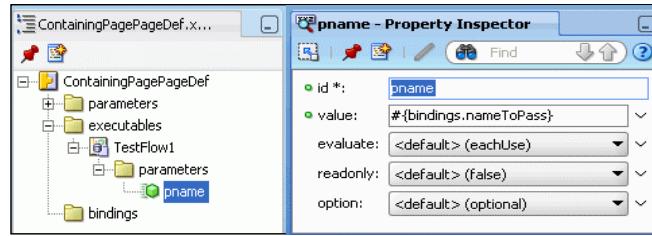
As previously stated and as depicted in the table in the slide, the bounded task flow parameter receives an input value and stores it in a memory-scoped attribute. The example shows storing it in `pageFlowScope`, so that the value of that attribute can then be used anywhere in the task flow, such as on a page fragment within the task flow.

But where does that input value come from? A task flow is independent of the page that contains it. It cannot receive a value directly from a page parameter of the containing page, so another mechanism is needed. In the example, there needs to be a way to pass the value in the `nameToPass` page output parameter into the `pname` task flow input parameter.

Using Region Parameters

Region parameters are:

- Input parameters to pass a value *into* a region on a page
- Stored in the binding container for the page
- Defined in the Structure window or Property Inspector for a page definition file:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Region Parameters

To pass a parameter from a containing page to a task flow, you need a region parameter, which you define as follows:

1. Define a page parameter and a task flow parameter as shown in the previous slides.
2. Pass the page parameter's value to the input parameter of the task flow. You do this by defining a region parameter in the page definition file of the containing page, either in the Structure window or, with the region selected in the Executables section, in the Property Inspector.
3. Then set the parameter's value to the EL value of the page parameter.

A simpler way to define a region parameter is when you create the region. If the task flow has input parameters defined on it, then when you drag a task flow to a page to create a region, you must define region parameters for the task flow parameters.

After you have defined the region parameters, you can access the value within a page on the contained region by using the expression `#{pageFlowScope.<input parameter name>}`, such as in the following example on the task flow's default page:

```
<af:outputLabel value="Hello, "/>
<af:outputText value="#{pageFlowScope.pname}"/>
```

Job of the Region Parameter

Passes the value from the page to its task flow:

| | Parameter type | Parameter name | Parameter value | Parameter function |
|-----------------------------|----------------|----------------|------------------------|---|
| Containing page parameter | output | nameToPass | #{'Pam'} | Receive a value and store it in the binding container. |
| Region parameter | output | pname | #{bindings.nameToPass} | Store binding value as parameter for input to a region. |
| Bounded task flow parameter | input | pname | #{pageFlowScope.pname} | Store parameter value in the memory-scoped attribute. |
| Page fragment in task flow | | | #{pageFlowScope.pname} | |

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Job of the Region Parameter

The region parameter acts as a bridge from the page parameter to the task flow. It takes the value stored in the page parameter and stores it in an output parameter, which should have the same name as the input parameter of the task flow. Now the task flow is able to receive that value and store it in the page flow-scoped attribute that can be used anywhere within that task flow.

Developing a Page Independently of a Task Flow

Page not aware of the task flow's page-flow-scoped attribute:

| | Parameter type | Parameter name | Parameter value | Parameter function |
|-----------------------------|----------------|----------------|------------------------|---|
| Containing page parameter | output | nameToPass | #{'Pam'} | Receive a value and store it in binding container. |
| Region parameter | output | pname | #{bindings.nameToPass} | Store binding value as parameter for input to a region. |
| Bounded task flow parameter | input | pname | #{pageFlowScope.pname} | Store parameter value in memory-scoped attribute. |
| Page fragment in task flow | | | #{pageFlowScope.ename} | |

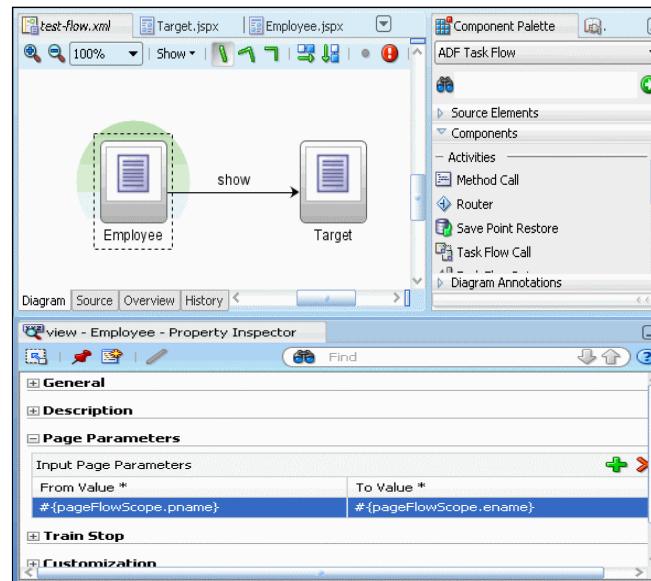
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Developing a Page Independently of a Task Flow

To make a page (or page fragment) reusable in different task flows, it should be independent of the task flow. The page flow-scoped attributes that are referenced on the page may not be named the same as the task flow's page flow-scoped attributes. So pages that are developed independently and used in different task flows need another mechanism to be able to use the values that are stored in page flow-scoped attributes for a particular task flow.

Using View Activity Parameters

Use the view activity parameters to pass values from a task flow to one of its pages:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Passing Parameters from a Task Flow to One of Its Pages

Instead of using task flow parameters directly on the pages that are in the task flow, you can make pages more reusable by using the view activity input page parameters to map the task flow's memory-scoped attributes to those expected by the page.

To define a view activity parameter, perform the following steps:

1. Select the view activity in the task flow diagram and in the Property Inspector, select the Page Parameters panel.
2. In the Input Page Parameters section, click Add.
3. Enter a **from-value** as an EL expression to specify where the parameter value is being passed from, such as `# {pageFlowScope . pname}`. If the value is coming from the task flow input parameter, it should match the value specified for the task flow input parameter.
4. Enter a **to-value** as an EL expression to specify where the page associated with the view activity can retrieve the value of the input page parameter, such as `# {pageFlowScope . ename}`.

This enables you to use a page in multiple task flows as the page for different views activities. The view activity acts as the mechanism for passing parameters from its task flow to its page.

Job of the View Activity Parameter

Populates the page-flow-scoped attribute expected by the page:

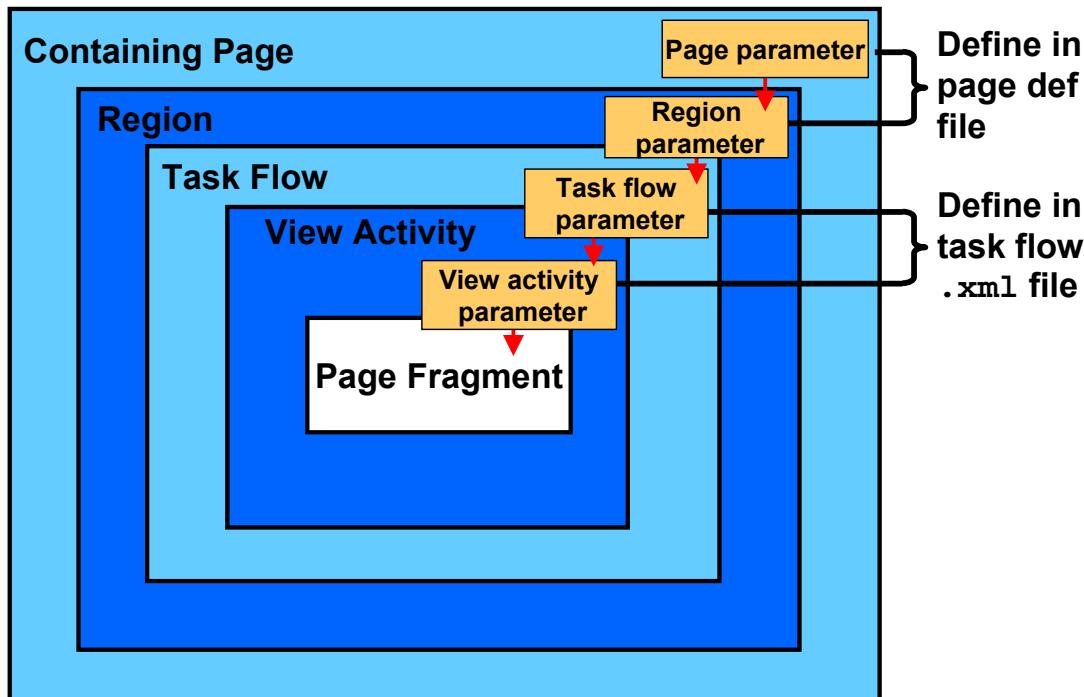
| | Parameter type | Parameter name | Parameter value | Parameter function |
|-----------------------------|----------------|------------------------|------------------------|---|
| Containing page parameter | output | nameToPass | #{'Pam'} | Receive a value and store it in the binding container. |
| Region parameter | output | pname | #{bindings.nameToPass} | Store binding value as parameter for input to a region. |
| Bounded task flow parameter | input | pname | #{pageFlowScope.pname} | Store parameter value in the memory-scoped attribute. |
| View activity parameter | internal | #{pageFlowScope.ename} | #{pageFlowScope.pname} | Move value from memory-scoped attribute into the memory-scoped attribute expected by the reusable page. |
| Page fragment in task flow | | #{pageFlowScope.ename} | | |

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Job of the View Activity Parameter

The view activity parameter simply takes the value that is stored in one memory-scoped attribute for the task flow and stores it in another—the memory-scoped attribute that is used by the page. In this way, you can integrate independently developed pages into multiple task flows without changing either the task flow or the page.

Summary: Passing a Value from a Containing Page to a Reusable Page Fragment in a Region



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Summary: Passing a Value from a Page to a Reusable Page Fragment in a Region

The more types of parameters you use, the more reusable your components are. The page fragment in this diagram is represented as a view activity in a task flow used as a region on a containing page, like the example that was just discussed. Because parameters are used at each level, the page can be developed independently of all other components without regard to the names of memory-scoped attributes that are used on the page:

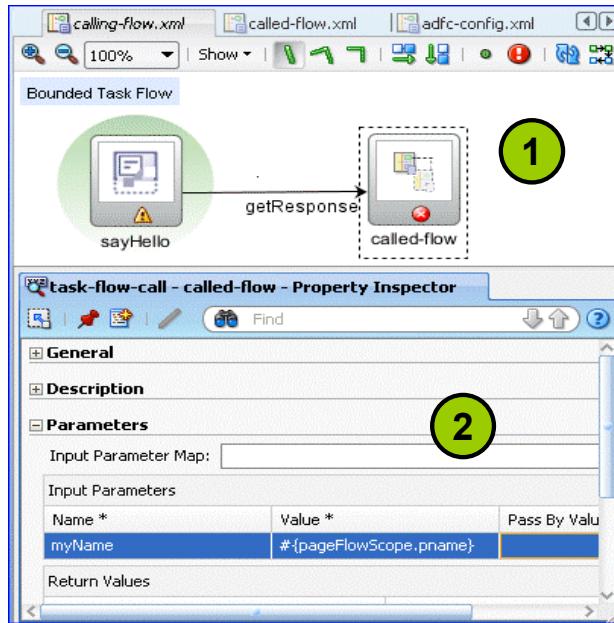
- The containing page uses a page parameter to pass its value to the region.
- The region uses a region parameter to pass the value on to the task flow, so the task flow can be developed independently of the page and reused on multiple pages.
- The task flow uses a task flow parameter to pass the value to its view activity.
- The view activity uses a view activity parameter to store the value in the memory-scoped attribute that is used by the page fragment.

All of these parameters could be named differently, as long as the output parameter of one level has the same name as the input parameter of the next level. For example, the region's output parameter would have the same name as the task flow's input parameter.

Page and region parameters are stored in the page's binding container, so you define them in the page definition file for the page. Task flow and view activity parameters are defined in the .xml file for the bounded task flow.

Passing Values to a Task Flow from a Task Flow Call Activity

1. Define an input parameter on the task flow call activity:



2. Define an input parameter on the called task flow.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Passing Values to a Task Flow from a Task Flow Call Activity

Another situation where you would need to pass values to a task flow is when you call one task flow from another. A called ADF bounded task flow can accept input parameters from another ADF unbounded or bounded task flow and can pass return values to the caller upon exit. To pass an input parameter to a bounded task flow, you must specify:

- Input parameter on the task flow call activity. This specifies where the calling task flow stores parameter values.
- Input parameter definition on the called bounded task flow. This specifies where the called bounded task flow can retrieve parameter values.

The input parameter name specified for each of the above options should be the same in order to map input parameter values back into the called bounded task flow. The value for each corresponds to the mapping between where the value will be retrieved within the caller and the called task flow.

In the input parameter definition for the called bounded task flow, you can specify whether an input parameter is required. If a required input parameter is not received, a run-time error occurs. An input parameter definition that is identified as not required can be ignored during the task flow call activity creation.

Passing Values to a Task Flow from a Task Flow Call Activity (continued)

By default, input parameters are passed by reference, although you can select the pass-by-value check box in the Property Inspector for the task flow call activity to pass by value. Mixing the two, however, can lead to unexpected behavior in cases where parameters reference each other.

Task flow call activity input parameters can be passed by reference only if managed bean objects are passed, not individual values, so the pass-by-value check box applies only to managed bean objects and is used to override the default setting of passing by reference. Individual values are only passed by value.

If you call a bounded task flow using a URL rather than a task flow call activity, you pass parameters and values on the URL itself.

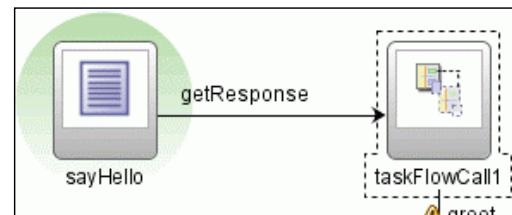
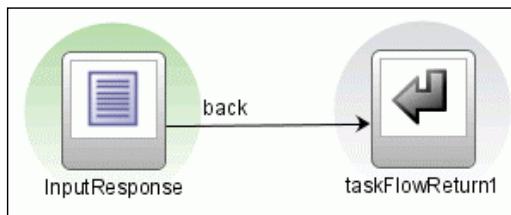
To specify input parameters on the task flow call activity, perform the following steps:

1. Select the task flow call activity in the task flow diagram.
2. In the Property Inspector, click Parameters and expand the Input Parameters section:
 - Click Add and enter a name for the parameter.
 - Enter a parameter value—for example, `# {pageFlowScope.callingTaskflowParm}`. The value specifies where the parameter value will be taken from within the calling task flow.
 - Optionally, select the pass-by-value check box. (By default, parameters are passed by reference to the called bounded task flow.)

After you have specified an input parameter for the task flow call activity, you can specify an input parameter definition for the called ADF bounded task flow as explained previously.

Returning Values to a Calling Task Flow

- To return a value, you must specify:
 - Return value definitions on the called task flow
 - Return values on the task flow call activity in the calling task flow
- Names must match.



| Return Value Definitions | | |
|--------------------------|-------|-------------------------|
| Name * | Class | Value * |
| newName | | #(pageFlowScope.myName) |

| Return Values | | |
|---------------|-------------------------|--|
| name * | value * | |
| newName | #(pageFlowScope.myName) | |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Returning Values to a Calling Task Flow

A task flow return activity causes the ADF bounded task flow to return to the task flow that called it. The called bounded task flow can pass return values back to the calling task flow.

To return a value, you must specify:

- Return value definitions on the called bounded task flow. These specify where the return value is to be taken from upon exit of the called bounded task flow.
- Return values on the task flow call activity in the calling task flow. These specify where the calling task flow can find return values.

The caller of the bounded task flow can choose to ignore the return value definition values by not identifying any task flow call activity return values back to the caller.

Return values on the task flow call activity are passed back by reference. Nothing inside the ADF bounded task flow still references them, so there is no need to pass by value and make a copy.

Returning Values to a Calling Task Flow (continued)

To specify a return value for a called ADF bounded task flow, perform the following steps:

1. In the task flow editor, open the called ADF bounded task flow.
2. On the Overview tab, select the Parameters panel and click Add in the Return Value Definitions section to add the return value:
 - In the name field, enter a name to identify the return value—for example, Return1.
 - In the value field, enter an EL expression that specifies where the return value is to be taken from upon exit of the called bounded task flow—for example, `# {pageFlowScope.ReturnValueDefinition}`.

To specify a return value for the calling task flow, perform the following steps:

1. In the task flow editor, open the calling ADF task flow and in the task flow diagram, select the task flow call activity.
2. In the Property Inspector, select the Parameters panel and click Add in the Return Values section.
 - In the name field, enter a name to identify the return value—for example, Return1. The name of the return value must match the name of the return value definition on the called task flow definition.
 - In the value field, enter an EL expression that specifies where the calling task flow can find return values—for example, `# {pageFlowScope.ReturnValue}`.

Tip: If you drop the bounded task flow that you intend to call on the task flow call activity, the name field is already specified. Therefore, the name field for the return value automatically matches the name of the return value definition on the called task flow definition.

Deciding Which Type of Parameter to Use

- Parameters are designed to:
 - Improve reusability
 - Encapsulate functionality
- You can use more than one type of parameter to achieve these goals.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Deciding Which Type of Parameter to Use

As you have seen in this lesson, the use of one type of parameter does not preclude the use of another, because they all have different functions. Defining parameters is all about providing reusability. Defining parameters on a task flow improves reusability of that task flow. Defining parameters on a page or page fragment improves its reusability. Parameters also make an object much more encapsulated (self-contained).

For example, if you have a bounded task flow that updates customers, you could have it take in CustomerId as a parameter. The task flow can then make sure that the correct customer is queried to edit—and there is no reliance on the consumer of that bounded task flow setting the current row in a view object in some application module that this bounded task flow uses. This ensures that the interface between consumer and producer is much better documented and separated from the internal implementation.

If you want to make the page a more reusable component, you may provide a parameter on the page for exactly the same reasons.

Summary

In this lesson, you should have learned how to:

- Define the data model to reduce the need to pass values
- Use a managed bean to hold values
- Store values in memory-scoped attributes
- Use parameters to pass values



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 17 Overview: Passing Values Between Pages

This practice covers the following topics:

- Conditional Rendering Based on Parameters
- Using Parameters for Dynamic Breadcrumbs
- Implementing the Add to Cart Functionality
- Implementing Create Supplier Functionality



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 17 Overview: Passing Values Between Pages

In this practice, you modify the ProductCatalog page to display only one of its three tables, depending on the value of parameters passed to the page. You place all of the tables under a single panel header whose title depends on the input parameters for the page.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Osi S.R.L. use only

18

Responding to Application Events

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Configure and use managed beans to contain code to respond to events
- Describe the different types of events
- Use phase listeners
- Create event listeners
- Define action listeners
- Create value change listeners
- Describe ADF Faces enhanced event handling
- List other types of server events used by ADF Faces components
- Explain how ADF components use AJAX events
- Use contextual events to coordinate regions



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

Web applications are often required to respond to a user interaction or other events. This lesson covers events for UI components. ADF enables you to define actions that take place when certain events occur, such as when a page loads or when the user clicks a button. Events may trigger a change in the user interface, invoke back-end application code, or both.

Adding UI Code

Managed beans:

- Are configured in the `adf-c-config.xml` or other task flow `.xml` file
- Consist of Plain Old Java Objects (POJOs), Lists, and Maps
- Have a no-argument constructor
- Use lazy initialization by JavaServer Faces framework “as needed”

The red bar spans most of the width of the slide, centered horizontally.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Adding UI Code

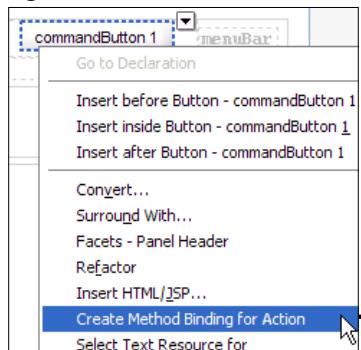
You can add UI code to the task flow by using managed beans.

Managed beans are Java classes with a no-argument constructor that you register with the application in the task flow `.xml` files. When the JSF application starts up, it parses these files and makes the beans available to be referenced in EL expressions (you learn about these in the lesson titled “Implementing Transactional Capabilities”) or Java code, enabling access to the beans’ properties and methods.

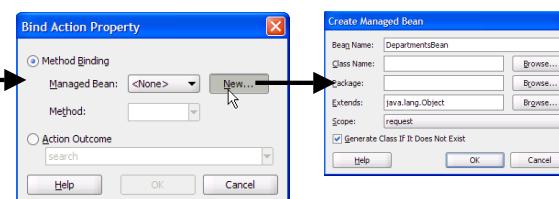
Whenever a managed bean is referenced for the first time and it does not already exist, the Managed Bean Creation Facility instantiates the bean by calling the default constructor method on the bean. If any properties are also declared, they are populated with the declared default values.

Creating Managed Beans

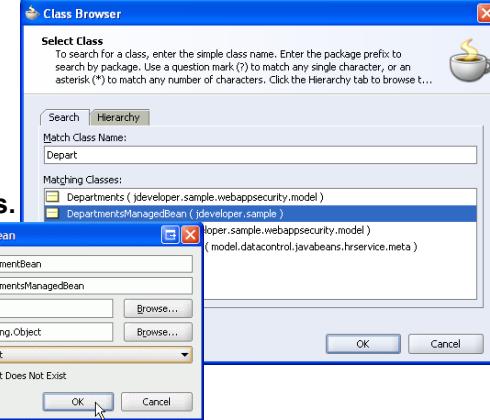
1. Right-click the command button.



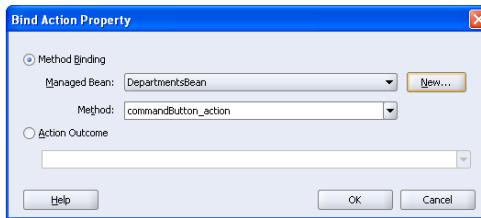
2. Browse for an existing Java class or create a new class.



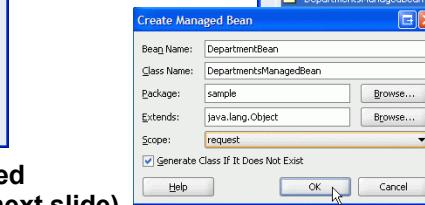
3. Search class.



5. Create the method.



4. Select the class.



6. The bean is automatically configured in the adfc-config.xml file (see next slide).

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Managed Beans

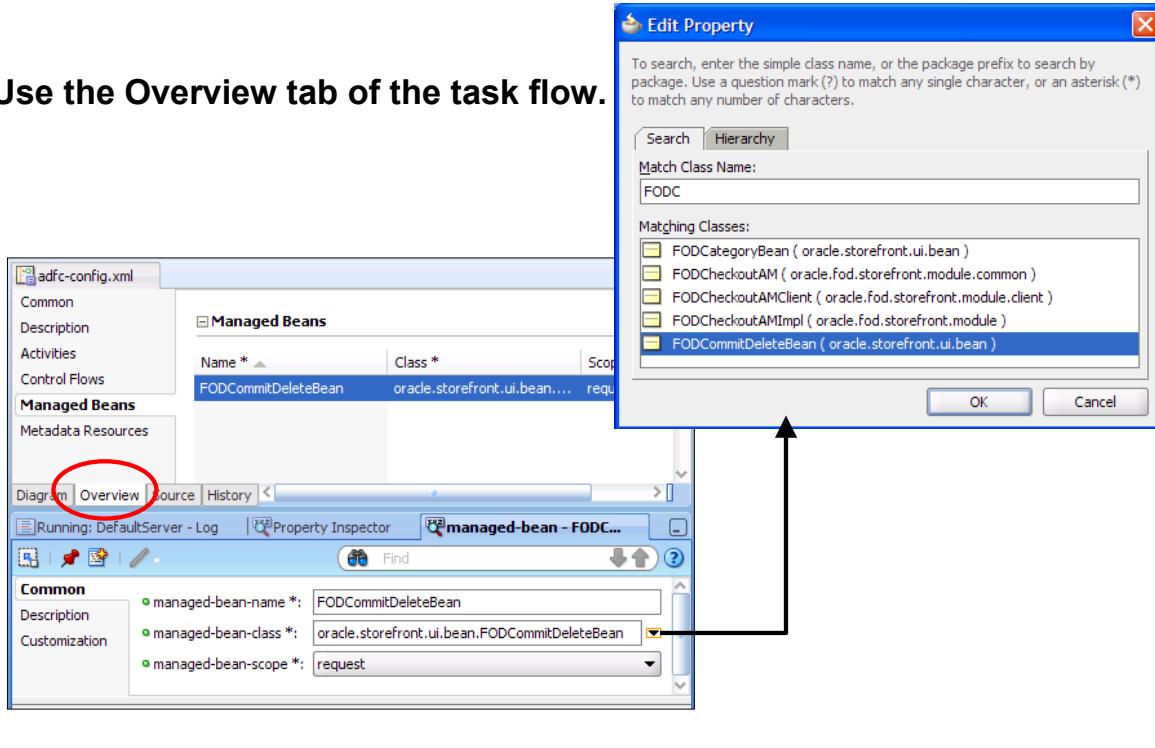
You can create managed beans as Java classes in a project before configuring them as a managed bean, or you can create them when binding a command button by performing the following:

1. Right-click the command button and select “Create Method Binding for Action” to launch the binding editor.
2. Choose to create a new class as a managed bean or to reuse an existing class.
3. To reuse an existing class, select a bean or search the class path.
4. If searching the class path, select the Java class you want to become a managed bean and click OK.
5. Click OK to create the managed bean.
6. The class is configured as a managed bean and the new method is created and bound to the command button (see next slide).

This creates the class and registers it as a managed bean in the `adfc-config.xml` file.

Registering Existing Java Classes as Managed Beans

Use the Overview tab of the task flow.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Registering Existing Java Classes as Managed Beans

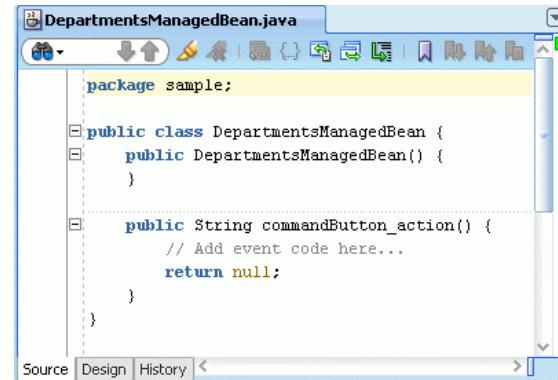
To register an existing class as a managed bean, open the task flow in the editor, click the Overview tab, select the Managed Beans panel, and click Create. Enter values for the managed bean attributes in the Property Inspector, browsing to locate the existing class.

Configuring Managed Beans

Entry in adfc-config.xml:

```
<managed-bean>
  <managed-bean-name> DepartmentBean </managed-bean-name>
  <managed-bean-class> sample.DepartmentsManagedBean
</managed-bean-class>
  <managed-bean-scope> request </managed-bean-scope>
</managed-bean>
```

Skeleton for new managed bean (add your own code):



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Managed Beans in adfc-config.xml

Managed beans can be defined only in ADF Controller source files (adfc-config.xml or bootstrap source files, and task flow definition files). They are not allowed in the faces-config.xml file. Managed beans are defined in the .xml files by using the <managed-bean> tag. Attributes of the managed bean, as shown in the example in the slide, are as follows:

- <managed-bean-name>: Specifies the name that is used to refer to the managed bean throughout the application
- <managed-bean-class>: Contains the fully qualified class name for the JavaBean
- <managed-bean-scope>: Defines the scope of the JavaBean. The possible values for this element are application, session, request, pageFlow, view, backingBean, or none. (These are described in the lesson titled “Passing Values Between UI Elements.”) If the value of the managed-bean-scope element is something other than none, the JavaBean created is stored in the corresponding object. For example, if the value is request, the JavaBean is stored in the request object of a given user.

If you have created a new Java class for your bean, JDeveloper creates the skeleton code to which you need to add your own code.

Referencing Managed Beans

Use in Expression Language on JSF Page:

```
<af:inputText  
value="#{DepartmentBean.firstName}" />
```

Use in Java code:

```
FacesContext facesContext =  
    FacesContext.getCurrentInstance();  
ELContext elContext = facesContext.getELContext();  
ExpressionFactory expressionFactory =  
    facesContext.getApplication().getExpressionFactory();  
ValueExpression exp =  
    expressionFactory.createValueExpression(elContext,  
    "#{DepartmentBean}", DepartmentsManagedBean.class);  
DepartmentsManagedBean dept =  
(DepartmentsManagedBean) exp.getValue(elContext);
```



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Referencing Managed Beans

You can reference managed beans by using either Expression Language (EL) or Java, as shown in the examples in the slide. Expression language is described in the lesson titled “Implementing Transactional Capabilities.”

What the Java code in the slide does is to assign the value of a managed bean (whose name is expressed in EL) to a User object. It uses the following classes and methods:

- `FacesContext` is a class that contains all of the per-request state information related to the processing of a single JSF request, and the rendering of the corresponding response. It is passed to, and potentially modified by, each phase of the request processing life cycle. (You learn about life-cycle phases in the next few slides.)
- `ELContext` is a class that is used to evaluate an expression.
- `ExpressionFactory` is a class that parses a `String` into a value expression (as in the code in the slide) or a method expression for later evaluation.
- The `createValueExpression()` method parses expressions that evaluate to values. The expression must be first parsed, and then evaluated.
- The `ValueExpression` class encapsulates a parsed expression, and its `getValue()` method evaluates the expression relative to the provided context, and returns the resulting value.

Describing JSF and ADF Life-Cycle Roles

The JSF life cycle handles presentation:

- Submission of values on the page
- Validation for components
- Navigation
- Displaying the components on the resulting page
- Saving and restoring state

The ADF life cycle handles data:

- Preparing and updating the data model
- Validating the data at the model layer
- Executing methods on the business layer



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

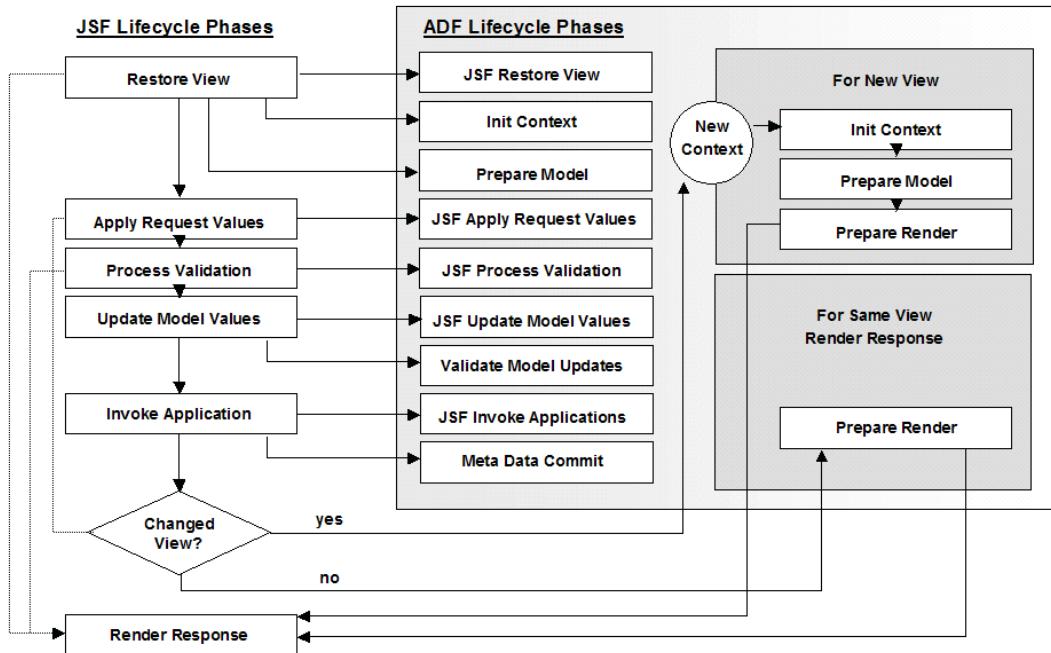
Describing JSF and ADF Life-Cycle Roles

When a page is submitted and a new page requested, the application invokes both the JSF request life cycle and the ADF page life cycle.

The JSF life cycle handles the submission of values on the page, validation for components, navigation, and displaying the components on the resulting page and saving and restoring state. The JSF life-cycle phases use a UI component tree to manage the display of the faces components. This tree is a run-time representation of a JSF page: each UI component tag on a page corresponds to a UI component instance in the tree. The `FacesServlet` object manages the request processing life cycle in JSF applications. `FacesServlet` creates an object called `FacesContext`, which contains the information necessary for request processing, and invokes an object that executes the life cycle.

The ADF life cycle handles preparing and updating the data model, validating the data at the model layer, and executing methods on the business layer. The ADF life cycle uses the binding container to make data available for easy referencing by the page during the current page request.

Coordinating JSF and ADF Life Cycles



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Coordinating JSF and ADF Life Cycles

You learned about the JSF life-cycle phases in the lesson titled “Understanding UI Technologies.” The ADF life cycle also contains phases that are defined simply to notify ADF life-cycle listeners before and after the corresponding JSF phase is executed (that is, there is no implementation for these phases), shown in the slide as ADF phases beginning with “JSF.” This enables you to create custom listeners and register them with any phase of both the JSF and ADF life cycles, so that you can customize the ADF page life cycle if needed.

Combining what you learned previously about the JSF life cycle with the ADF life cycle, a page goes through the following phases:

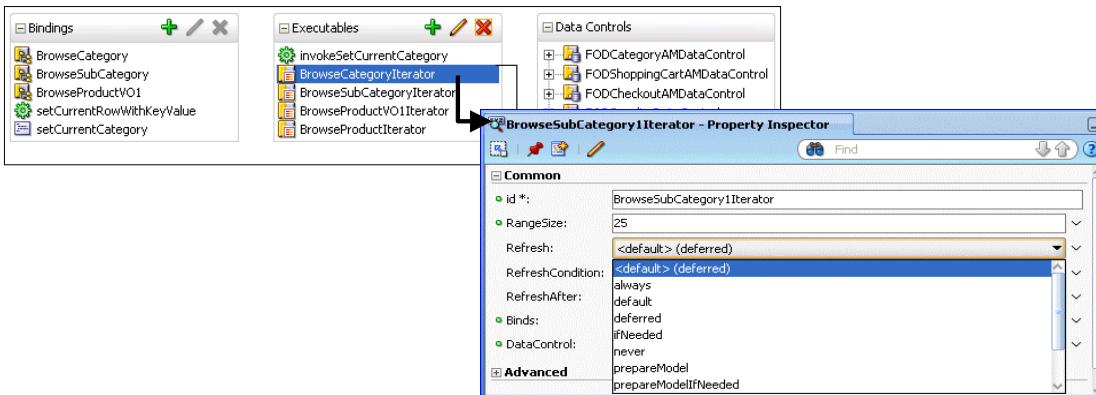
- **Restore View:** The URL for the requested page is passed to `bindingContext`, which finds the page definition file that matches the URL. The component tree of the requested page is either newly built or restored. All the component tags, event handlers, converters, and validators on the submitted page have access to the `FacesContext` instance. If it is a new empty tree (that is, there is no data from the submitted page), the page life cycle proceeds directly to the Render Response phase.
- **Initialize Context:** The page definition file is used to create the binding context. The `LifecycleContext` class is initialized with values for request, binding container, and life cycle.

Coordinating JSF and ADF Life Cycles (continued)

- **Prepare Model:** This phase prepares and evaluates any page or task flow parameters and refreshes application executables.
- **Apply Request Values:** Values provided in components that hold a value (such as input fields) have their values applied to their counterparts in the view tree. All events such as ValueChangeEvent s (VCE) or ActionEvent s (AE) are queued. If a component has its immediate attribute set to true, then validation, conversion, and events associated with the component are processed during this phase.
- **Process Validation:** Conversion and validation logic is executed for each component. This means both built-in validation/data conversion and custom validation/conversion are added onto the components. If a validation error is reported, an exception is thrown. The life cycle halts and the response is rendered with validation error messages. At the end of this phase, new component values are set, any validation or conversion error messages and events are queued on FacesContext, and any value change events are delivered.
- **Update Model:** The component's validated local values are moved to the model and the local copies are discarded. If you are using a backing bean for a JSF page to manage your UI components, any managed bean properties that are value-bound to a UI component using the value attribute are updated with the value of the component.
- **Validate Model Updates:** The updated model is now validated against any validation routines set on the model. Exceptions are caught by the binding container and cached.
- **Invoke Application:** Any action bindings for command components or events are invoked. Navigation is handled here depending on the outcome of the action method (if any).
- **Metadata Commit:** Run-time metadata changes to the model are committed, which stores any run-time changes made to the application using the Metadata Service (MDS).
- **Prepare Render:** The binding container is refreshed to allow for any changes that may have occurred in the Apply Request Values or Validation phases. Any iterators that correspond to read-only components (such as an outputText component) are refreshed. The prepareRender event and the afterJsfRenderResponse event are sent to all registered listeners.
Note: Instead of displaying prepareRender as a valid phase for a selection, JDeveloper displays renderModel, which represents the refresh (RENDER_MODEL) method called on the binding container.
- **Render Response:** The components in the tree are rendered as the Java EE Web container traverses the tags on the page. State information is saved for subsequent requests and the Restore View phase.

Specifying When to Refresh Binding Executables

- Refreshing the iterator reconnects the binding with the RowSetIterator object.
- Refreshing the invoke action binding invokes the action.
- You can set the Refresh property to determine when to refresh.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Refreshing Binding Executables

Refreshing Iterator Bindings

For scalability reasons, at run time the iterator bindings in the binding container release any reference they have to a row set iterator at the end of each request. During the next request, the iterator bindings are refreshed to again point at a “live” row set iterator that is tracking the current row of some data collection. The act of refreshing an ADF iterator binding during the ADF page life cycle is the operation of accessing the row set iterator to reunite the binding to the row set iterator to which it is bound.

Refreshing Invoke Action Bindings

Refreshing an invoke action binding invokes the action.

Determining When to Refresh

You can use the Refresh property on iterator bindings and invoke action executables in your page definition file to control when each is evaluated during the ADF page life cycle. The valid values for the Refresh property are as follows:

- **deferred** (the default): Enforces execution whenever demanded the first time, such as when a binding value is referenced in an EL expression. After it is called, it does not reexecute unless the binding itself or any parameter values for the binding have changed.
- **prepareModel**: Refreshes just during the Prepare Model phase

Refreshing Binding Executables (continued)

- `renderModel`: Refreshes just during the Prepare Render phase
Note: The key distinction between the Prepare Model phase and the Prepare Render phase is that one comes before JSF's Invoke Application phase, and the other after. Because JSF's Invoke Application phase is when action listeners fire, if you need your iterator to refresh after these action listeners have performed their processing, you will want to set the Refresh property to `renderModel`.
- `ifNeeded`: Refreshes the iterator only if it has not been refreshed, based on the refresh condition or if no condition is specified, during the Prepare Model phase
- `prepareModelIfNeeded`: Same as `ifNeeded`, during the Prepare Model phase
- `renderModelIfNeeded`: Same as `ifNeeded`, during the Render Model phase
- `RefreshAfter`: Specifies that one executable refreshes after another (you also then need to set the `RefreshAfter` property in addition to the `Refresh` property). `RefreshAfter` is used to handle dependencies between executables.
- `never` (valid for iterator bindings only): Refreshes only when your own code calls `getRowSetIterator()` on the iterator binding
- `always` (valid for iterator bindings only): Iterator always refreshed during both the Prepare Model and Prepare Render phases

Note: Refreshing an iterator binding does not forcibly reexecute its query each time. The first time the view object instance's row set iterator is accessed during a particular user's unit of work, this implicitly executes the view object's query if it was not already executed. Subsequent refreshing of the iterator binding related to that view object instance on page requests that are part of the same logical unit of work only accesses the row set iterator again, but does not forcibly reexecute the query. When you want to reexecute the query to refresh its data, use the `Execute` or `ExecuteWithParams` built-in operation, or programmatically call the `executeQuery()` method on the iterator binding.

Specifying Whether to Refresh Binding Executables

- Set RefreshCondition to an EL expression evaluating to true (default) or false.
- Example: A search page that initially shows no data has iterator binding with settings:
 - RefreshCondition= "#{!adfFacesContext.isInitialRender}"
(false when page is first rendered)
 - Refresh="renderModel"



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Specifying Whether to Refresh Binding Executables

The Refresh property determines the phase in which to invoke the executable, whereas the RefreshCondition property determines whether to refresh the binding. Before refreshing any bindings, the ADF run-time evaluates the RefreshCondition attribute of the executables, which specifies the conditions under which the executable should be refreshed. You can specify the RefreshCondition value using a Boolean EL expression (default is true).

If an iterator binding is not refreshed during the life cycle, it is not pointing to any row set iterator for that request. This results in the value bindings related to that iterator binding not having any data to display. This can be a desirable result when, for example, you want a page, such as a search page, to initially show no data. To achieve this, you can use the expression `#{adfFacesContext.isInitialRender}`, which evaluates to true when a page is first rendered. So that data does not display when the page is first rendered, use a RefreshCondition of: `#{!adfFacesContext.isInitialRender}`.

You should set the Refresh attribute of an executable to renderModel when the RefreshCondition is dependent on the model. So if you want to use the `#{adfFacesContext.isInitialRender}` expression in a RefreshCondition of an executable, you must set the Refresh property to either renderModel or renderModelIfNeeded, so that the executable refreshes during the Prepare Render phase.

Describing Types of Events

JSF supports:

- Phase events
 - Execute as part of the JSF and ADF life cycle
 - Can be used to augment standard behavior
- Action events
 - Occur when a command component is activated, such as when a user clicks a button or a link
 - Return a control flow outcome
- Value change events
 - Occur when the local value of a input component changes, such as when a user selects a check box
 - Are used for managing UI elements



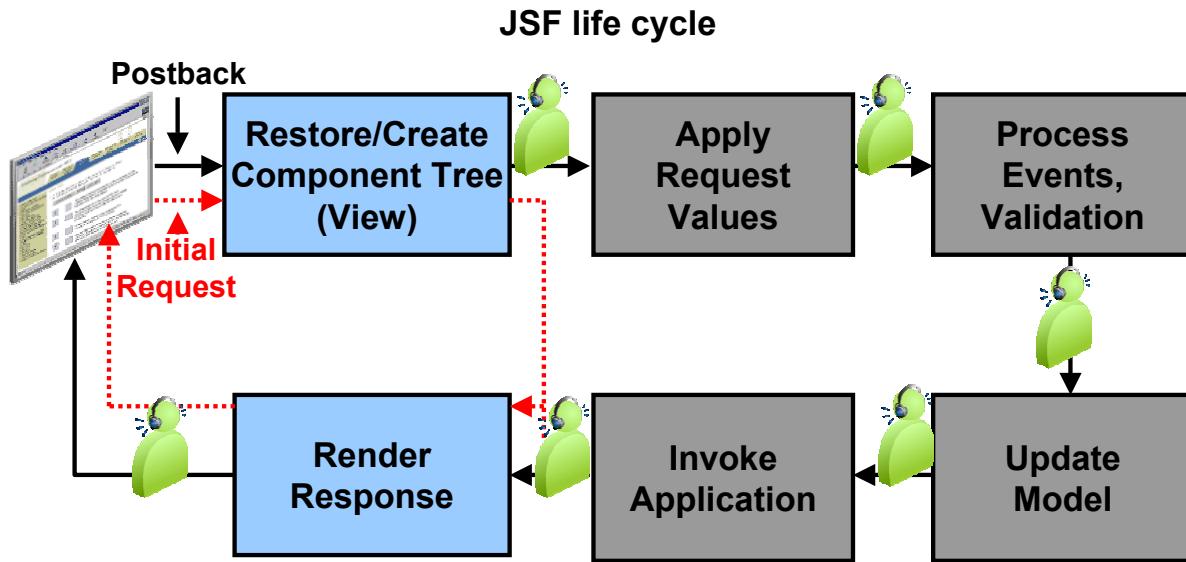
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Types of JSF Events

In traditional JSF applications, event handling typically takes place on the server. JSF event handling is based on the JavaBeans event model, where event classes and event listener interfaces are used by the JSF application to handle events generated by components. For a JSF application to be able to respond to user events, you typically register event listeners on the components that would generate events. The following types of events are supported:

- A phase event or listener is executed as part of the standard JSF life cycle. You can add to the standard behavior by implementing your own phase listener.
- An action event occurs when a command component is activated. For example, when a user clicks a button or a link, the form containing the component is submitted, and the component creates an event object that stores information about the event and identifies the component that generated the event. The event is also added to an event queue. At the appropriate time in the JSF life cycle, JSF tells the component to broadcast the event to the appropriate registered listener, which invokes the listener method that processes the event. Events may trigger a change in the user interface or invoke the back-end application code, or both. The return value of an action event is usually used for navigation.
- A value change event occurs when the local value of an input component changes; it fires only after component value validation and form submission.

Using Phase Listeners



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Phase Listeners

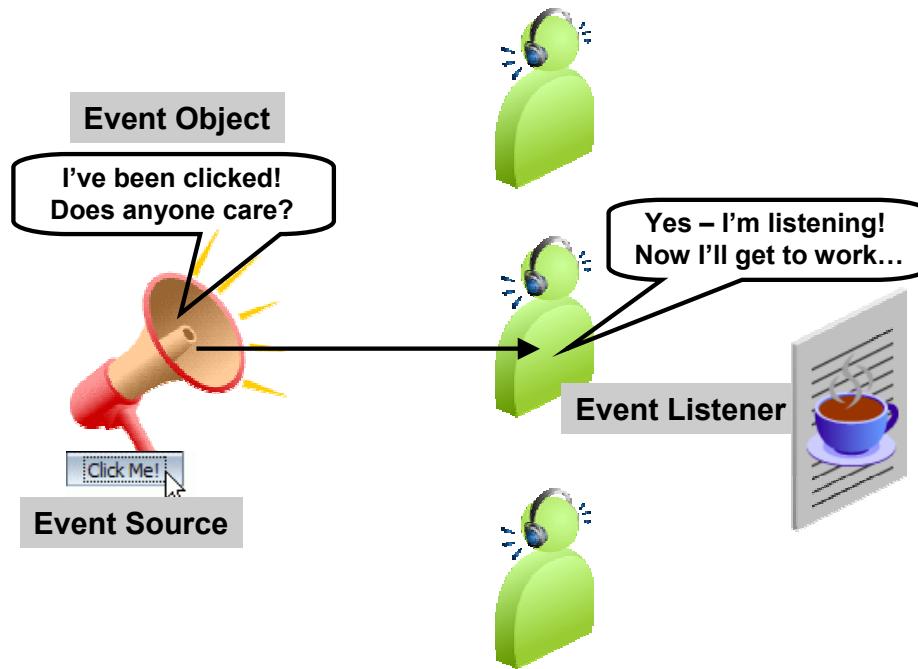
It is possible to create event listeners, called phase listeners, that are triggered at different stages of the JSF life cycle. This enables you to execute code processing that should occur—for example, before the rendering of a page.

Using a phase listener, you can listen for phase events for any of the ADF life-cycle phases, and execute your own functionality before and after the events. The basic steps for implementing your own phase listener in an ADF application are as follows:

1. Create a class that implements the `javax.faces.event.PhaseListener` interface.
2. Add your own functionality in the `beforePhase()` and `afterPhase()` methods of your class.
3. Use the Overview tab to register your listener with your `faces-config.xml` file.

You can accomplish similar functionality, but for a single page, by using a page phase listener that implements the `oracle.adf.controller.v2.lifecycle.PagePhaseListener` interface.

Using Event Listeners



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Event Listeners

Components can generate events, which are occurrences that take place as users interact with an application, such as when they click a button or change a value. There are also events, such as gaining or losing focus, that are not the result of user interactions.

If there is a listener for the event, the event communicates some information about what happened, so that the listener can respond. An event, then, is a component's way of letting a listener know that something happened.

An event listener is a class that implements one of the listener interfaces and contains code to respond to the event. A component has mechanisms to register and deregister event listeners for the types of events that are generated by the component.

Responding to Action Events

- Command components raise action events.
- Code to respond to the action event can be in the backing bean or external class.
- Stub code is generated by JDeveloper on demand.
- The action is registered in the page source.
- Action events are called in the Invoke Application phase of the life cycle.
- Action methods are the last to execute after other listeners, such as valueChange events.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Responding to Action Events

Action events are events that are associated with command components (such as commandButton, commandLink, or commandMenuItem).

An action event enables you to define an action method that returns an outcome to be used for navigation. Instead of entering a control flow outcome directly in the action property of a command component, you refer to a bean method that returns a string that corresponds to the control flow outcome. An example is shown in the next slide.

Action method code can be either in the page-backing bean or in a separate class. The benefit of having the code in the backing bean is that the code that is associated with a specific page is all local to the page (single source). However, the benefit of a separate class is reusability. For example, if you have standard code that should be executed regardless of the page, the separate class approach makes more sense. If you use the backing bean approach, JDeveloper creates a method stub for you; all you have to do is add your custom code. JDeveloper also registers the action in the page source.

When the page is submitted, the action method is added to the queue of application events to be executed. Action methods are the last to fire in the application phase of the life cycle. For this reason, they are generally used to determine navigation.

Creating Action Methods

- To create an action method:
 1. Invoke the action binding editor.
 2. Select an existing bean and a method, or create new.
- JDeveloper:
 - If the bean does not exist, creates Java class and registers as a managed bean
 - If the method does not exist, creates a method stub in the managed bean class
 - Adds the method to the `action` property of component
- Add your own code.
- Action methods return a `String` outcome.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Action Methods

You can create an action in JDeveloper by performing the following steps:

1. Invoke the action-binding editor by either double-clicking the component in the editor or by right-clicking it in the Structure window and select “Create Method Binding for Action.”
2. In the action-binding editor, you can accept the proposed method name or change it to a name of your choice. You can select to create a new bean and method if desired.

You then add whatever code you need to the action method. Remember that action methods can be used for controlling the UI, modifying values, or navigation; however, action methods return a `String` outcome.

For example, the `myBean.selectWhere` method depicted in the slide could have code similar to the following:

```
public String selectWhere() {  
    if <some condition> {  
        return ("cart"); /* navigate to shopping cart */  
    }  
    else {  
        return("prod"); /* navigate to Product page */  
    }  
}
```

Using Action Listeners

Action listeners differ from action methods:

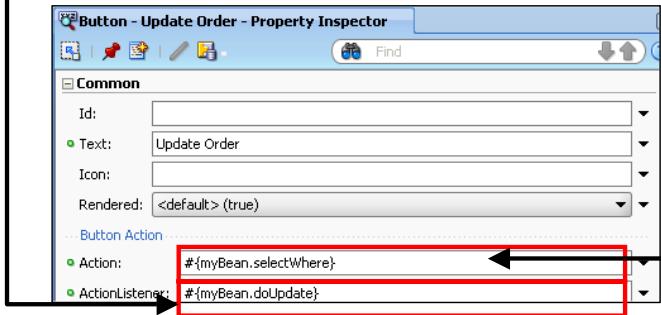
Action listeners:

- Contain code to respond to an action
- Do not return a value
- Execute after value change listeners

However, both can be initiated by the same action, such as a button click.

Action methods:

- Are used for navigation
- Return a String outcome
- Execute last



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Action Listeners

Command components, such as buttons and links, generate actions when clicked. Action listeners are event listeners that define code to execute in response to that action. Action listeners contain code that processes the action event object passed to it by the command component.

Value Change Events

- Input components raise value change events.
- Code for the value change listener can be in the backing bean or external class.
- Stub code is generated by JDeveloper.
- The value change event is registered in the page source.
- Value change events are processed in the Invoke Application phase of the life cycle.
- Value change events fire before action events.

The red bar spans most of the width of the slide, centered horizontally.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Value Change Events

Value change events are attached to input and select components. When there is a change to the value of a component, a value change event is added to the queue of application events to be executed. Just like action events, the code can be either in the backing bean or in a separate class. JDeveloper creates the method stub in the backing bean and registers the listener in the page source by using the `valueChangeListener` property of the component tag.

The registered value change listener is added to the queue of listeners to be processed before any action events. Action events, which are linked to navigation, are executed last.

Listening for Value Change Events

- Select the input component in the visual editor.
- Edit the `valueChangeListener` property and select a bean and method, or create a new one.
- JDeveloper:
 - Creates the method in the managed bean if it does not already exist
 - Adds the method to the `valueChangeListener` property in the page source
- Add your code.

The red bar spans most of the width of the slide, centered horizontally.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

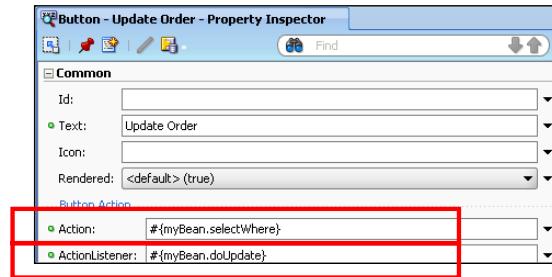
Listening for Value Change Events

You create value change events by entering the name of the method in the `valueChangeListener` property of the component. If you are creating a new method, JDeveloper adds the method stub to the Java class. You then add your custom code.

Event and Listener Execution Order

Following is the order in which the events and listeners fire:

1. Validator (managed bean method that performs validation on the component's value)
2. Value change listener (managed bean method to handle value-change events)
3. Action listener (managed bean method to handle action events)
4. Action method
(managed bean method to return a logical outcome String that is used in navigation)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Event and Listener Execution Order

There is a specific order in which the events and listeners are executed. The validators are fired first within the Process Validation phase of the JSF life cycle. The next phase is the Invoke Application phase that handles all the events and listeners.

The value change listeners are executed first, followed by the action listeners. The action method is executed last because that is what determines navigation. Any of these listeners can raise an `AbortProcessingException` to abort the process.

If an action listener raises an exception, it does not prevent navigation. In most cases, however, you would not want to navigate, but rather redisplay the page with an error message. To implement that behavior, in the `action` property, you can call a method that returns `null` if there is an exception, but otherwise returns the `String` outcome to use for navigation.

For example, the `selectWhere()` method referenced in the `Action` property of the Update Order button shown in the slide may contain code to navigate to a manager task flow call if the value of the order is over a certain amount, or to a commit method for an order below that amount. The navigation is dependent on the `String` outcome returned by the `selectWhere()` method.

ADF Faces Enhanced Event Handling

ADF Faces adds to JSF event handling by providing:

- Partial page rendering (PPR)
- JavaScript



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ADF Faces Enhanced Event Handling

Like standard JSF components, ADF Faces command components deliver ActionEvent events when the components are activated, and ADF Faces input and select components deliver ValueChangeEvent events when the component local values change. Though ADF Faces adheres to standard JSF event handling techniques, it also enhances event handling. Unlike standard JSF events, ADF Faces events support AJAX-style partial postbacks to enable partial page rendering (PPR). Instead of full page rendering, ADF Faces events and components can trigger partial page rendering, that is, only portions of a page refresh upon request.

Using JavaScript in ADF Faces Applications

You can use JavaScript:

- To create custom components
- To respond to events
- On individual pages or bundled in a library



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using JavaScript in ADF Faces Applications

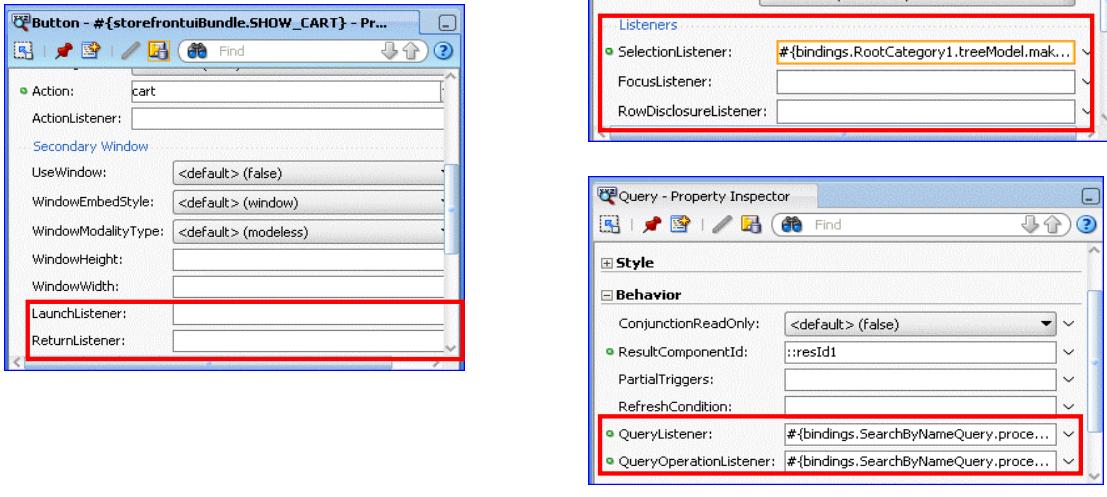
ADF Rich Client uses JavaScript on the client side. The framework itself provides most of the functionality needed. However, sometimes there is a need to write some custom JavaScript code and have your pages access that code. You can use JavaScript in custom components or to respond to events.

To enhance performance, you should bundle all JavaScript code into one JavaScript file and deliver it to the client. The easiest approach is to use the MyFaces Trinidad tag
`<trh:script source=" " />`.

For more information about using JavaScript, see the *Web User Interface Developer's Guide for ADF*.

Other ADF Faces Server Events

The Property Inspector of ADF Faces components displays appropriate event listeners:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Other ADF Faces Server Events

In addition to server-side action and value change events, ADF Faces components also fire other kinds of server events. ADF Faces server events, and the components that generate them, include the following:

- **ActionEvent:** All command components
- **DialogEvent:** dialog
- **DisclosureEvent:** showDetail, showDetailHeader, showDetailItem
- **FocusEvent** (generated when focusing in on a specific subtree, which is not the same as a client-side keyboard focus event): tree, treeTable
- **LaunchEvent:** All command components
- **LaunchPopupEvent:** inputListOfValues, inputComboboxListOfValues
- **PollEvent:** poll
- **QueryEvent:** query, quickQuery, table
- **QueryOperationEvent:** query, queryCriteria, quickQuery
- **RangeChangeEvent:** table
- **RegionNavigationEvent:** region

Other ADF Faces Events (continued)

- ReturnEvent: All command components
- ReturnPopupDataEvent: inputListOfValues, inputComboboxListOfValues
- ReturnPopupEvent: inputListOfValues, inputComboboxListOfValues
- RowDisclosureEvent: tree, treeTable
- SelectionEvent: tree, treeTable, table
- SortEvent: treeTable, table
- ValueChangeEvent: All input and select components (components that implement EditableValueHolder)

All server events have associated event listeners. For example, to process a LaunchEvent event, you would create a LaunchListener event listener implementation. All server event listener implementations must override a processEvent () method, where Event is the event type. For example, LaunchListener accepts an instance of LaunchEvent as the single argument; in your implementation, you must override the event processing method, as shown in the method signature here:

```
public void processLaunch (LaunchEvent evt) {  
    // your code here  
}
```

Instead of creating an event listener implementation, you can create an event listener method in a backing bean. Any event listener method must be a public method that accepts the event as the only parameter and returns void.

As you can see by the examples in the slide, the Property Inspector in JDeveloper exposes the proper listeners for the type of ADF Faces component being used.

Using Table Model Methods in a Selection Listener

You can do the following programmatically:

- Make row current:
`setRowIndex()`
or `setRowKey()`.
- Access data in the current row:
`getRowData()`.
- Obtain number of rows:
`getRowCount()`.
- Determine whether you are at the end of the table: `isRowAvailable()`.

The screenshot shows a Java API reference page for the `CollectionModel` class. The top navigation bar includes links for Overview, Package, Class, Use, Tree, Deprecated, Index, and Help. On the right, there's a sidebar for Oracle Fusion Middleware Java API Reference for Oracle ADF Faces 11g Release 1 (11.1.1) with the identifier E10684-03. The main content area displays the `CollectionModel` class definition under the `oracle.adf.view.rich.model` package. It shows the inheritance chain: `java.lang.Object` → `javax.faces.model.DataModel` → `org.apache.myfaces.trinidad.model.CollectionModel` → `oracle.adf.view.rich.model.CollectionModel`. Below this, it lists all implemented interfaces: `org.apache.myfaces.trinidad.model.RowIndex`.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Table Model Methods in a Selection Listener

The ADF Faces table components use a model to access the data in the underlying list. The specific model class is `oracle.adf.view.rich.model.CollectionModel`. You may also use other model instances, such as `java.util.List`, `java.util.Array`, and `javax.faces.model.DataModel`. If you use one of these other classes, the table component automatically converts the instance into a `CollectionModel`.

You can use table model methods, for example, in a method called by a selection listener on the table. You can programmatically manipulate the table in the following ways:

- To make a row current, call `setRowIndex()` with the appropriate index or with the appropriate `rowKey`.
- To access a particular row in the list, first make that row current, and then call the `getRowData()` method on the table.
- To obtain the total number of rows in the list, call `getRowCount()`; if the model does not yet know the total number of rows that are available, `getRowCount()` returns -1.

The table has an `isRowAvailable()` method that returns `true` if the current row is available. This method is especially useful when the total number of rows is unknown.

You cannot use `af:forEach` to iterate over data within a table component; you can use `af:iterator` instead.

Using Tree Model Methods in a Selection Listener

You can do the following programmatically:

- Determine whether a row has children:
`isContainer()`
- Access children of current row:
`enterContainer()`
- Revert to parent collection:
`exitContainer()`

The screenshot shows a Java API reference page for the `oracle.adf.view.rich.model.TreeModel` class. The page includes navigation links like Overview, Package, Class, Use, Tree, Deprecated, Index, Help, Prev Class, Next Class, and links for Frames and No Frames. It also shows the summary for nested fields, constructors, and methods. The class hierarchy is shown as:

```

java.lang.Object
  └ javax.faces.model.DataModel
    └ org.apache.myfaces.trinidad.model.CollectionModel
      └ org.apache.myfaces.trinidad.model.TreeModel
        └ oracle.adf.view.rich.model.TreeModel
  
```

All Implemented Interfaces:

- `org.apache.myfaces.trinidad.model.RowKeyIndex`

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Tree Model Methods in a Selection Listener

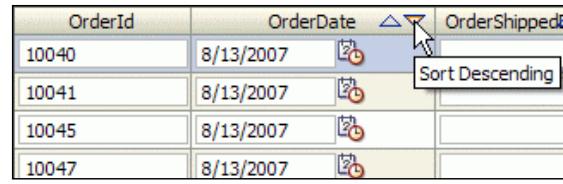
The ADF Faces tree components use a model to access the data in the underlying hierarchy. The specific model class is `oracle.adf.view.rich.model.TreeModel`, which extends `CollectionModel`. You can use code from this class in a managed bean method and call it from a tree's selection listener.

The `TreeModel` is a collection of rows. It has an `isContainer()` method that returns `true` if the current row contains child rows. To access the children of the current row, call the `enterContainer()` method. Calling this method results in the `TreeModel` instance changing to become a collection of the child rows. To revert to the parent collection, call the `exitContainer()` method.

The `oracle.adf.view.faces.model.ChildPropertyTreeModel` class can be useful when constructing a `TreeModel`.

Additional AJAX Events

Additional event types include AJAX events:



A screenshot of an ADF Rich Client table component. The table has three columns: OrderId, OrderDate, and OrderShipped. The OrderDate column contains the date '8/13/2007' for all four rows. The OrderShipped column contains a small icon for each row. The header of the OrderDate column has a blue triangle icon pointing down, indicating it is a sortable column. A mouse cursor is hovering over this triangle icon. A tooltip box labeled 'Sort Descending' is displayed next to the cursor. The table rows are numbered 10040, 10041, 10045, and 10047.

| OrderId | OrderDate | OrderShipped |
|---------|-----------|--------------|
| 10040 | 8/13/2007 | |
| 10041 | 8/13/2007 | |
| 10045 | 8/13/2007 | |
| 10047 | 8/13/2007 | |



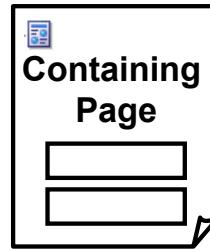
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Additional Event Types

Asynchronous JavaScript and XML (AJAX) combined with the component-based approach of JavaServer Faces (JSF) is the technology behind ADF Faces' highly interactive components. ADF Rich Client components automatically provide various AJAX-like interactions, such as sorting on table components.

Characteristics of the Contextual Event Framework

- Contextual events:
 - Provide a way to coordinate regions
- Can be invoked and consumed only by method action of a data control
- Producer and consumer may be the same or different data controls.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of the Contextual Event Framework

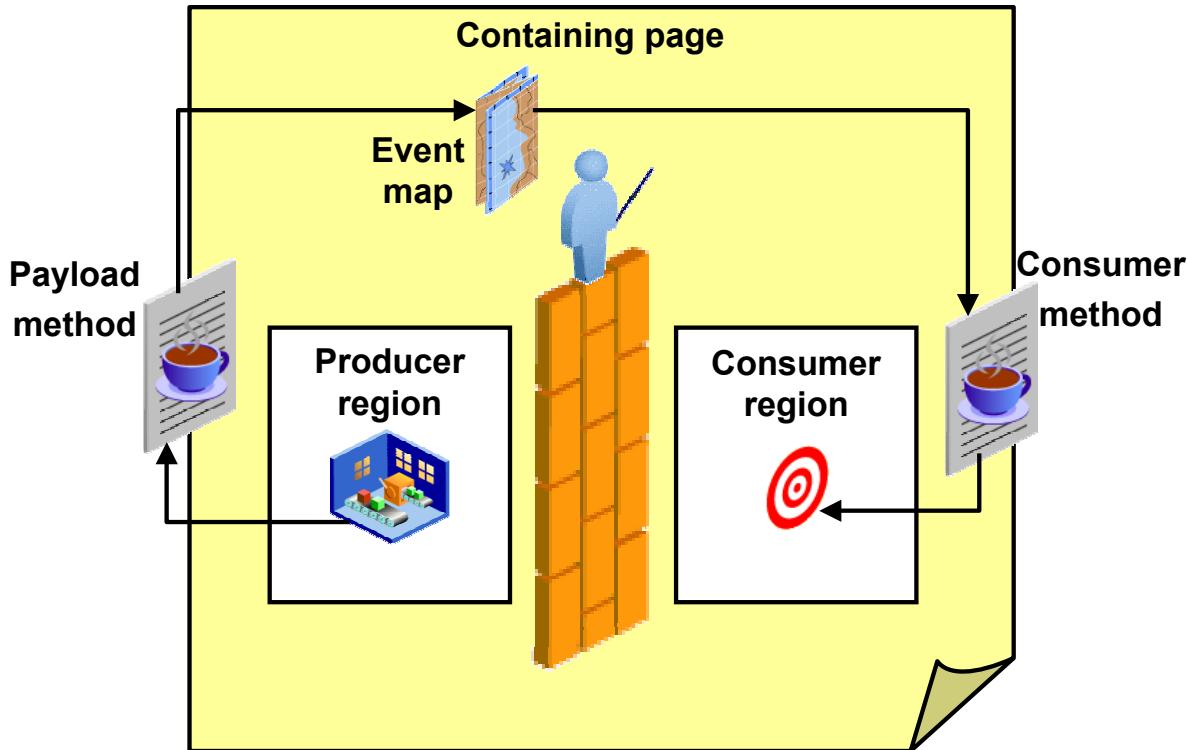
When a page contains multiple regions that need to work together to display relevant information to the user, the need to communicate between regions becomes very important. This communication can be handled through an event framework called the Contextual Event Framework. This framework provides the page with the ability to map events to be produced and consumed by the page's various regions, along with the information to be passed to the consumer regions.

In a typical scenario, a consumer would need to register with the producer to get event notification. However, the Contextual Event Framework in ADF enables the producer and consumer to be tied through an event routing mechanism. The producer publishes the contextual event it is going to raise. The page listens for the event and then routes it to the consumer.

The Contextual Event Framework includes the following characteristics:

- A contextual event can go across data controls.
- A contextual event can be raised or consumed only with a method action binding that must be only data control methods, not managed bean methods.
- Event payload can either be primitive or any serializable object.

Contextual Events: Overview



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Contextual Events: Example

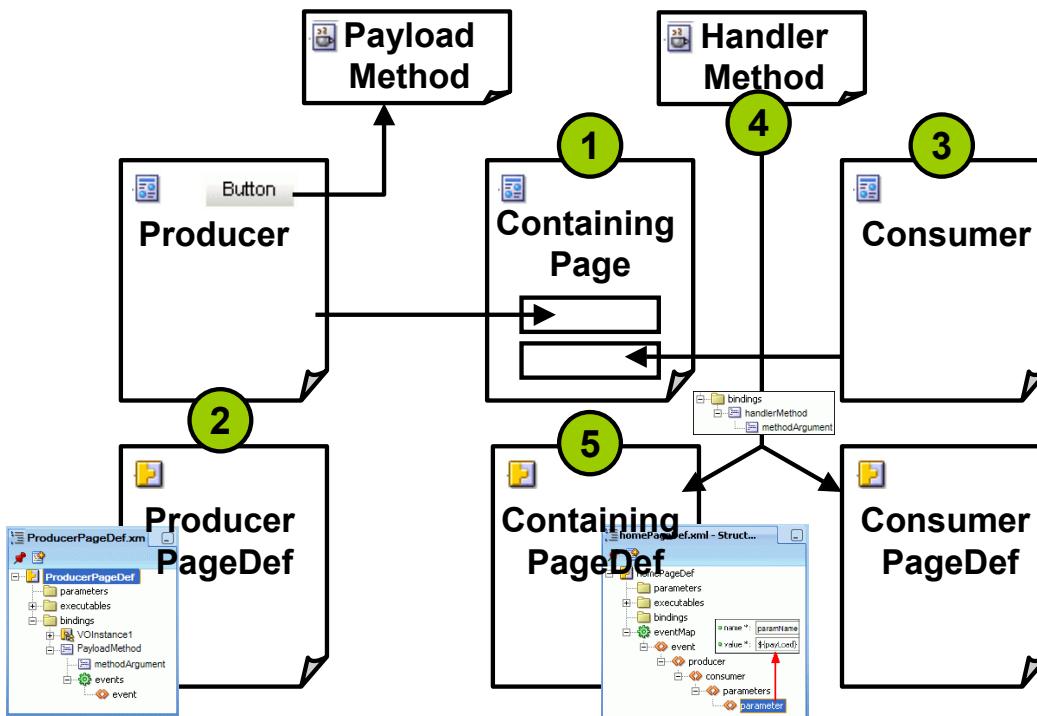
As an example of using the Contextual Event Framework, consider a containing page that contains two regions representing two task flows. The task flows in the separate regions have no knowledge of one another, as if there is a wall between them. Only the containing page has knowledge of both regions, so it is the containing page that must coordinate the interaction between them.

The goal of creating a contextual event in this example is to accomplish the following:

- Pass information from the first region to the second region.
- React in the second region based on the information that is passed.

When an action, such as a button click, occurs in the first region, the event is initiated. There is a method defined that constructs the payload for the event, that is, the information that is being passed to the other region. There is another method on the target, or consumer region, that specifies how that region should respond. The job of the containing page is to map the event: specify which is the payload method, which is the handler method, and so on.

Using the Contextual Event Framework to Coordinate Page Regions



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

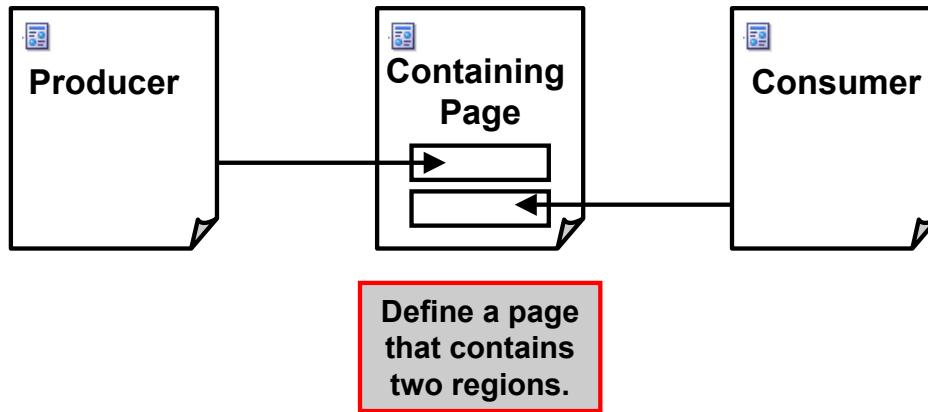
Using Contextual Events

There are several aspects that define a contextual event:

1. A container page that contains the regions to be coordinated
2. A producer that invokes a method to raise the event and provide a payload (information needed by the event)
3. A consumer that is the target of the event
4. A handler method that is bound either to the consumer page fragment or to the page containing the regions
5. An event mapping on the container page definition that specifies:
 - The producer region and payload method
 - The event name
 - The consumer and event handler
 - The parameter (payload) to pass to the event

The steps required to set up a contextual event are described in the next few slides.

Using the Contextual Event Framework to Coordinate Page Regions: Step 1



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Contextual Events: Step 1

Although they do not need to be performed exactly in order, defining a contextual event requires several steps as explained in this and the following slides:

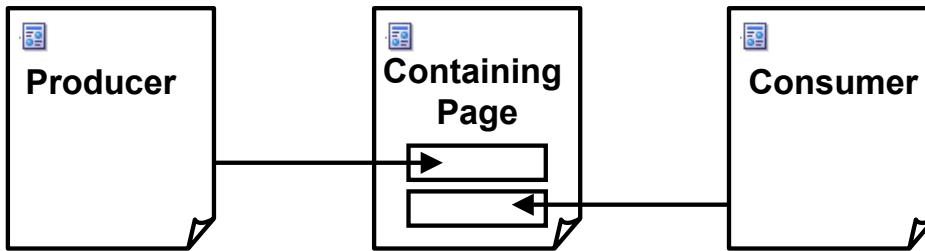
1. Create the container, producer, and consumer:

If you have two regions on a containing page that you want to coordinate by using contextual events, you first need to decide which is the producer and which is the consumer. For example, you may want the data in the consumer region to display differently depending on what is clicked in the producer region, such as displaying products or categories when the user clicks a subcategory or a root category in a category tree. (This is the example that is used in the course application.)

Using the Contextual Event Framework to Coordinate Page Regions: Step 2

Construct an event payload.

Payload Method



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

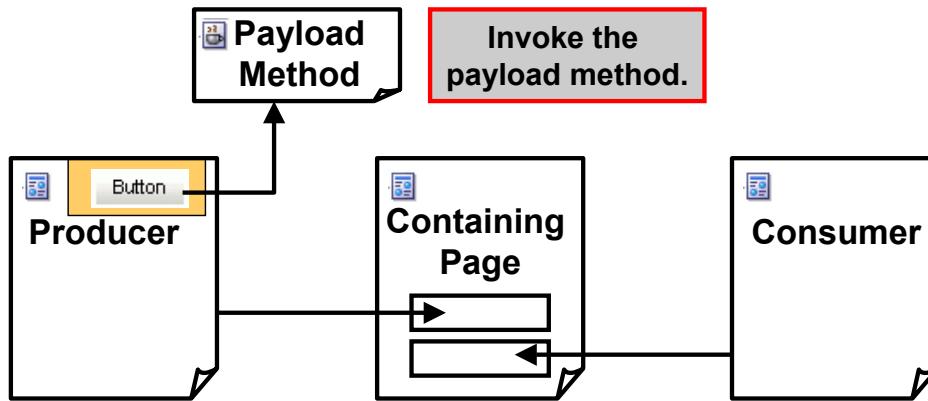
Using Contextual Events: Step 2

2. Create the method that constructs the payload and expose it as a data control:

After you have designated the producer and the consumer, you can determine what data, or payload, the consumer of the event requires to refresh its region appropriately. This can be as simple as passing a foreign key value, but some complex business logic may be required to produce the data needed by the consumer region. In any case, you need to create a method to construct the event payload. This can be in a separate Java class, or it can be a service method in the application module. The method must always return the event payload.

No matter how you create the method, you must expose it as a data control. If it is in a separate Java class, you can simply drag it to the Data Control Palette. If it is an application module method, you must expose it to the client interface.

Using the Contextual Event Framework to Coordinate Page Regions: Step 3



ORACLE

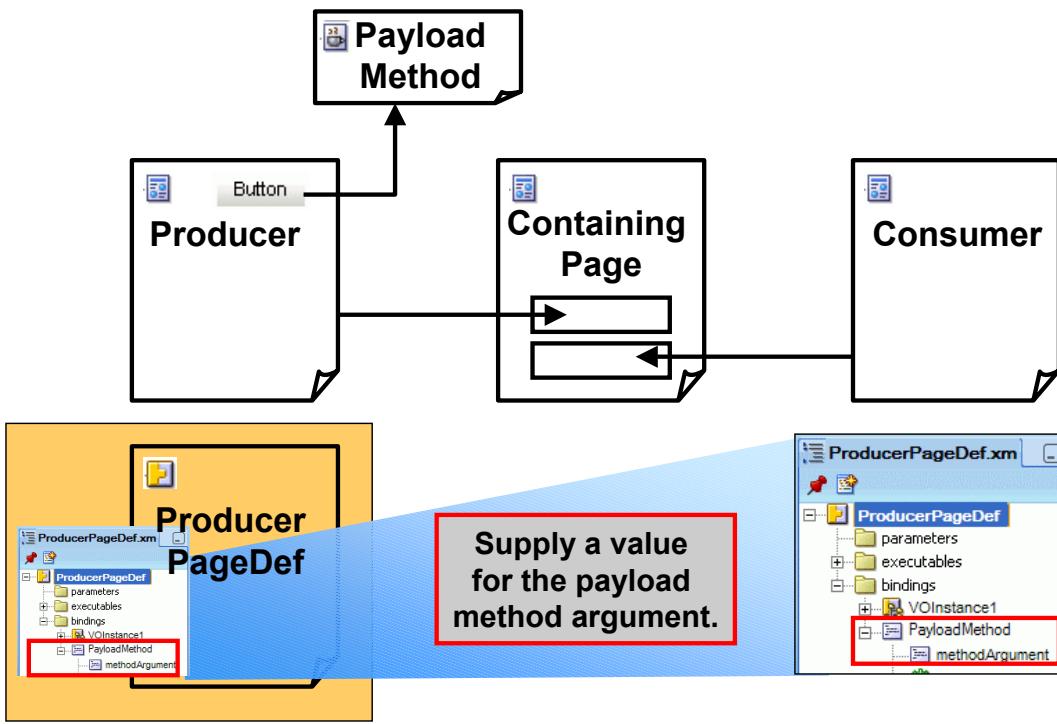
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Contextual Events: Step 3

3. Invoke the payload method:

You need to call the method that you just created, so you can create a command component (button or link) on the producer region's page and drag the method to that component from the Data Control Palette.

Using the Contextual Event Framework to Coordinate Page Regions: Step 4



ORACLE

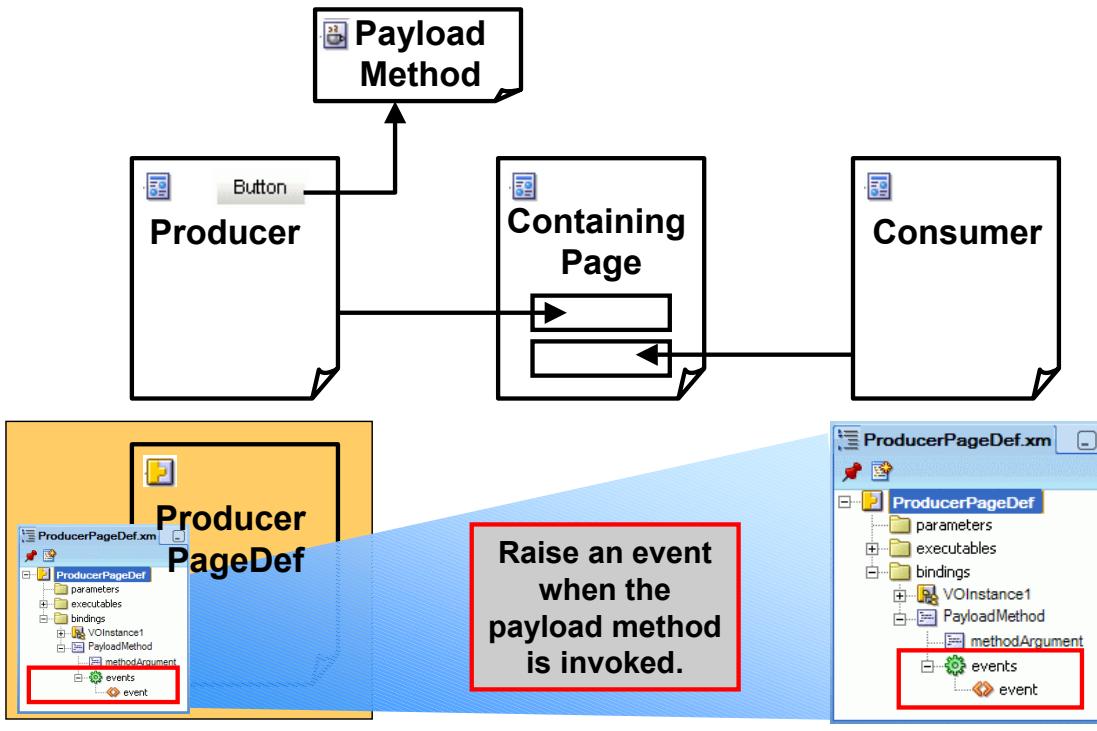
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Contextual Events: Step 4

4. Supply values for the payload method arguments, if any:

The previous step creates a method action binding in the producer page definition. In that page definition file, you may need to supply a value for any arguments that are passed to the method. In the Structure window, select the method argument and set its value in the Property Inspector. You can use EL to pass a value from the data that is on the page.

Using the Contextual Event Framework to Coordinate Page Regions: Step 5



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Using Contextual Events: Step 5

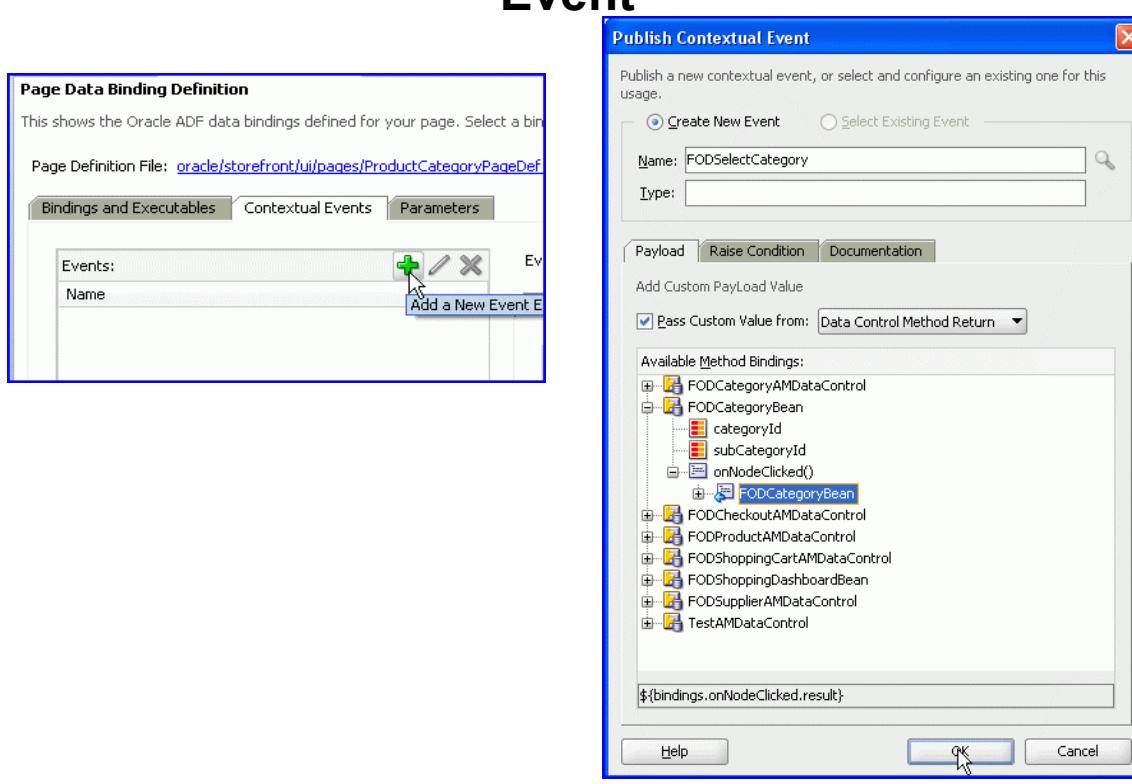
- Raise the event when the payload method is invoked:

You need to specify that an event be raised when the payload method is invoked. In the Structure window for the page definition for the producer page fragment, insert `<events>` inside the method binding, and insert `<event>` inside `<events>`. Specify the event name to be raised as any meaningful name.

Note: One method action can raise one or more events.

You can also use the Contextual Events tab in the page definition file for the producer page to define an event. (This is explained in the next slide.)

Using the Contextual Events Tab to Define an Event



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

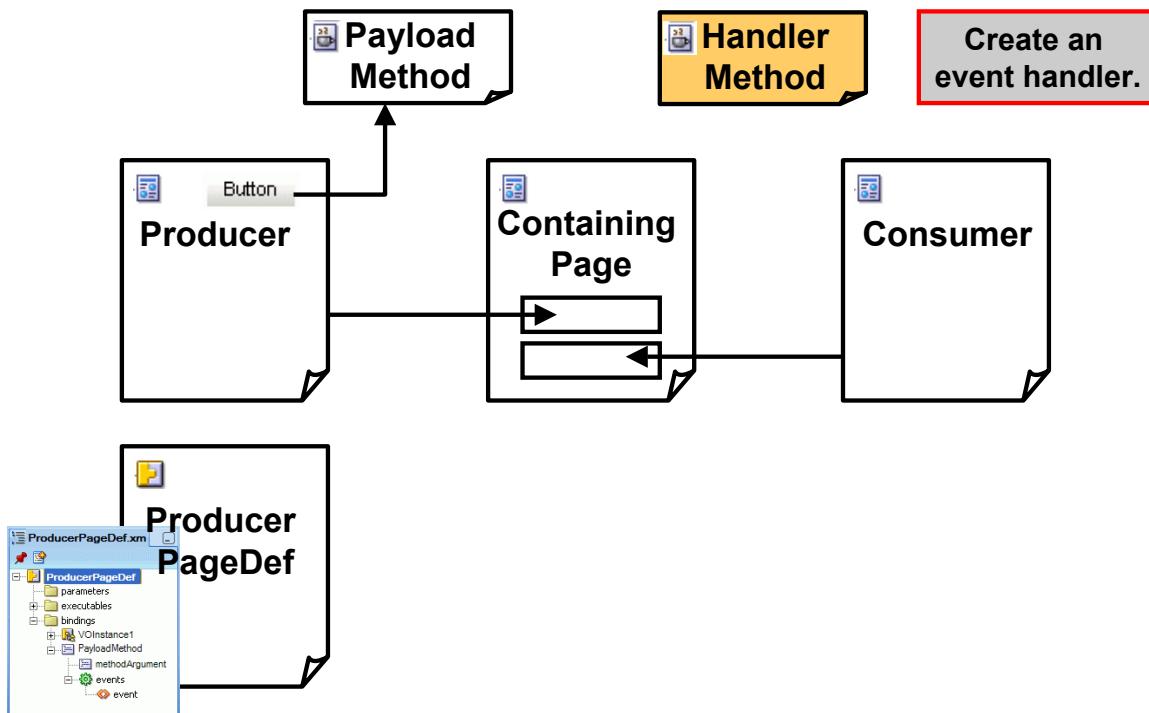
Using Contextual Events: Step 5

You can also use the Contextual Events tab in the page definition file itself to define a contextual event on the producer page.

1. In the overview editor for the producer page definition, click the **Contextual Events** tab.
2. In the Events section, click the **Add** icon.
3. In the Publish Contextual Events dialog box, perform the following steps:
 - a. Select **Create New Event**.
 - b. Enter the name of the event.
 - c. Select **Pass Custom Value from** if you want to pass payLoad data to the consumer.
 - d. If you are passing payload data, select the type of data from the drop-down list.
 - e. You can conditionally raise the event by entering an EL expression on the **Raise Condition** tab.
 - f. Click **OK**.

The event is created on the page, but it is not ready for publishing until it is associated with the component binding.

Using the Contextual Event Framework to Coordinate Page Regions: Step 6



ORACLE

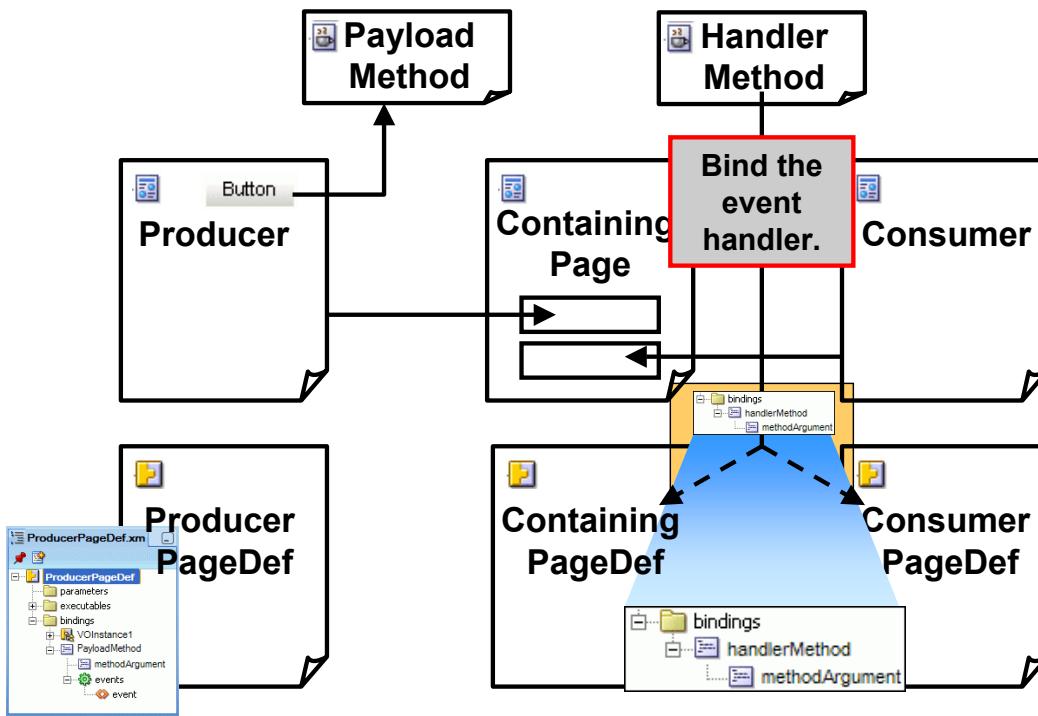
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Contextual Events: Step 6

6. Create an event handler method and expose as a data control:

You also need to create an event handler that specifies what happens in the consumer region after the event is raised. Does it need to requery, using the payload data, or does it simply need to refresh the region? Again, this event handler method, which can be defined in a separate class or as an application module service method, must be exposed as a data control. The handler method expects event parameters or the event payload itself.

Using the Contextual Event Framework to Coordinate Page Regions: Step 7



ORACLE

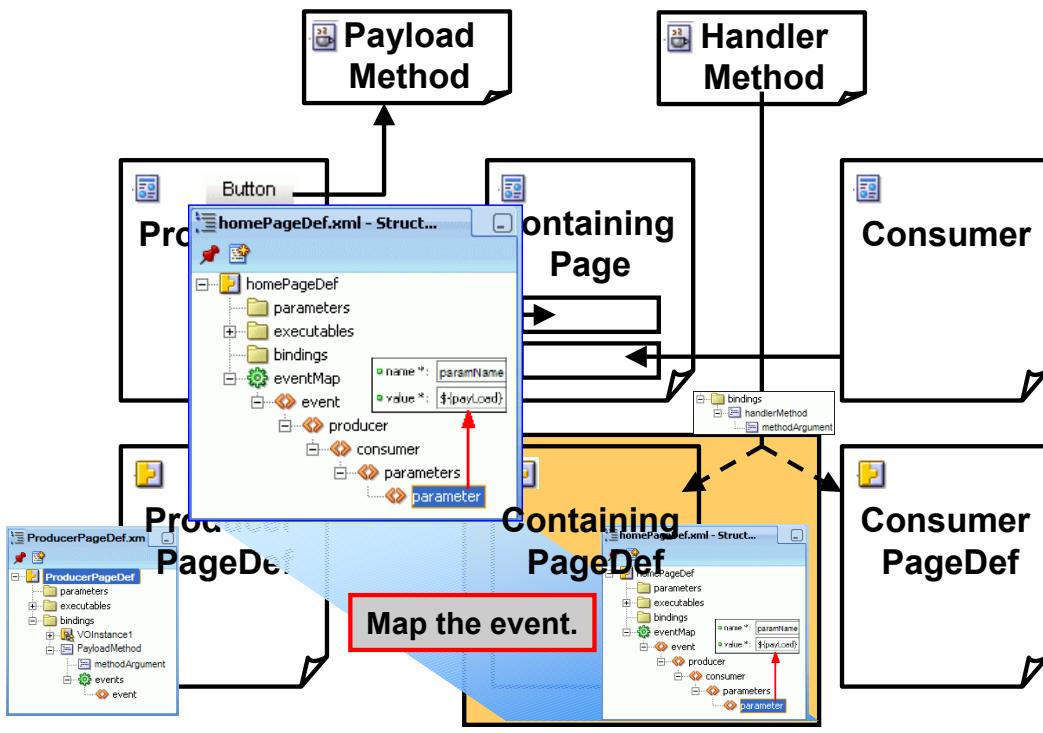
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Contextual Events: Step 7

7. Create a method binding for the event handler method:

Depending on the situation, you need to bind that handler method to either the consumer page fragment or the containing page (the one that contains the regions). You do this by opening the page definition for that page and creating method action as a generic binding. A dialog box enables you to select the data control and operation for the method action.

Using the Contextual Event Framework to Coordinate Page Regions: Step 8



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Contextual Events: Step 8

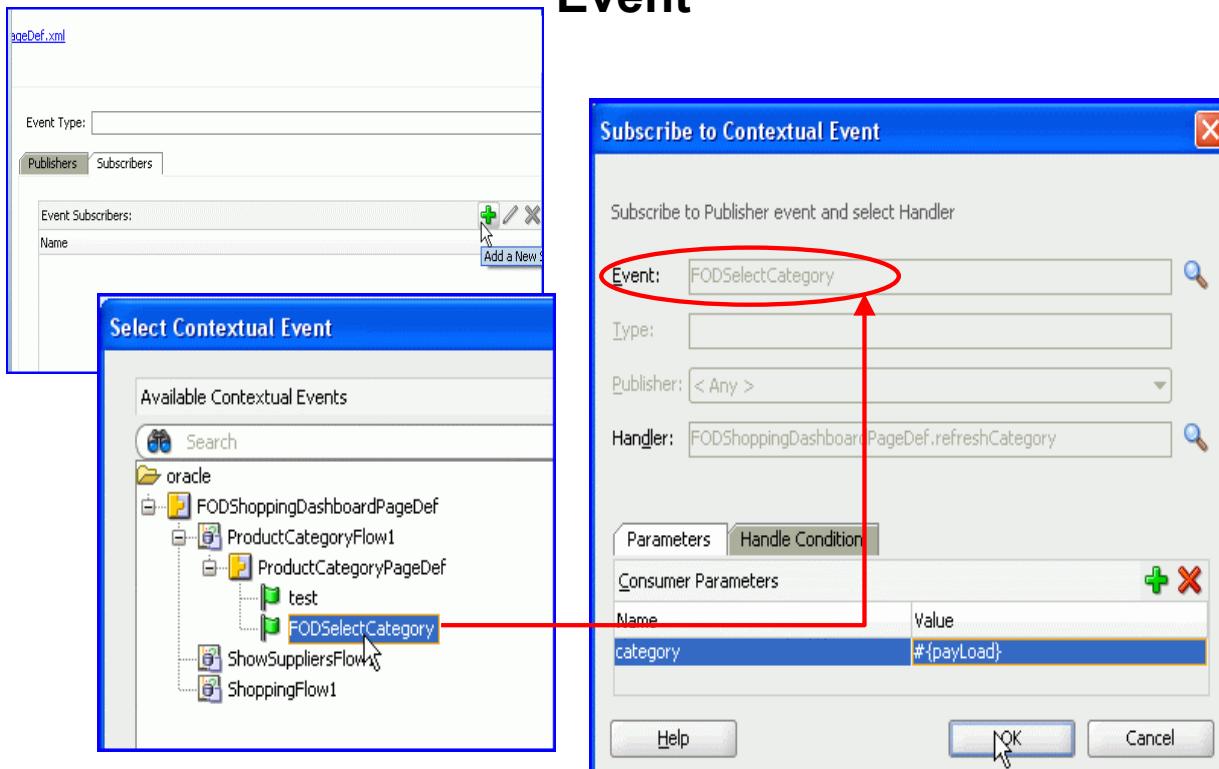
8. Map the event:

Event mapping means that you tell the containing page how to route the event. You do this on the page that contains the regions to be coordinated. In the Structure window for its page definition file for the producer page fragment, right-click the root node and select Edit Event Map. In the upper portion of the Event Map editor, you specify the mapping:

- Select the Producer (the method on the producer page fragment that constructs the payload for the event). This should populate the Event Name for you, because in the page definition file for the producer page fragment, you have already specified the event to be raised when the method is invoked.
- Select the consumer, which is the method to be invoked to handle the event.
- Click the ellipsis button beside Parameters to supply the payload for the event. The Edit Consumer Params dialog box enables you to name the parameter and to use EL to specify the payload, which must be `#{payLoad}`. You can click the ellipsis in the Param Value column to use Expression Builder to choose this value from Payload Data.
- After you have defined the mapping, click Add New to add it to the Event Entries Summary.

You can also use the Contextual Events tab in the page definition file to map an event (explained in the next slide).

Using the Contextual Events Tab to Map the Event



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Using Contextual Events: Step 8

Alternatively, you can use the Contextual Events tab in the page definition file for the parent page to create a map for the contextual event – defining subscribers to, and consumers of, the event.

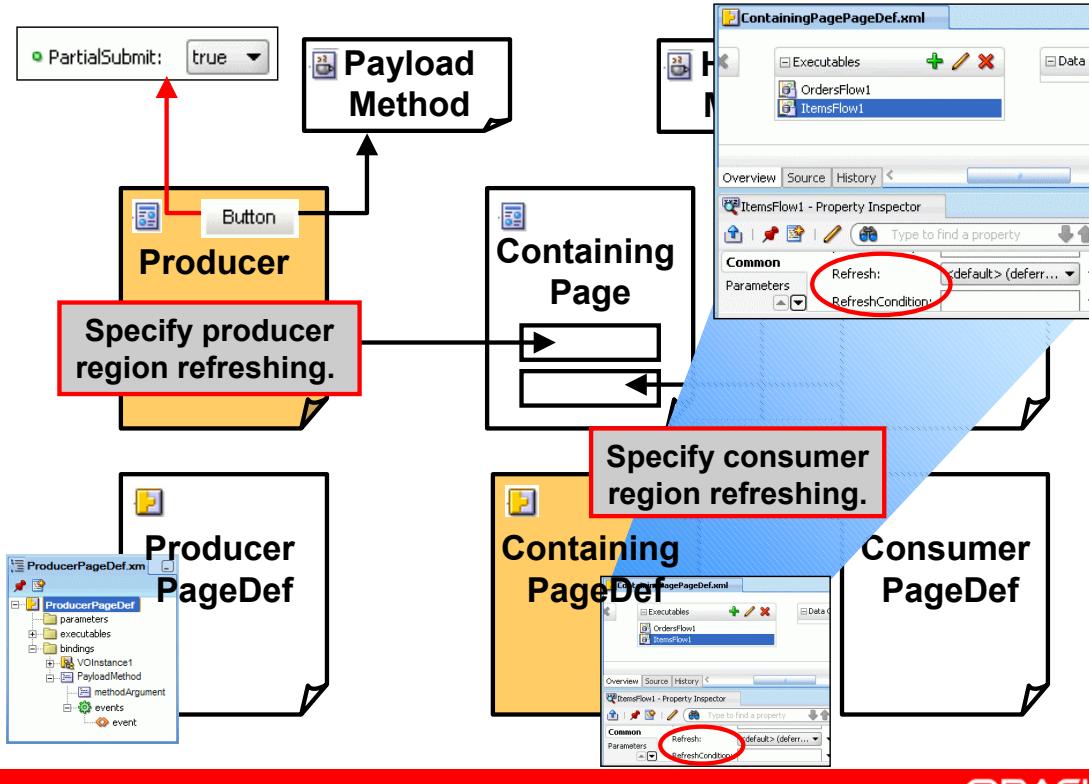
1. In the overview editor of the parent page definition, click the **Contextual Events** tab.
2. On the right of the page, click the **Subscribers** tab, and click the **Add** icon in the Event Subscribers section.
3. In the Subscribe to Contextual Event dialog box, click the **Search** icon.
4. In the Select Contextual Event dialog box, select the event you want to subscribe to from the tree and click **OK**.
5. Click the **Search** icon next to the Handler field.
6. In the Select Handler dialog box, select the event handler from the tree and click **OK**.
7. The Parameters tab enables you to specify consumer parameters for the event. Click **Add**, and enter name-value pair as parameters.
8. If you want to conditionally handle the event, click the **Handle Condition** tab, and enter an EL Expression that determines the conditions under which the handler will process the event.

Using Contextual Events: Step 8 (continued)

9. Click **OK**.
10. In the Publish Contextual Events dialog:
 - a. Select **Create New Event**.
 - b. Enter the name of the event.
 - c. Select **Pass Custom Value from** if you want to pass payLoad data to the consumer.
 - d. If you are passing payload data, select the type of data from the dropdown list.
 - e. You can conditionally raise the event by entering an EL expression in the **Raise Condition** tab.
 - f. Click **OK**.

The event is created on the page, but it is not ready for publishing until it is associated with the component binding.

Using the Contextual Event Framework to Coordinate Page Regions: Step 9



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Contextual Events: Step 9

9. Specify region refreshing:

You also need to define how the regions will be refreshed:

- Producer region: Typically you do not want to refresh the producer region, so for the button that invokes the event, set the `PartialSubmit` property to `true`.
- Consumer region: If you are performing an action in the handler method that refreshes the page (such as requerying), then you do not need to worry about refreshing the consumer page fragment or region. The event consumer processes the payload and updates its model accordingly. Otherwise, you need to set `refresh` and `refreshCondition` appropriately for the consumer region on the page definition for the containing page.

At run time, when the user clicks the button or link in the producer task flow, a PPR action submit in the event producer region invokes a method action in the event producer task flow. When the method action in the event producer task flow is invoked, it raises an ADFm event. The ADFm event is consumed by a method action inside the consumer task flow according to the event mapping defined in the containing page's page definition. The event consumer processes the payload and updates its model accordingly.

Summary

In this lesson, you should have learned how to:

- Configure and use managed beans to contain code to respond to events
- Describe the different types of events
- Use phase listeners
- Create event listeners
- Define action listeners
- Create value change listeners
- Describe ADF Faces enhanced event handling
- List other types of server events used by ADF Faces components
- Explain how ADF components use AJAX events
- Use contextual events to coordinate regions



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 18 Overview: Responding to Events

This practice covers the following topics:

- Defining Task Flow Parameters
- Creating a Helper Method to Evaluate EL
- Creating the Producer (Payload) Method
- Initiating the Contextual Event
- Creating the Consumer (Handler) Method
- Mapping the Contextual Event
- Passing Values Through Region Parameters
- Modifying the JSF Life Cycle



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 18 Overview: Responding to Events

In this practice, you create task flow parameters that enable a page to accept values from outside the task flow. You define a contextual event to coordinate two regions, and you optionally write phase listeners to display JSF life-cycle phases when a user navigates to a new page.

19

Implementing Transactional Capabilities

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Explain ADF BC transaction handling
- Implement task flow transaction control
- Handle transaction exceptions
- Define response to the browser's Back button
- Enable the save for later functionality



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

This lesson describes how to build pages to support a multipage transaction. You add transaction capability to the checkout flow and also define pages to insert, update, and delete suppliers.

Handling Transactions with ADF BC

- Application modules handle transaction and concurrency support.
- No coding is required unless you want to customize the default behavior.
- For nested application modules, the outermost application module provides the transaction context for the others.



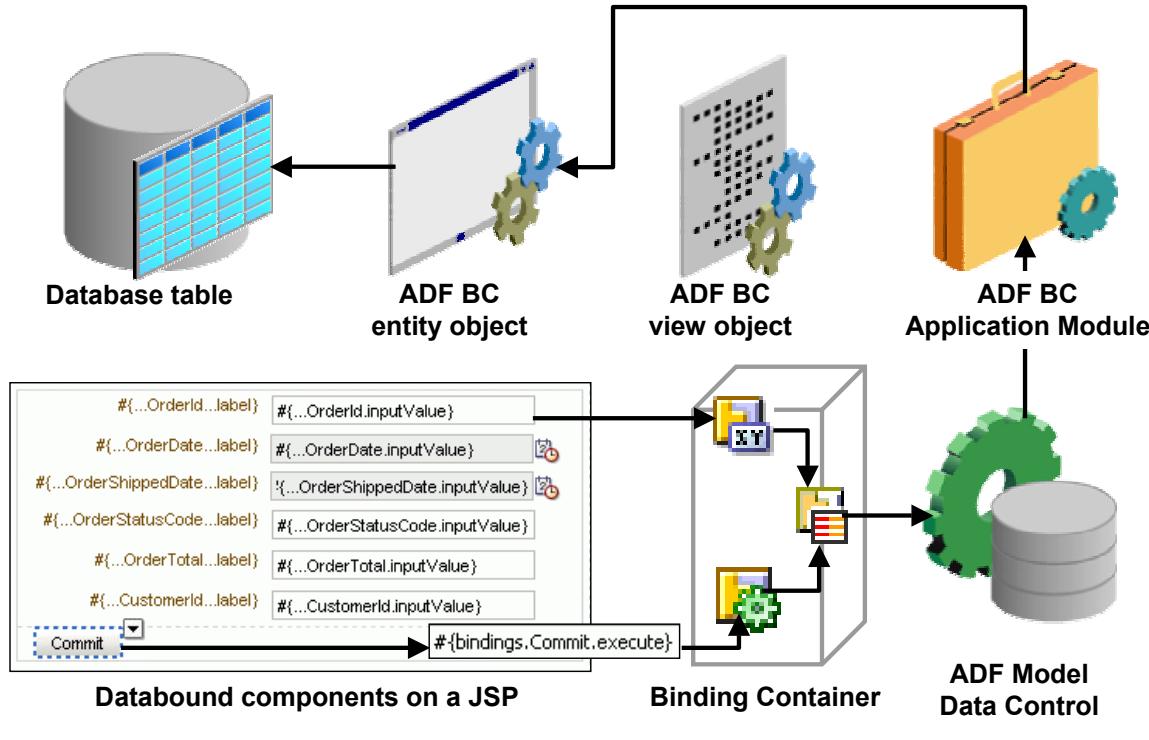
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Business Components Transactions

A transaction is a persisted collection of work that can be committed or rolled back together as a group.

The Business Components framework handles all transactions at the application module level. The application module provides a transaction context, with a single database connection. When you commit a change, the commit applies to all view objects in the application module. For nested application modules, the outermost or root application module provides the transaction context for all nested application modules.

Default ADF Model Transactions



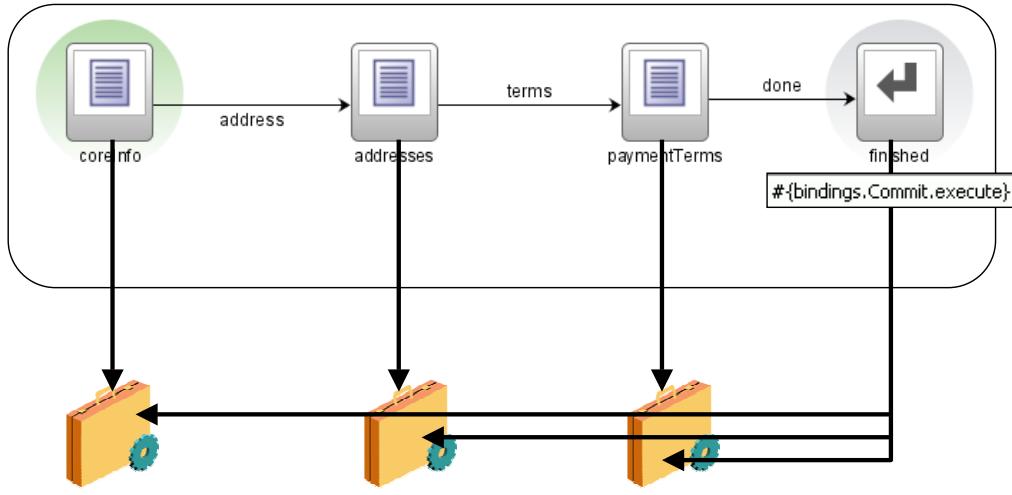
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Default ADF Model Transactions

The default behavior when an ADF model change is committed is to issue the commit directly to the Transaction object of the current associated ADF BC application module instance to handle the transaction.

Transactions in Task Flows



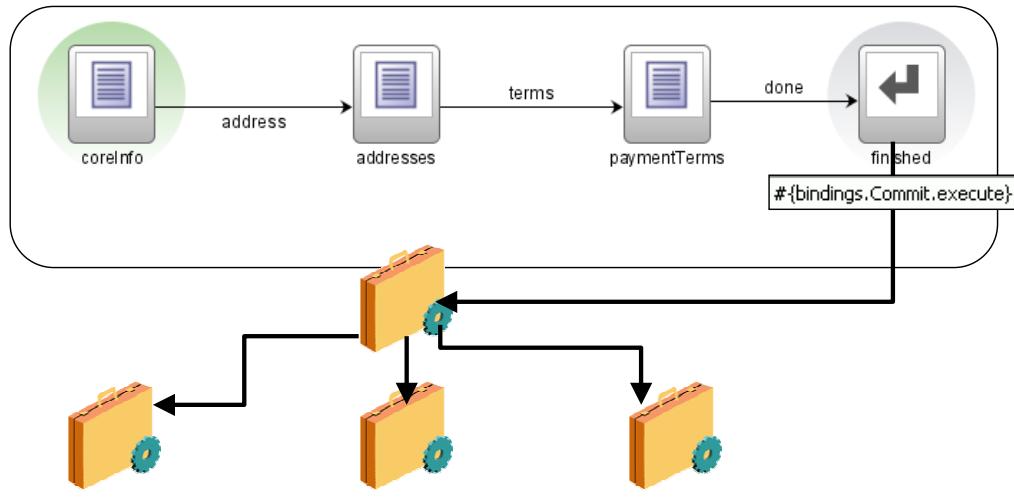
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Transactions in Task Flows

When you issue a commit within a bounded task flow, the entire task flow is committed as a unit. If changes have been made on multiple pages within the task flow, they are all committed. If you have multiple pages that are based on different application modules, allowing the application modules to control the transactions means that the changes on each page get committed separately. A commit failure on one of the application modules does not prevent the other changes from being committed, because the Transaction object of each application module is separate from the others.

Controlling Transactions in Task Flows



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

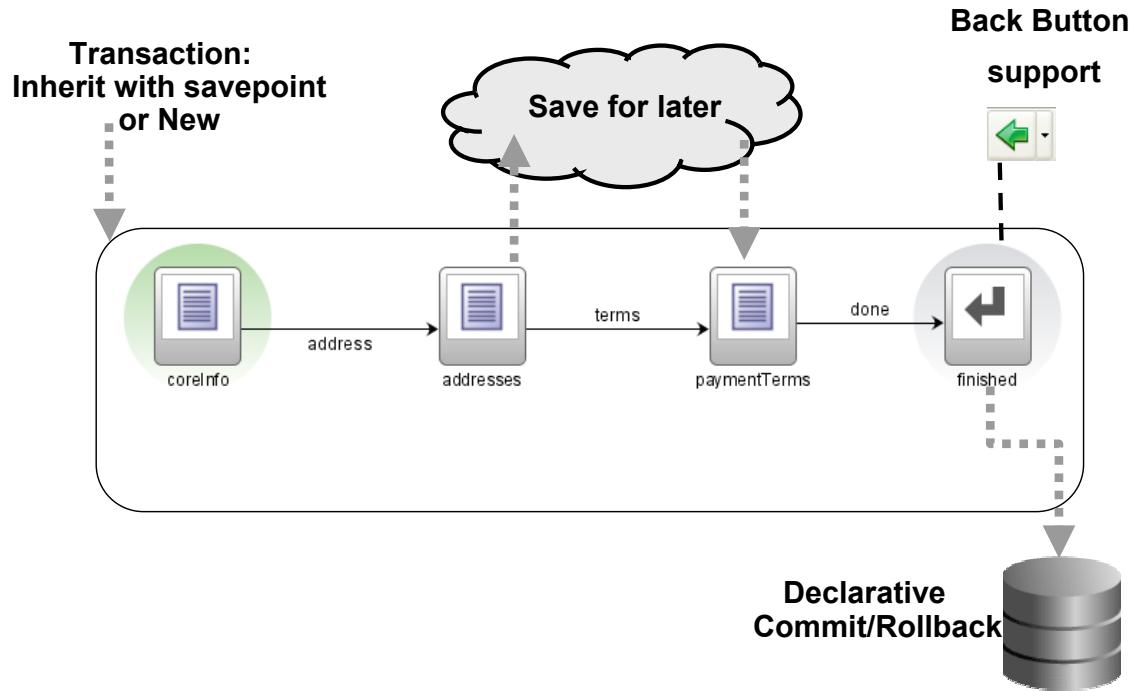
ORACLE

Controlling Transactions in Task Flows

You can use an ADF bounded task flow to represent a transactional unit of work and to declaratively manage transaction boundaries.

When you allow the bounded task flow to control the transaction, any application modules involved are treated as nested application modules. The task flow interacts with the `Transaction` object of a “dummy” application module, which provides the transaction context for all the nested AMs. If the commit on one of the nested AMs should fail, the entire transaction fails.

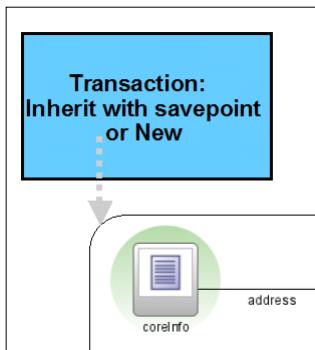
Transaction Support Features of Bounded Task Flows



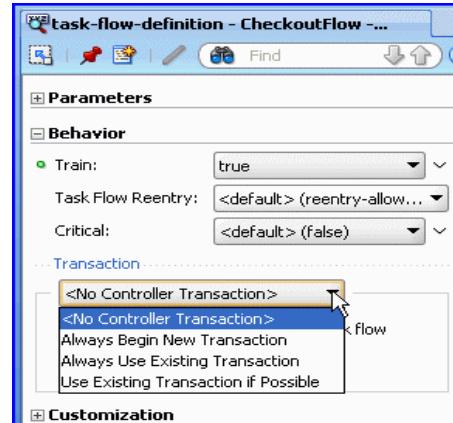
Transaction Support in Bounded Task Flows

- Transaction options on the called task flow definition specify whether a called ADF bounded task flow should join an existing transaction, create a new one, or create a new one only if there is no existing transaction.
- Savepoints: ADF creates a savepoint when a task flow is entered with an existing transaction. The savepoint is a snapshot of the model state at the time the task flow was entered. The savepoint enables the model state to be restored to what it was at the time the task flow was entered.
- If the called ADF bounded task flow is able to start a new transaction (based on the transaction option that you select), you can specify whether the transaction is committed or rolled back when the task flow returns to its caller.
- In a called task flow definition, you can specify two different return task flow activities that result in either committing or rolling back a transaction in the called ADF bounded task flow.
- You can enable users to save the state of the UI and model at any point and come back to it later.
- You can specify how a page should behave when users navigate to it by using the Back button.

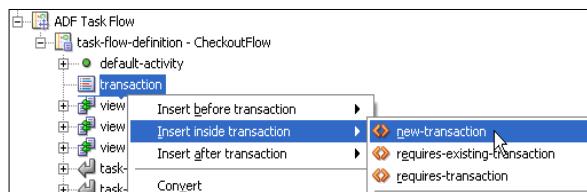
Specifying Task Flow Transaction Start Options



Defining task flow transaction options:



In Structure window



In Property Inspector

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

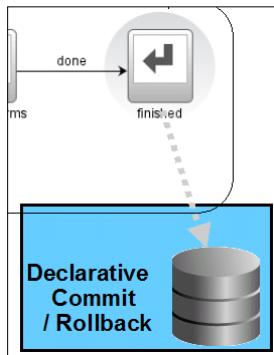
Specifying Task Flow Transaction Start Options

Transaction start options on the called task flow definition specify whether a called ADF bounded task flow should:

- Always use an existing transaction (`requires-existing-transaction`)
- Always begin a new transaction (`new-transaction`)
- Use an existing transaction if possible (`requires-transaction`)

If no transaction option is specified, a transaction is not started on entry of the called ADF bounded task flow. A run-time exception is thrown if the ADF bounded task flow attempts to access transactional services.

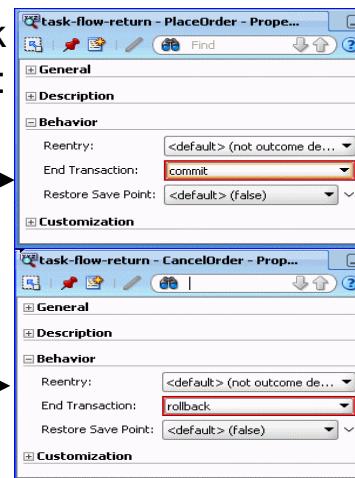
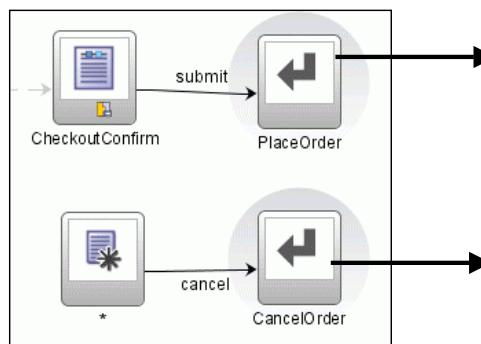
Specifying Task Flow Return Options



Commit or rollback must be specified for some transactions:

| Transaction Option | Commit or Rollback |
|-------------------------------|--|
| New Transaction | Commit or rollback must be specified |
| Requires Existing Transaction | Commit and rollback can not be specified |
| Requires Transaction | Commit or rollback must be specified |
| None | Commit and rollback can not be specified |

You specify commit or rollback on transaction return activities:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Specifying Task Flow Return Options

If the called ADF bounded task flow is able to start a new transaction (based on the transaction option that you selected), you can specify whether the transaction should be committed or rolled back when the task flow returns its caller. The table at the top of the slide shows the requirements to specify commit or rollback for each transaction option.

The commit and rollback options are set on the task flow return activity that returns control back to the calling task flow. You use the Property Inspector in the End Transaction property on the Behavior tab. The same task flow that starts a transaction must also resolve the transaction.

In a called task flow definition, you can specify two different return task flow activities that result in either committing or rolling back a transaction in the called ADF bounded task flow. Each of the task flow return activities passes control back to the same calling task flow. The difference is that one task flow return activity specifies the `commit` option, whereas the other specifies the `rollback` option.

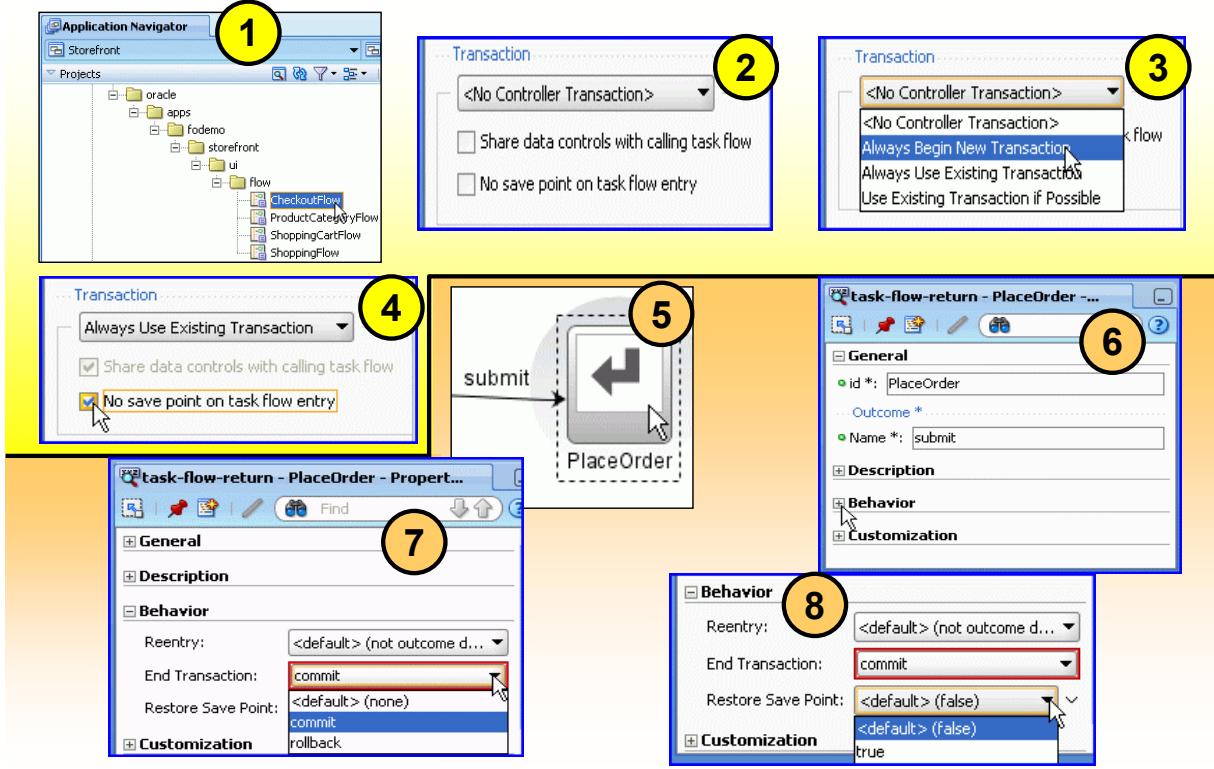
In the example in the slide, if transaction processing completes successfully, the `submit` control flow rule passes to the `PlaceOrder` task flow return activity, which specifies the options to commit the transaction. If the transaction is cancelled before completion, the `cancel` control flow rule passes control to the `CancelOrder` task flow activity, which specifies the options to roll back the transaction.

Specifying Task Flow Return Options (continued)

Only the task flows that begin a transaction can commit or roll back the transaction. Also, a task flow that always begins a transaction (the `new-transaction` option) must specify the resolution of the transaction (either commit or rollback) on all of its task flow return activities' outcomes. It is invalid for the task flow to return without resolving a transaction it started.

If you want changes made within the called ADF bounded task flow to be discarded when it is exited, you can specify the `restore-save-point` option on the task flow return activity. If `restore-save-point` is true when the task flow is exited, the ADF controller rolls back to the previous ADF Model savepoint that was created when the ADF bounded task flow was entered. The `restore-save-point` option applies only to cases when an ADF bounded task flow is entered by joining an existing transaction (either the `requires-existing-transaction` or `requires-transaction` option is also specified) and a savepoint is created upon entry.

Enabling Transactions on a Task Flow



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enabling Transactions on a Task Flow

Before you begin, you should have two task flows; one calls the second ADF bounded task flow using a task flow call activity.

To enable an ADF bounded task flow to run as a transaction, in the called ADF bounded task flow, you specify a transaction option that defines whether or not a new transaction will be created when it is called. You also define a task flow return activity that specifies an outcome upon return to the caller. To accomplish this, perform the following steps (1–4 specify transaction start options, whereas 5–8 are transaction return options):

Transaction Start Options

1. In the Application Navigator, double-click the called ADF task flow to open it in the editor.
2. On the Overview tab for the called ADF bounded task flow, click Behavior and scroll down to the Transaction section.
3. Select one of the following from the Transaction drop-down list: (If you do not select any item, the called ADF bounded task flow does not participate in any transaction management.)
 - **Always Use Existing Transaction:** When called, the ADF bounded task flow participates in an existing transaction that is already in progress.

Enabling Transactions on a Task Flow (continued)

- **Use Existing Transaction if Possible:** When called, the ADF bounded task flow either participates in an existing transaction if one exists, or starts a new transaction upon entry of the ADF bounded task flow if one does not exist.
 - **Always Begin New Transaction:** A new transaction is always started when the ADF bounded task flow is entered, regardless of whether or not a transaction is in progress. The new transaction is completed when the ADF bounded task flow exits.
4. If you select Always Use Existing Transaction or “Use Existing Transaction if Possible,” you can optionally select the “No save point on task flow entry” check box. If you select the check box, an ADF Model savepoint is not created on task flow entry. An ADF Model savepoint is a saved snapshot of the ADF Model state. Selecting the check box means that the overhead associated with a savepoint is not created for the transaction.

Transaction Return Options

5. In the editor, select the task flow return activity in the called ADF bounded task flow.
6. In the Property Inspector, click Behavior.
7. If the called task flow definition supports creation of a new transaction (task flow definition specifies the “Use Existing Transaction if Possible” or Always Begin New Transaction option), select one of the following in the End Transaction drop-down list:
 - `commit`: Commits the existing transaction to the database
 - `rollback`: Rolls back a new transaction to its initial state on task flow entry. This has the same effect as canceling the transaction.
8. In the Restore Save Point drop-down list, select `true` if you want changes that the user made within the called ADF bounded task flow to be discarded when it is exited. The savepoint that was created upon task flow entry will be restored.

Sharing Data Controls

Data control scopes:

- Shared (default): Called flow shares data control instances with calling flow
- Isolated (check box deselected): Called flow has unique instance of data controls

```
<data-control-scope>
<shared/>
</data-control-scope>
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Sharing Data Controls

The ADF Model layer provides a scoping mechanism for data controls called a `DataControlFrame`. At any given time there is one current `DataControlFrame`. Whenever a data control is referenced, for example, by the EL expression `#{bindings.foo}`, the ADF Model layer goes to the current `DataControlFrame` to find the data control.

By default, the called task flow shares all data control instances with its caller (shared data control scope). You can change that by deselecting the “Share data controls with calling task flow” check box. With this setting, a `DataControlFrame` is created for each task flow that is entered, with the result that each task flow has its own unique instance of any data controls it uses—it is isolated from its caller and does not share any data control instances.

The model layer uses the `DataControlFrame` to manage the transaction that the data controls within the frame participate in. The behavior of various task flow transactional settings is dependent on the data control scope, as shown in the following table:

Sharing Data Controls (continued)

Interaction of Transaction and Data-Controls-Scope Option Settings

| Setting | Shared | Isolated |
|-------------------------------|--|--|
| Requires Transaction | Begins a new transaction if one is not already open | Always begins a new transaction |
| Requires New Transaction | Begins a new transaction if one is not open and throws an exception if one is already open | Always begins a new transaction |
| Requires Existing Transaction | Throws an exception if a transaction is not already open | Invalid: This option cannot be selected at design time. |
| None | N/A | A new DataControlFrame is created without an open transaction. |

Synchronizing with the Model

By default, a transaction is not committed until the task flow is exited, because that is when the DataControlFrame is committed. If you attempt to commit in the middle of the task flow, the state of the UI gets out of synchronization with the state of the model. However, you can enable a midtask flow commit by manually committing the DataControlFrame, as in the following example:

```
static public void saveAndContinue() {
    Map sessionMap =
        FacesContext.getCurrentInstance().getExternalContext().getSessionMap();
    BindingContext context =
        (BindingContext) sessionMap.get(BindingContext.CONTEXT_ID);
    String currentFrameName = context.getCurrentDataControlFrame();
    DataControlFrame dcFrame = context.findDataControlFrame(currentFrameName);
    dcFrame.commit();
    dcFrame.beginTransaction(null);}
```

FacesContext contains the information necessary for request processing, and invokes an object that executes the life cycle. BindingContext is a container object that defines a hierarchy of data controls and data binding objects derived from the Oracle ADF Model Layer. It is the handle through which the client accesses the data binding layer, and is represented by the bindings variable in EL data binding expressions.

Handling Transaction Exceptions

- You should designate an exception handling activity on transactional task flows.
- Can be any activity type, such as:
 - View: To display a message
 - Router: To call a method depending on the type of exception

The red bar spans most of the width of the slide, centered horizontally.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

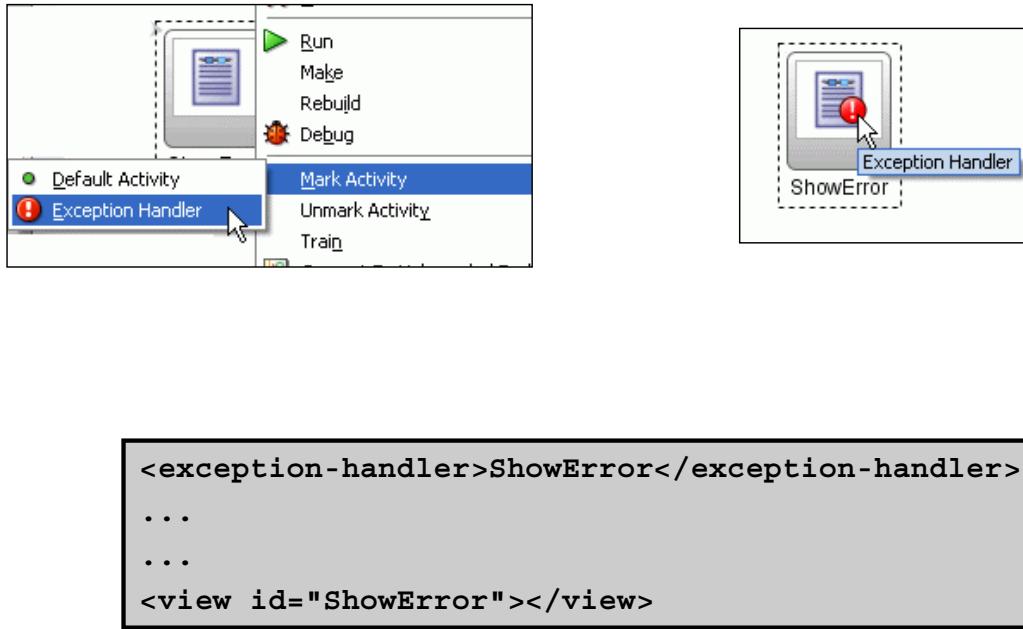
Handling Transaction Exceptions

During the execution of an ADF task flow, exceptions can occur that may require some kind of exception handling, such as when a method call activity throws an exception or when a user is not authorized to execute the activity.

To handle exceptions thrown from an activity or caused by some other type of ADF controller error, you can create an exception handler for an ADF bounded or unbounded task flow. When an exception is raised within the task flow, the control flow passes to the designated exception handling activity. The exception handler activity can be any supported activity type, for example, a view that displays an error message, or a router activity that passes the control flow to a method, based on an expression that evaluates the type of exception.

As a best practice, any ADF bounded task flow representing a managed transaction should designate an exception handling activity. When running an ADF bounded task flow managed transaction, the ADF controller attempts to commit the transaction when it reaches a task flow return activity identified in metadata for a commit. If the commit throws an exception, the control is passed to the ADF bounded task flow's exception handling activity. This provides the end user a chance to correct any data and then reattempt to commit it. You can use the exception handling activity to display a warning message on a page that tells the user to correct the data and attempt to commit it again.

Designating an Exception Handler Activity



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

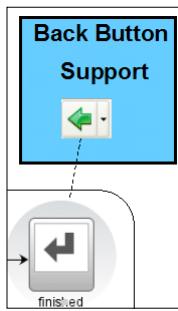
Designating an Exception Handler Activity

To designate an activity as an exception handler for a task flow, perform the following steps:

1. Right-click the activity in the task flow diagram, and then select Mark Activity > Exception Handler. A red exclamation mark is superimposed on the activity in the task flow to indicate that it is an exception handler.
2. To unmark the activity, right-click the activity in the task flow diagram, and then select Unmark Activity > Exception Handler. If you mark an activity as an exception handler in a task flow that already has a designated exception handler, the old handler is unmarked.

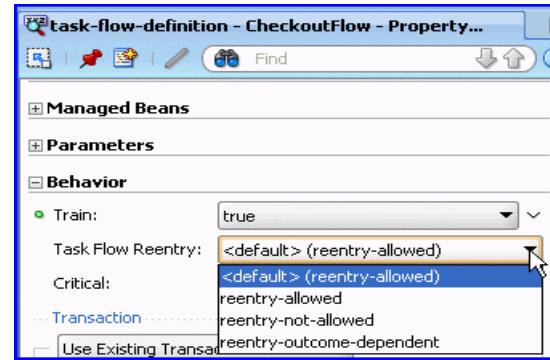
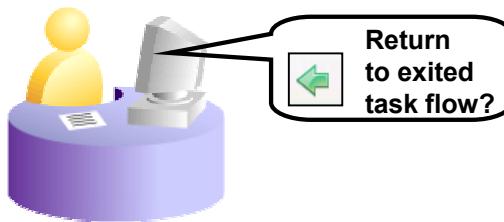
After you designate that an activity is the exception handling activity for a task flow, the task flow metadata updates with an `<exception-handler>` element that specifies the ID of the activity, as shown in the slide.

Defining Response to the Back Button



The Task Flow Reentry property determines whether the user can return to an exited task flow by clicking the browser's Back button:

- reentry-allowed: OK
- reentry-not-allowed: Throws exception
- reentry-outcome-dependent: Depends on the outcome from exited task flow



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Response to the Back Button

To deal with cases in which the end user clicks the Back button to navigate back into an ADF bounded task flow that was already exited, you can specify the following `task-flow-reentry` options for the task flow definition that specify whether a page in the ADF bounded task flow can be reentered:

- **reentry-allowed:** Reentry is allowed on any view activity within the ADF bounded task flow.
- **reentry-not-allowed:** Reentry of the ADF bounded task flow is not allowed. An end user can still click the browser's Back button and return to a page within the bounded task flow. However, if the user does anything on the page such as clicking a button, an exception (for example, `InvalidTaskFlowReentry`) is thrown indicating that the bounded task flow was reentered improperly. The actual reentry condition is identified upon the submit of the reentered page.
- **reentry-outcome-dependent:** Reentry of an ADF bounded task flow using the browser's Back button is dependent on the outcome that was received when the same ADF bounded task flow was previously exited via task flow return activities. For example, a task flow representing a shopping cart can be reentered if the user exited by canceling an order, but not if the user exited by completing the order. Upon reentry, ADF bounded task flow input parameters are evaluated using the current state of the application, not the application state existing at the time of the original ADF bounded task flow entry.

Defining Response to the Back Button (continued)

If the Task Flow Reentry property of the task flow is set to `reentry-outcome-dependent`, a property of the task flow return activity determines whether the user can use the Back button to navigate back to the called task flow. On the Behavior tab of the task flow return's Property Inspector, you can set the Reentry property to `reentry-allowed` or `reentry-not-allowed`. This enables you to specify a different response to the Back button depending on the outcome that is returned to the calling task flow.

When an end user reenters an ADF bounded task flow by using the browser's Back button and reentry is allowed, the value of a managed bean on the reentered task flow is set back to its original value—the same value that it was before the end user left the original task flow.

This results in the managed bean value resetting back to its original value before a view activity in the reentered task flow is rendered. Any changes that occurred before reentry are lost. To change this behavior, specify the `<redirect>` element on the view activity in the reentered bounded task flow. When the end user reenters the bounded task flow using the Back button, the managed bean has the new value from the parent task flow, not the original value from the child task flow that is reentered.

Saving for Later

Quitting time

I'll work some more on this tomorrow.

Next morning

Great—right where I left off!

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Saving for Later

In some applications, an end user may need to stop without completing a task, for example, when additional information is needed to answer questions. As the developer of the application, you may want to retain the current state of the task if a user leaves a page without finalizing it. You can also enable the user to complete the rest of the task at a future point, with former unsaved values restored to the page. The ability to save the current application state is called save for later.

There are two general categories of saving for later:

- **Explicit:** For example, a page contains a button that the user can click to save all the data entered so far on the page. Explicit save for later is available for both ADF unbounded and bounded task flows. For example, a user may click a button to save, but not submit, a partial expense report, and then may cancel out of the application. Later the user can invoke a list of saved expense reports and select the one to continue working on.
- **Implicit:** For example, a user accidentally closes a browser window without saving the data, or the session times out. Implicit save for later can originate only from a bounded task flow.

Save for later saves the data without enforcing validation rules or submitting it. The end user can resume working on the application later with the same data that was saved at the point when save for later was performed.

Saving for Later (continued)

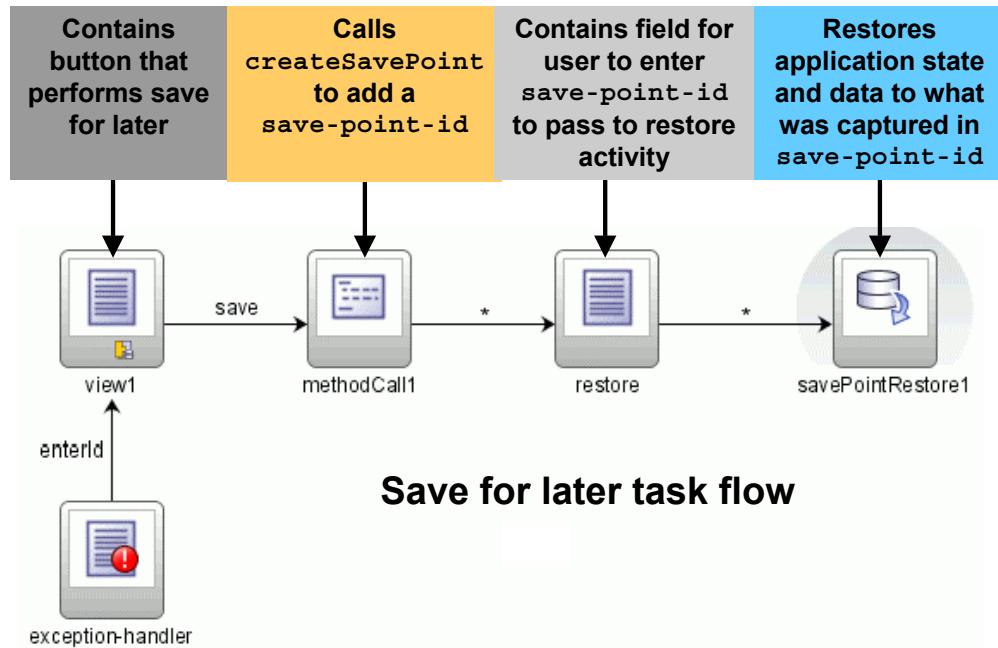
Save for later is implemented by using a `save-point-id` that captures a snapshot of the ADF application at a specific point in time. This includes everything from the time the savepoint is created in the originating task flow up the call stack. Application state information is also saved, including:

- UI state, including table sort order and selected tabs, check boxes, table rows, and so on
- Managed beans: Saves the state information that is saved in memory scopes, except for request scope and application scope
- Navigation state: Task flow call stack, which tracks where the user is in the application and the navigational path for getting there, and also the starting point of any data transactions
- Model state: Any data model updates made since beginning the current bounded task flow, subject to limits imposed by the model layer on the saved state lifetime

The same `save-point-id` can be used when the same user repeatedly performs a save for later on the same instance of a task flow within the same browser window within the same session. If a user performs a save for later following every page of a task flow, only one `save-point-id` is created and is overlaid each time.

An example of when you might use an explicit save for later is an expense report application. A user may click a button to save, but not submit, the partial expense report, and then may cancel out of the application. Later the user can invoke a list of saved expense reports and select the one to continue working on.

Enabling Explicit Save for Later



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enabling Explicit Save for Later

To enable explicit save for later, you must incorporate a call to the `createSavePoint()` API to create a savepoint. The `save-point-id` saves data and state information about a region, view port, or portlet. Later, you use the Save Point Restore activity to restore application state and data associated with a `save-point-id`.

A `save-point-id` is stored in a database table or Java object cache, depending on the value that you have specified for the `savepoint-manager` configuration option (discussed later in this lesson). When `save-point-id` is restored, it is deleted from the database or Java object cache.

To add the save for later capability to a task flow, perform the following steps:

1. Drag a method call activity from the ADF Task Flow page of the Component Palette to the diagram for the ADF bounded task flow.
2. In the diagram, select the Method Call activity.
3. On the Common page of the Property Inspector, enter an ID for the method—for example, `callCreateSavePoint`.
4. In the method field, enter an EL expression for the savepoint method, for example, `#{controllerContext.currentViewPort.createSavePoint}`. If you created your own method to call `createSavePoint`, enter that method name instead (see the example that follows this list of steps).

Enabling Explicit Save for Later (continued)

5. After adding the method call activity to the diagram, connect it to the other existing activities in the diagram using control flows.
6. You can optionally set properties in the `adf-c-config.xml` file, such as the savepoint default expiration time and where savepoints are stored. This is described later in this lesson.

You can access the `createSavePoint()` method under the `currentViewPort` node under ADF Controller Objects. Instead of calling this method directly, you may want to create your own method that calls `createSavePoint()`, as shown in the following example: (If you do this, you can update `save-point-id` with any attributes you create on your method.)

```
package viewcontroller;
import java.io.Serializable;
import oracle.adf.controller.ControllerContext;
import oracle.adf.controller.ViewPortContext;
public class SaveForLater implements Serializable {
    public SaveForLater() {}
    public String saveTaskFlow() {
        ControllerContext cc = ControllerContext.getInstance();
        if (cc != null) {
            ViewPortContext vpc = cc.getCurrentViewPort();
            if (vpc != null) {
                String id = vpc.createSavePoint();
                return id;
            }
        }
        else
            return null;
    }
    else
        return null;
}
```

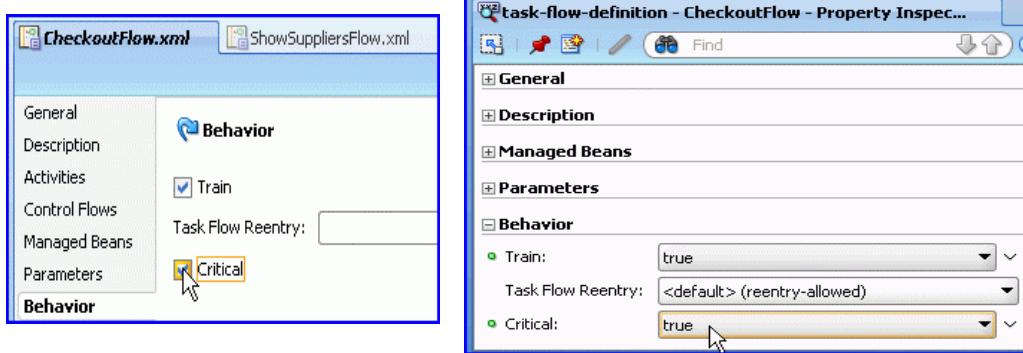
You can override the global settings for individual savepoints for the amount of time that a savepoint remains in effect by calling the `setSavePointTimeToLive()` method. You can include this in the same method that you use to create the savepoint. The method signature is:

```
void setSavePointTimeToLive(long timeInSeconds)
```

Enabling Implicit Save for Later

To enable implicit save for later:

- Change a setting in `adfc-config.xml`
- Mark the task flow as critical



You can use the task flow editor or the Property Inspector to mark the task flow as critical.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enabling Implicit Save for Later

Implicit savepoints are generated only if a critical task flow is present in any of the page flow stacks for any view port under the current root view port. An implicit savepoint is not generated if the request is a request for an ADF Controller resource, such as:

- The task flow call activity
- The task flow return activity
- A Save Point Restore activity
- A dialog

Implicit savepoints are deleted when the task flow at the bottom of the stack completes or a new implicit savepoint is generated, whichever comes earlier.

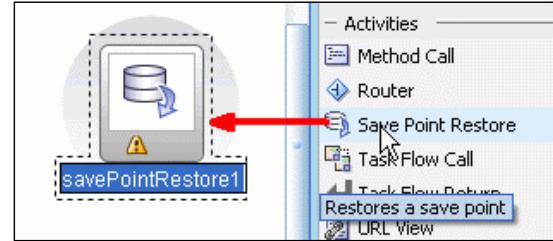
If you want an implicit save for later to occur when the session times out, the user logs out without saving the data, or the user closes the only browser window, then you can select the Critical check box on the Behavior panel of the Overview tab for the task flow definition, or set Critical to `true` on the Behavior panel of the Property Inspector. You must also set an attribute in the `adfc-config.xml` file, as described later in this lesson.

If multiple windows are open when the implicit savepoint is created, a different `save-point-id` is created for each browser window. This includes everything from the root view port of the browser window and down. To return a list of implicitly saved savepoints, you can use `controllerContext.savePointManager.listSavePointIds`.

Restoring Savepoints

A Save Point Restore activity:

- Restores the state of the application
- Deletes the savepoint from its persistence store
- Optionally performs additional logic with a Save Point Restore Finalizer
- Is required in applications that are responsible for restoring `save-point-id` (not necessarily the same application that was saved for later)



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Restoring Savepoints

Restoring a `save-point-id` returns the end user back to the part of the application where the ID was created and restores the application state. Use the Save Point Restore activity to restore a previously persistent savepoint for an application.

When a savepoint is restored, the ADF controller terminates the saved application and restarts the application that was executing when the end user performed a save. The end user's original location in the application is displayed. After the `save-point-id` is restored, it is deleted from its method of persistence (database or Java object cache).

A Save Point Restore activity is not required within every individual application supporting save for later capabilities. It is only required within the applications responsible for restoring the previously persistent `save-point-id`. For example, a Save Point Restore activity would not be required within a Create Expense Report application, but would be within the application used to select previously saved Expense Reports for further updates.

Restoring Savepoints (continued)

Save Point Restore Finalizers

When used in conjunction with a Save Point Restore activity, a Save Point Restore Finalizer contains application-specific logic that may need to be performed to restore the application's state. The Save Point Restore Finalizer is a method that is invoked after the ADF bounded task flow's state has been restored. This method performs any necessary application logic to ensure that the application's state is correct before proceeding with the restore. For example, you might use a Save Point Restore Finalizer to restore logic for time validations.

During the restore process, each bounded task flow on the call stack being restored gets a chance to verify that its restored state is valid and to perform any restore-specific logic it might need.

To add a Save Point Restore Finalizer method to an ADF bounded task flow, perform the following steps:

1. In the editor, open the ADF bounded task flow on which you want to execute a Save Point Restore Finalizer method. This is the ADF bounded task flow being restored, not the task flow executing the Save Point Restore activity.
2. In the editor, click the Overview tab and click Common.
3. Click the button next to the Finalizer.
4. In the dialog box, specify a custom Save Point Restore Finalizer method on a managed bean—for example: `# {pageFlowScope.taskflowBean.finalizerMethod}`

Setting Global Save for Later Properties

Set properties in the `adfc-config.xml` file:

- `savepoint-expiration`
- `savepoint-manager`
- `savepoint-datasource`
- `enable-implicit-savepoints`



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Setting Global Save for Later Properties

In the `adfc-config.xml` file, you can set the following properties that govern the save for later behavior:

- `savepoint-expiration`: Number of seconds that, by default, savepoints in an application are retained. The default is 86400 (24 hours). You can override for individual savepoints by setting `time-to-live` on the savepoint.
- `savepoint-manager`: Type of savepoint persistence. The default is DATABASE, but can alternatively be set to JOC (Java object cache).
- `savepoint-datasource`: The JNDI name of the data source, such as `jdbc/MyConnectionDS` (Typically, this is the name of the database connection appended with DS.)
- `enable-implicit-savepoints`: Set to `true` to enable implicit save for later.

Summary

In this lesson, you should have learned how to:

- Explain ADF BC transaction handling
- Implement task flow transaction control
- Handle transaction exceptions
- Define response to the browser's Back button
- Enable save for later functionality



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 19 Overview: Controlling Transactions

This practice covers the following topics:

- Saving the Shopping Cart
- Updating, Deleting, and Displaying Details of Shopping Cart Items
- Controlling Transactions in the Checkout Flow
- Adding CRUD Functionality for Suppliers



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 19 Overview: Controlling Transactions

In this practice, you complete the application by adding transaction management. So far, the application navigates to all the pages and displays data, but the commit functions are missing. You add commit functionality to the shopping cart, and you add transaction management to the checkout process. You also add the ability to update, delete, and add suppliers.

20

Implementing Security in ADF Applications

ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to:

- Explain the need to secure Web applications
- Describe the security aspects of a Web application
- Implement ADF security:
 - Authentication
 - Authorization:
 - In the data model
 - In the UI for task flows and pages
- Access security information programmatically
- Use Expression Language to extend security capabilities



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Benefits of Securing Web Applications

- Web applications often connect with a single database user account. Therefore, separate application users accounts must be used.
- Identity can be used to:
 - Ensure that only authenticated users can access the application
 - Restrict access to parts of the application
 - Customize the UI (such as pick lists)
 - Provide the username for auditing
 - Set up a Virtual Private Database (VPD)

The red bar spans most of the width of the slide, centered horizontally.

ORACLE

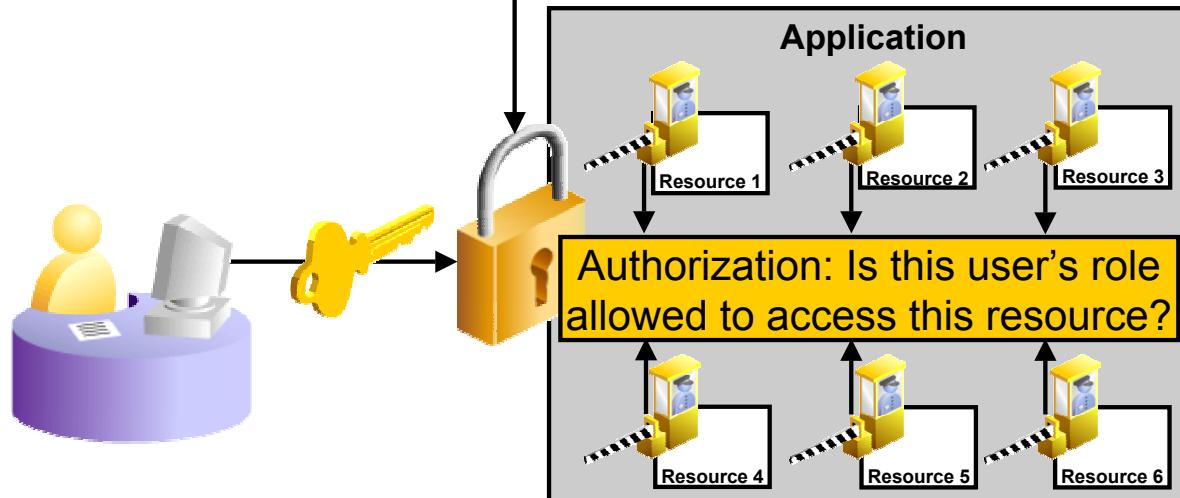
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Benefits of Securing Web Applications

One of the big problems of Web applications is the notion of identity when authentication is not done by the database that is providing the data. The identity of the end user is often needed for a variety of purposes as shown above.

Examining Security Aspects

Authentication: Is this user allowed to access this application?



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Authentication and Authorization

There are two aspects of security for Web applications. You should implement both authentication and authorization in the environment where the application runs:

- **Authentication** determines which users can access the application. Java Authentication and Authorization Service (JAAS) is a package that enables services to authenticate and enforce user-access controls. JAAS is a standard security API that is added to the Java language through the Java Community Process and enables applications to authenticate users and enforce authorization. Java AuthoriZatioN (JAZN) is Oracle's implementation of JAAS. JAAS authentication is defined in the `jazn-data.xml` configuration file, which is referenced by the `jazn.xml` file. In the `jazn-data.xml` file, you can specify user IDs and passwords, create roles, and assign roles to users. You can also set application-specific roles that map to roles that you defined for JAAS.
- **Authorization** determines what functions users are allowed to perform after they enter the application. You can control this by granting certain functionality to the roles that are defined for the application. You specify which objects, pages, and task flows are available to each role.

ADF Security Framework: Overview

- The ADF security framework provides:
 - Standard features required to secure ADF applications
 - More granular declarative security
 - Hierarchical roles with permission inheritance
 - Utility methods for use in EL expressions
 - Different access defined for different roles at the same URL
- It uses JAAS enforced by the ADF binding servlet filter.
- It can authenticate users against a resource provider:
 - LDAP
 - OID
 - XML-based



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ADF Security Framework: Overview

The goal of the Oracle ADF security framework is to provide a standard set of security features that typical ADF applications require, so that the individual applications need not contain their own mechanisms.

JAAS requires custom code at the application level that makes implementing authorization more difficult. Oracle ADF Security simplifies the implementation of a JAAS authorization model by exposing it in a declarative way on various Fusion Web application resources that JDeveloper supports.

Java EE security roles are flat, but ADF security provides simplified permission assignment by using hierarchical roles, allowing for inheritance of permissions.

You can use the ADF Security utility methods in EL to determine whether the user is allowed to perform a known operation, such as accessing a particular task flow or data value.

Within the Oracle ADF framework, JAAS-based security is enforced by the ADF binding servlet filter and the ADF Model Layer of the application. The filter is configured to protect ADF resources and requires the current user to have sufficient access right grants to view the ADF resource, thus providing a much more granular approach to security.

Configure ADF Security Wizard: Configuring ADF Security Authentication

Application > Secure > Configure ADF Security



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Configuring ADF Security Authentication

The Configure ADF Security Wizard enables you to configure ADF Authentication and Authorization, or to configure only ADF Authentication. A third option enables you to remove all ADF security.

To invoke the wizard, select in the Application Navigator the user interface project that contains the pages you want to protect, and then select Application > Secure > Configure ADF Security from the menu.

The first page of the wizard enables you to choose which type of security to configure. When you choose only ADF Authentication, you enable the ADF authentication servlet to support dynamic authentication, but you are not implementing the fine-grained authorization of resources available with ADF authorization.

When you select only ADF Authentication, the servlet requires the user to log in the first time a page in the application is accessed. The servlet also redirects the user to the requested page when the user has sufficient access rights as defined by security constraints. In this case, the developer is responsible for defining security constraints for Web pages; otherwise, all pages are accessible by the authenticated user.

Configure ADF Security Wizard: Choosing the Authentication Type

Most commonly used:

- **HTTP basic authentication:**
 - It uses the browser login dialog box.
 - Cached credentials prevent logout.
- **Form-based authentication:** Developer-designed login page



Select the [authentication type](#) to use for the current web project (StorefrontModel.jpr). Not sure what to choose? The most likely development configuration is selected for you by default.

HTTP Basic Authentication
 HTTP Digest Authentication
 HTTPS Client Authentication (Public Key Certificate)
 Form-Based Authentication
 Generate Default Pages

Login Page: 

Error Page: 

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Choosing the Authentication Type

The second page of the Configure ADF Security Wizard enables you to choose the type of authentication that the application should use. The choices are:

- **HTTP Basic Authentication:** Browser authentication is used. The username and password are not encrypted. This is useful for testing authentication without the need for a custom login page.
- **HTTP Digest Authentication:** Browser authentication is used. The username and password are encrypted, and for this reason it is more secure than basic authentication.
- **HTTPS Client Authentication (Public Key Certificate):** This strong authentication mechanism uses a certificate and transmits over SSL.
- **Form-Based Authentication:** The developer can specify a login page and a page to display when login fails. If you specify a page that uses ADF Faces components, you must use `faces/` in the path, such as `faces/login.html`. If you select the Generate Default Pages check box, JDeveloper generates a login page and an error page for you.

Using Form-Based Authentication

You implement authentication in the UI by:

- Configuring the login in `web.xml` (done by Configure ADF Security Wizard)
- Setting up a login page to accept user credentials with the following elements:
 - A form attribute: `action="j_security_check"`
 - An input text item: `name="j_username"`
 - An input password: `name="j_password"`
- Presenting the login page in HTTPS mode



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Form-Based Authentication

In the Java EE deployment descriptor file, `web.xml`, you set up an application-specific login configuration to specify the type of authentication to perform, such as form-based to display a login form. For a login page to be successful, the form must contain the following tags:

```
<form action="j_security_check" method="post" >
<input type="text" name="j_username">
<input type="password" name="j_password">
</form>
```

This type of login page is automatically created if you choose to use form-based authentication and generate a login page when you implement ADF security. If you define a custom login page, it should be a plain HTML or JSP page that does not use ADF Faces or JSF.

The form submission from the browser is in clear text unless the login page is secured using HTTPS.

Configure ADF Security Wizard: Choosing the Welcome Page

- Specify page where user should go upon authentication
- Ignored if page specified on URL
- If no page specified, user returns to the login page



ORACLE

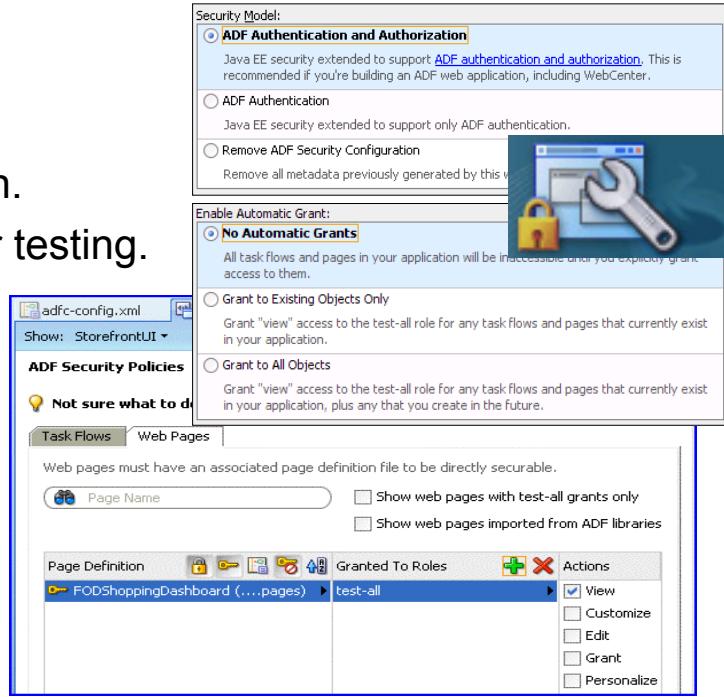
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Choosing the Welcome Page

The final step in the wizard is to optionally redirect the user to a welcome page upon successful authentication. The application uses the specified Web page only when a destination page is not specified as a parameter to the ADF authentication servlet. If you do not specify a redirect Web page, the servlet directs the user back to the page from which login was initiated.

Configure ADF Security Wizard: Enabling ADF Authorization

- Choose the ADF Authentication and Authorization option.
- Select an option for testing.
- To change existing authorizations, use the overview editor for jazn-data.xml.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Enabling ADF Authorization

To use the Configure ADF Security Wizard to configure authorization (in addition to authentication), you can select the “ADF Authentication and Authorization” option on the first page. Then you are presented with an additional wizard page where you specify whether to automatically grant view access on bounded task flows and pages to a special `test-all` role. This enables you to test your application before creating ADF policy grants because the `anonymous-role` built-in role is a member of this role. No login is required to access pages that are granted to the `test-all` role.

You can choose one of the following options for automatic grants:

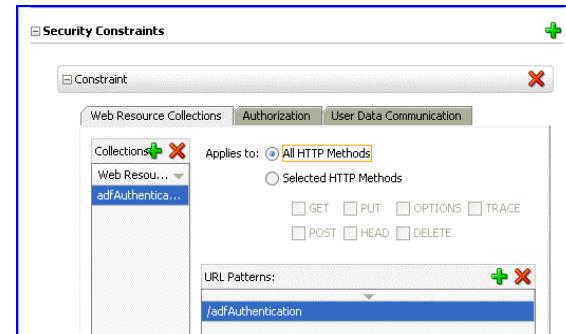
- **No automatic grants:** You must explicitly grant access to bounded task flows and pages.
- **Existing objects only:** View access is granted to the `test-all` role for existing bounded task flows and pages, but not to new ones.
- **All objects:** View access is granted to the `test-all` role for both new and existing bounded task flows and pages.

You can rerun the Configure ADF Security Wizard during different stages of application development to enable or disable automatic grants to the `test-all` application role as desired. If you decide to disable automatic grants, you must remove the ADF security policies by using the overview editor for ADF security policies that you display by double-clicking the `jazn-data.xml` file in the Application Resources window. The editor displays a “Show task flows/web pages with `test-all` grants only” check box to enable you to easily locate the `test-all` application role grants. Click Remove Role (red X) to remove the grants made for testing purposes.

Files Modified by Configure ADF Security Wizard: web.xml

web.xml modifications:

- ADF authentication servlet definition and mapping
- Security constraint
- Login configuration



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Files Modified by Configure ADF Security Wizard: web.xml

The web.xml file contains configuration and deployment information for a Web application. When you define ADF authentication, the Configure ADF Security Wizard modifies the web.xml file as follows:

- Defines the authentication servlet
- Provides the servlet mapping for enforcing security
- Defines a security constraint on authentication Web resource (You can use this page to add further security constraints.)
- Defines the login configuration
- Specifies required security roles (You may use this page to add further roles.)

The web.xml file is in the /public_html/WEB-INF subdirectory of the UI project.

Because every user of the application is required to be able to log on, the security constraint defined against the ADF authentication servlet should allow all users to access this Web resource. As such, the security role associated with the constraint should encompass all users. To simplify this task, the valid-users role is provided.

Note: You must not delete the adfAuthentication constraint from the web.xml file. Deleting it would prevent users from logging on to the application in the manner supported by ADF. Any other static security constraints would continue to work and invoke a login if required.

Other Files Modified or Created by Configure ADF Security Wizard

The screenshot shows the 'ADF Security: ADF Authentication' project structure in the Oracle JDeveloper IDE. It includes sections for 'Policy Store', 'Credential Store', 'Anonymous Provider', 'Authentication Type: Form-Based Authentication', and 'Identity Store: Application XML'. A legend indicates that blue boxes represent 'Modified' files and grey boxes represent 'Created' files. The 'adf-config.xml' file is highlighted as modified. To the right, two code snippets are shown:

```

<CredentialStoreContext credentialStoreClass=
"oracle.adf.share.security.providers.jazn.JAZNCredentialStore"
credentialStoreDefaultUser="anonymous"
credentialStoreLocation=".//credential-jazn-data.xml"/>
<sec:JaasSecurityContext initialContextFactoryClass=
"oracle.adf.share.security.JAASInitialContextFactory"
jaasProviderClass=
"oracle.adf.share.security.providers.jps.JpsSecurityContext"
authorizationEnforce="true"
authentication="..."/>
```



```

<serviceProviders>
<serviceProvider
class="oracle.security.jps.internal..."/>
...
</serviceProvider>
...
<serviceInstances>
<serviceInstance
provider="credstore.provider"/>
...
</serviceInstance>
...
<jpsContexts default="Storefront-12">
<jpsContext name="anonymous">
<serviceInstanceRef ref="credstore"/>
...
</jpsContext>
</jpsContexts>
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Other Files Modified or Created by Configure ADF Security Wizard

The Configure ADF Security Wizard performs the following additional file creations or modifications:

| File | Location | Configuration Performed |
|--|---|--|
| adf-config.xml | /adf/META-INF relative to Web application | <ul style="list-style-type: none"> Credential store context JAAS security context |
| jps-config.xml (JPS stands for Java Platform Security.) | /src/META-INF relative to Web application | <ul style="list-style-type: none"> Oracle Platform Security context for optional credential store Oracle Platform Security context for optional policy store Oracle Platform Security context for optional anonymous user |
| jazn-data.xml | /src/META-INF relative to Web application | <ul style="list-style-type: none"> Default realm name for optional lightweight XML application-specific identity store Policy store |

Enabling Users to Access Resources

To give users access:

- Define a security realm in the identity store:
 - Create users.
 - Create roles.
 - Assign users to roles.
- Define an application policy in the policy store:
 - Create application roles.
 - Map the identity roles to the application roles.
 - Grant the application roles access to resources.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

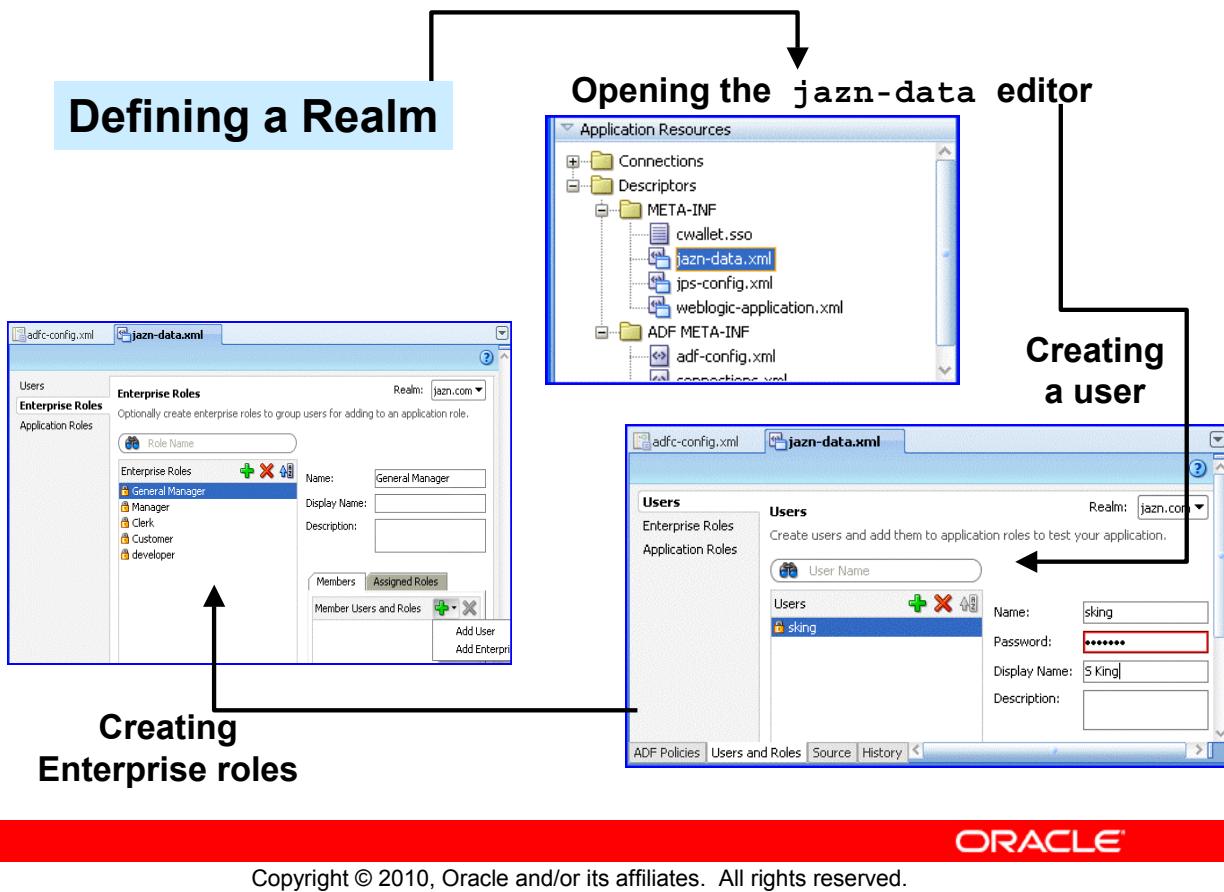
Enabling Users to Access Resources

To grant users access to resources, you first set up users and roles in the identity store as part of a security realm. You then define application roles in the policy store and assign identity store roles to the application roles.

Permissions to use resources are granted to roles, rather than to users directly.

The next few slides elaborate on these concepts.

Defining Users and Roles in the Identity Store



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Users and Roles in the Identity Store

In ADF security, a realm is a set of users and roles in an identity store. You create users and roles, and then you assign users to roles.

You can use the same realm for many different applications. Within a realm, applications participate in browser-based single sign-on, so that being authenticated in one application in a realm means that you do not have to be authenticated again in another application in the same realm, as long as you do not close the browser between requests.

A default realm is a policy space that is used whenever no realm is explicitly mentioned during the user login process. The default realm is `jazn.com`, but you can create additional realms as needed.

The identity store can be file-based or LDAP-based, with grants stored in an LDAPv3-compliant directory, such as Oracle Internet Directory. You may want to seed the identity store with a temporary set of users to mirror the actual users' experience in your production environment.

To add test users, perform the following steps:

1. Double-click `jazn-data.xml` to open the editor.
2. In the `jazn.com` realm, click the “Users and Roles” tab at the bottom of the screen.
3. You click Add to create a user on the Users finger tab, and to create a role on the Enterprise Roles or Application Roles finger tab.

Note: By default, the `anonymous-role` built-in role is a member of the `test-all` role, and, therefore, any resources granted to `test-all` do not require a login.

Defining Security Policies

- A security policy is a set of grants made to roles.
- To define a security policy:
 - Create application roles.
 - Assign identity store roles to application roles.
 - Grant permissions to roles.

The red bar spans most of the slide width, centered horizontally.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

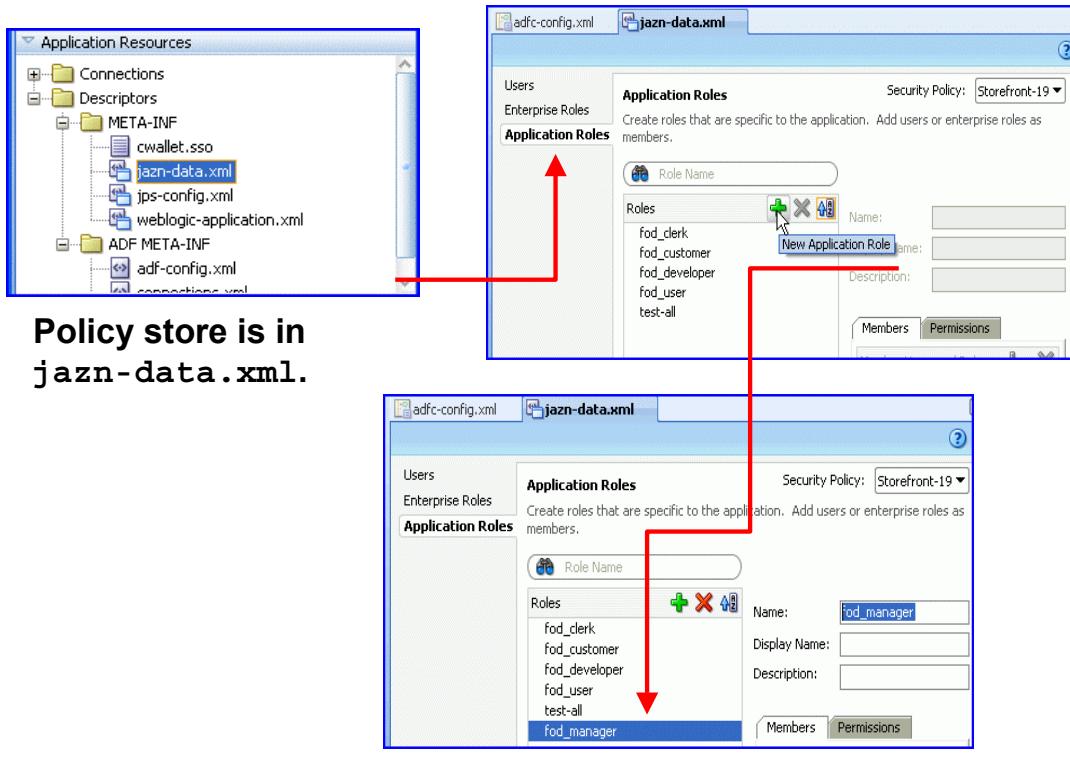
Defining Security Policies

When you enforce ADF authorization, all ADF resources are secure by default. To make these resources available to authenticated users, you must configure a policy store that consists of grants made to specific application roles, and you must assign to those roles the roles that you created in the identity store. Subsequent slides elaborate on these actions.

When you first configure ADF authorization, you are not required to have a policy store in place. The `test-all` role enables you to run and test your application before creating the policy store. However, eventually you need to customize the rights for members of actual application roles to access ADF resources.

Oracle ADF Security enables you to define an access policy for a variety of application resources. For example, you can control access to a particular task flow based on the access right grants that you make in the policy store for the ADF task flow.

Defining Application Roles in the Policy Store



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

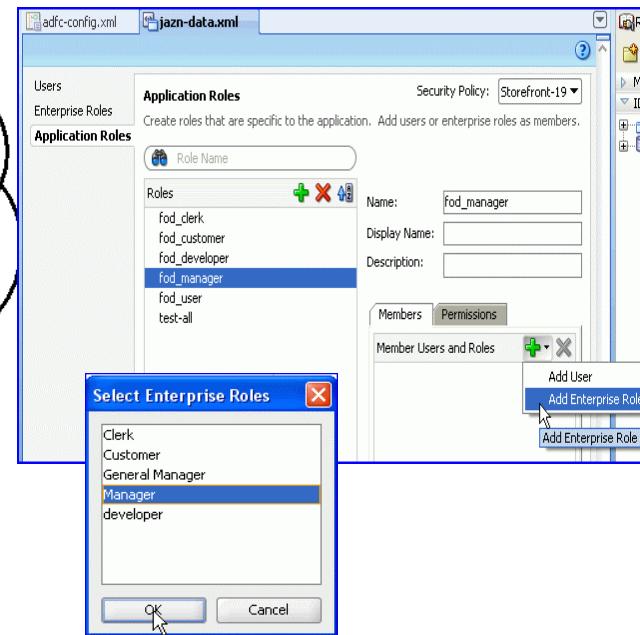
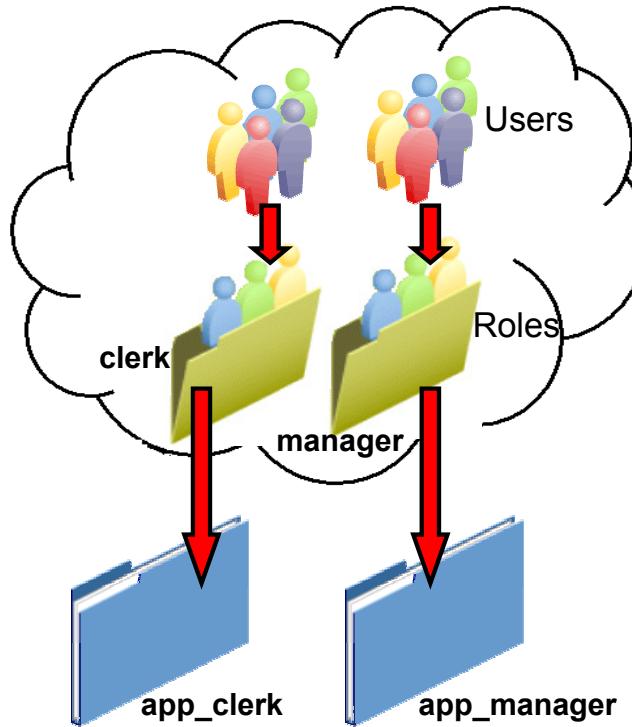
Defining Application Roles in the Policy Store

Because Web application security is a role-based access control mechanism with permissions granted to application roles, you must define a set of roles in the policy store that are specific to your application. You eventually map the application roles of your policy store to enterprise roles defined in the deployment environment. This enables a user who is a member of a given enterprise role to access resources that are accessible from the associated application role. Enterprise roles are defined in the identity store of the security provider and are controlled only at an administrator level. This provides a level of abstraction to the enterprise roles and enables development to proceed against functional roles.

During development, the policy store is file-based, with access right grants stored in the `jazn-data.xml` file in `<app-role>` elements under `<policy-store>`. To add a role to the policy store, perform the following steps:

1. Double-click `jazn-data.xml` in the Application Resources panel.
2. Click the “Users and Roles” tab at the bottom of the screen.
3. Click the Application Roles finger tab.
 - a. Check whether the Application in the Security Policy field is the one you want to define roles for.
 - b. Click the New Application Role icon (the green + sign), and in the fields to the right, enter a name and, optionally, a Display Name and Description.

Assigning Identity Store Roles to Application Roles



Mapping an identity role to an application role

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Assigning Roles to Users

Now that you have defined a realm with users and roles and have also defined application roles, the next step is to map the application roles to the roles in the identity store realm.

On the “Users and Roles” tab for `jazn-data.xml`, click the Application Roles finger tab. Ensure that the appropriate application is selected in the Security Policy field at the top right. Then select the application role that you want to map to an enterprise role, in the Roles panel, and on the Members tab on the right, click the green + sign. From the Select Enterprise Roles drop-down list, select the role that you want to assign to the application role.

Granting Permissions to Roles

You can associate roles to grants on resources:

| Authorization Point | Grants Issued On: | Defined In: |
|---------------------|------------------------------|---------------------------------------|
| Groups of pages | Bounded task flows | jazn-data.xml editor |
| Individual pages | Page definitions * | jazn-data.xml editor |
| Rows | Entity objects or attributes | EO security and authorization editors |

* To secure page with no data, create an empty page definition file.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Granting Permissions to Roles

You eventually associate each application role with a specific set of grants to ADF resources. The security policy names the application role as the principal of the grant, rather than specific users.

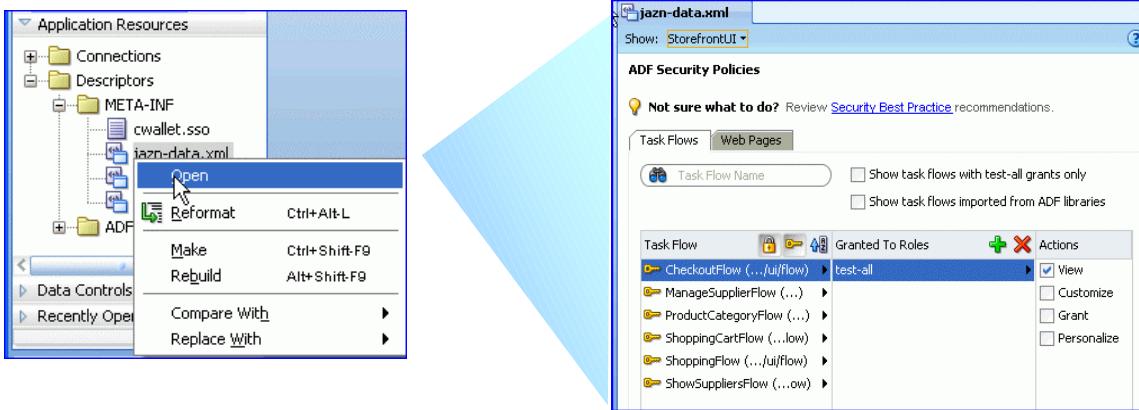
Oracle ADF Security defines the following authorization points for your application's access policies:

- Groups of Web pages with grants issued on ADF task flows. The pages that the task flow defines are all securable with grants.
- Individual Web pages with grants issued on ADF page definitions. This is useful for any Web page that is not contained in an ADF task flow. A page definition is created automatically for Web pages with components that are bound to data. If you want to secure a Web page that does not display databound components, you can create an empty page definition file for it.
- Specific row-level data accessed through the Business Components data model project, with grants issued on entity objects or their attributes

For the view-controller project, you use the overview editor for ADF security policies to secure ADF resources, including ADF task flows and ADF page definitions. To open the overview editor, double-click jazn-data.xml in the Application Resources panel in JDeveloper. This editor is available only if you have ADF Security configured.

Securing Groups of Pages (Bounded Task Flows)

- Prevent unauthorized access to secured task flows.
- Provide developers with the ability to:
 - Secure a bounded task flow as a logical entity
 - Write security-aware bounded task flows and pages



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Granting Permissions on Task Flows

ADF unbounded task flows are not securable on their own, although the individual pages are securable. Bounded task flows are secured by default when ADF Security is enabled. ADF enables you to grant the `view` permission on bounded task flows. Users who belong to a role with the `view` permission can read and execute a bounded task flow and view its pages.

Objects within a task flow inherit the containing object's security unless overridden on the local object. Page-level security resides with the page, and is not held in the view activity metadata. This ensures that there is only one place to access security information for that page, regardless of how many task flows reference it.

Securing task flows can provide for simpler security administration by allowing security to be controlled in terms of a unit of application functionality, rather than having to deal with all the individual pages that may arise in the implementation of that application function.

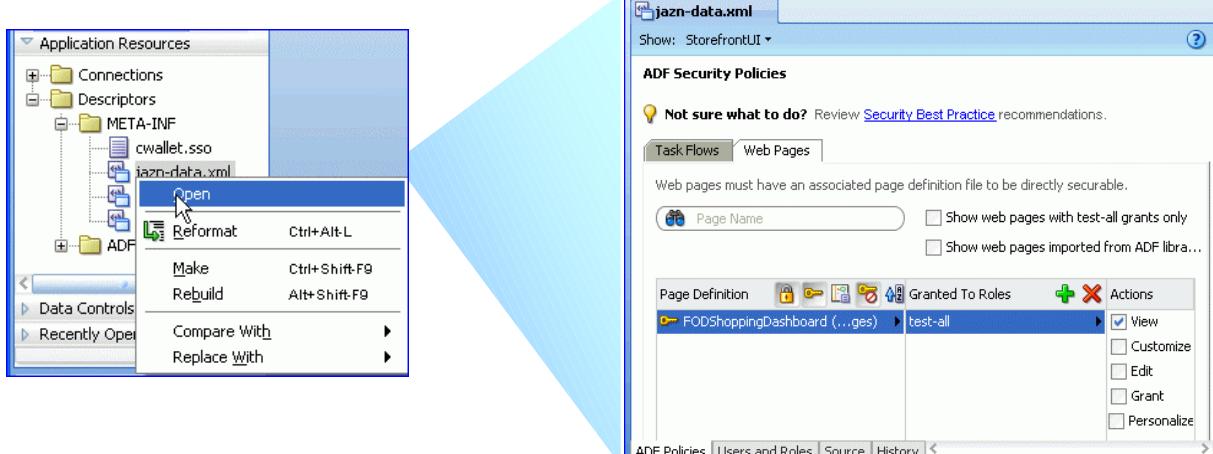
The controller calls the security module on each request to ensure that the required access has been granted. The controller will not attempt to cache or retain any authorization information.

Authorization failure results in navigation to the designated exception activity. The exception thrown is `ADFSecurityRuntimeException`.

You can use Expression Language (EL) to check whether the user is authorized to access a task flow. When a user is required to log in to view a page, you can use EL within the page to check whether a user is authorized to view specific components.

Securing Individual Pages (Page Definitions)

- Determines whether the user is allowed to navigate to (view) a page
- Not needed on pages in secured task flows



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Granting Permissions on Individual Pages

The permissions you can grant for an ADF page definition can control if the authenticated user can view the page. Later JDeveloper releases enable you to specify additional permissions.

Page definitions are secure by default when ADF Security is enabled. However, page-level security is not checked within bounded task flows. If a developer intends to have a higher-security page within a bounded task flow, the developer must include that page in a nested task flow.

To define security for a page, perform the following steps:

1. Open `jazn-data.xml` and click the Web Pages tab in the ADF Policies editor. This displays all pages for which there are page definition files.
2. In the Page Definition column, select the page.
3. In the "Granted to Roles" column, click Add and select the previously defined application role.
4. In the Actions column, select the action that you want to grant to the role.

ADF BC Model Authorization

The purpose is to:

- Prevent unauthorized access to entity objects or attributes
- Enable developers to:
 - Secure access to an entire entity object or only certain attributes
 - Specify the actions that members of a role can perform on entity objects or attributes



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ADF BC Model Authorization

Oracle ADF entity objects in the model project are security-aware, which means that predefined resource-specific permissions exist that a developer can grant. Additionally, you can secure just the individual attributes of entity objects.

Securing Row Data (Entity Objects or Attributes)

You can enable security on:

- Entire entity objects:

| Security: OrderEO | |
|-------------------|-------------------------------------|
| Operation | Enabled |
| read | <input type="checkbox"/> |
| update | <input checked="" type="checkbox"/> |
| removeCurrentRow | <input checked="" type="checkbox"/> |

- Individual attributes:

| Security: OrderStatusCode | |
|---------------------------|-------------------------------------|
| Operation | Enabled |
| update | <input checked="" type="checkbox"/> |

| ADF Business Component | Securable Operation | Expected Mapped Action | Implementation |
|------------------------|---------------------|------------------------|---|
| Entity object | read | Read | View rows of result set. |
| | removeCurrentRow | Delete | Delete a row from the bound collection. |
| | update | Update | Update any attribute. |
| Attribute of EO | update | Update | Update specific attribute. |

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Securing Row Data (Entity Objects or Attributes)

Entity objects that you secure restrict users from updating data on any Web page that renders a UI component bound by an ADF binding to the data accessed by the secured entity object. Additionally, when you secure an entity object, you effectively secure any view object in the data model project that relies on that entity object. As such, entity objects that you secure define an even broader access policy that applies to all UI components bound to this set of view objects.

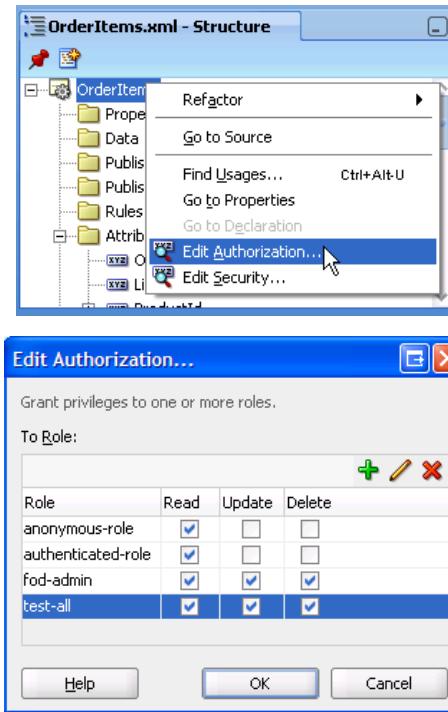
To enforce row-level security checks on entity objects or their attributes, perform the following steps:

- Invoke the editor for the entity object.
- Select either General (for the entire entity object) or Attributes in the tree at the left.
- If you selected Attributes, select the attribute on which to define security.
- In the Security section of the page, select the check boxes for the operations on which to enable security.

Alternatively, you can right-click the entity object in the Application Navigator and select Edit Security.

Granting Privileges on Entity Objects or Attributes

- In the Structure window, right-click the entity object or attribute and select Edit Authorization.
- Select privileges to grant to application roles.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Granting Privileges on Entity Objects or Attributes

After you have enabled security for an entity object or attribute, you can assign privileges for updating on entity objects and attributes, and for reading and deleting on entity objects. You can also add, remove, or edit application roles from the Edit Authorization dialog box.

You invoke the Edit Authorization dialog box from the context menu of the entity object in the Structure window. The dialog box displays all application roles; you check the appropriate permissions to grant to each role.

Application Authentication at Run Time

Two types:

- **Implicit:** Based on JAAS permissions for anonymous-role role
- **Explicit:** Based on security constraint on authentication servlet that you can define by using the Configure ADF Security Wizard

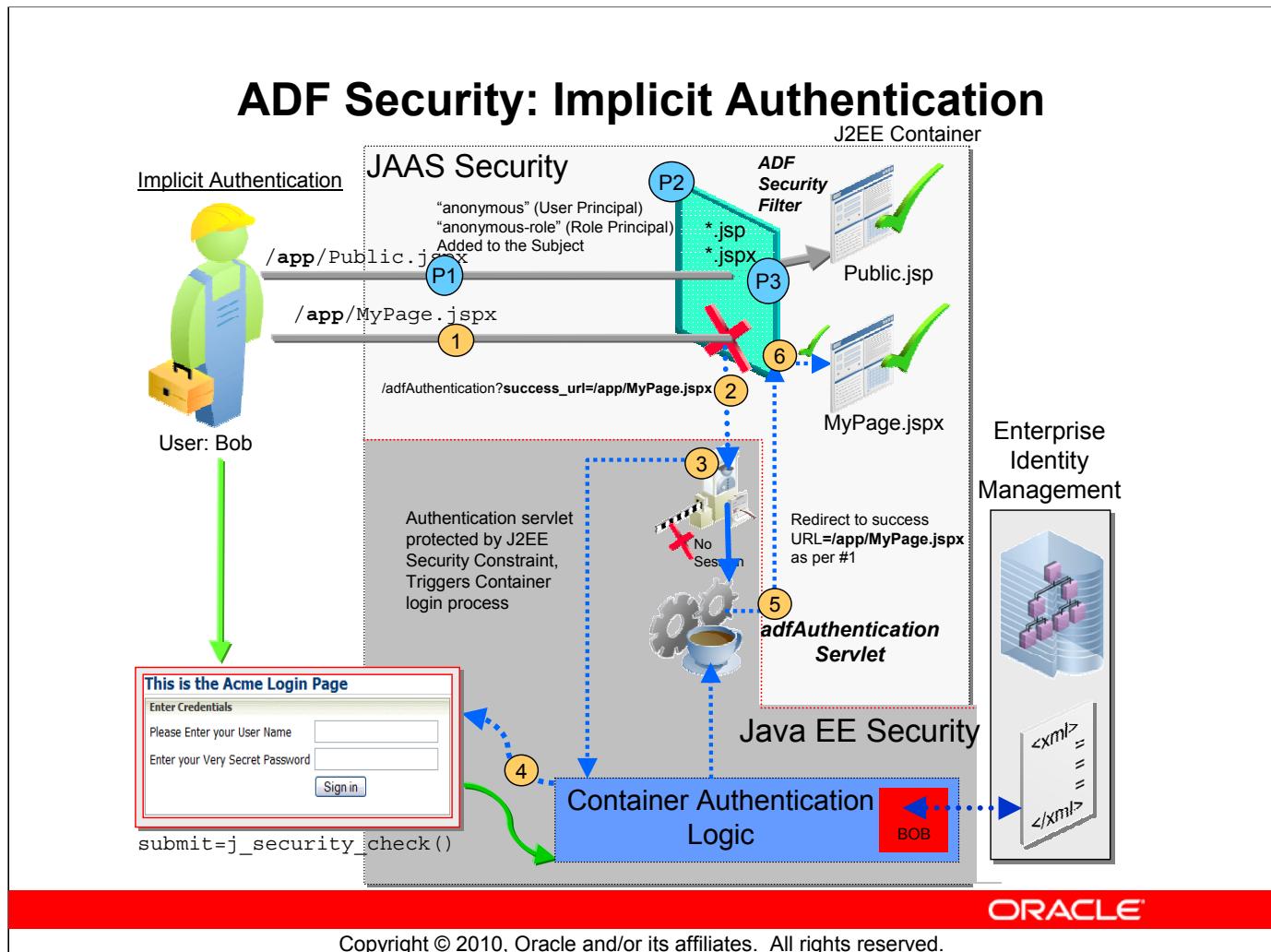


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ADF Security: Configuring Application Authentication

Application authentication can be one of two types:

- **Implicit:** The adfBindings servlet filter checks to see if page is viewable based on JAAS permission that you have assigned for the anonymous-role role, which can access any page on which no web.xml security constraint is defined for the anonymous-role role, and the view privilege has been granted.
- **Explicit:** A login link is directed to the adfAuthentication servlet, which is secured through a Java Platform, Enterprise Edition (Java EE) security constraint. In an explicit authentication scenario, a public page has a login link, which when clicked, triggers an authentication challenge to log in the user. The login link may optionally specify some other target page that should be displayed (assuming the authenticated user has access) after the successful authentication. On the first access to a page, if there is no Subject defined, then one is created containing the anonymous user principal and the anonymous-role role principal. With this role principal, the user can access any page on which no web.xml security constraint is defined for the anonymous-role role principal and for which the view privilege has been granted.



ADF Security: Implicit Authentication

In an implicit authentication scenario, authentication is triggered automatically if the user is not yet authenticated and the user tries to access a page that is not granted to the `anonymous-role` role. After successfully logging in, another check is performed to verify whether the authenticated user has view access to the requested page.

When an unauthenticated user tries to access a page, the `adfBindings` servlet filter intercepts the request and checks to see whether the page is defined as viewable by the `anonymous-role` role. If the requested page is public:

P1: If this is the first access to a page within the application and if there is no subject currently defined.

P2: The security layer creates a subject containing the anonymous user principal and the `anonymous-role` role principal.

P3: The user is then allowed access to the public page.

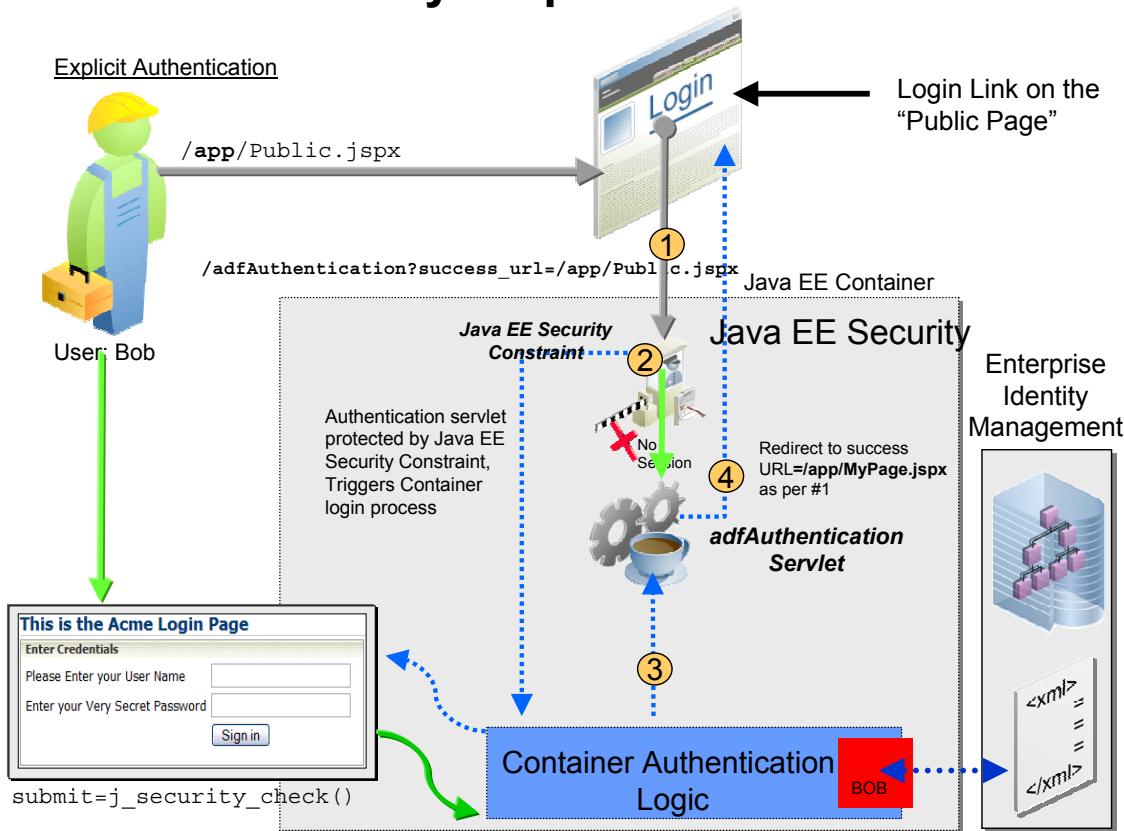
For a request to a secured page:

1. The user requests the secured `mypage.jspx`.
2. The `adfBindings` servlet filter redirects the request to the Oracle ADF authentication servlet, passing in the URL to the requested page as the success URL.

ADF Security: Implicit Authentication (continued)

3. Because the `adfAuthentication` servlet has a Java EE security constraint on it, calling the `adfAuthentication` Servlet results in the Java EE container invoking the configured login mechanism.
4. Based on the container's login configuration, the user is prompted to authenticate. The example shown in the slide uses a form-based login, so the appropriate login form is displayed. The user enters credentials and posts the form back to the container's `j_security_check()` method, so that the Java EE container can authenticate the user.
5. Upon successful authentication, the container redirects the user back to the `adfAuthentication` servlet.
6. The `adfAuthentication` servlet forwards the user to the requested page. If ADF Security is enforced, that resource appears if the user has access privileges.

ADF Security: Explicit Authentication



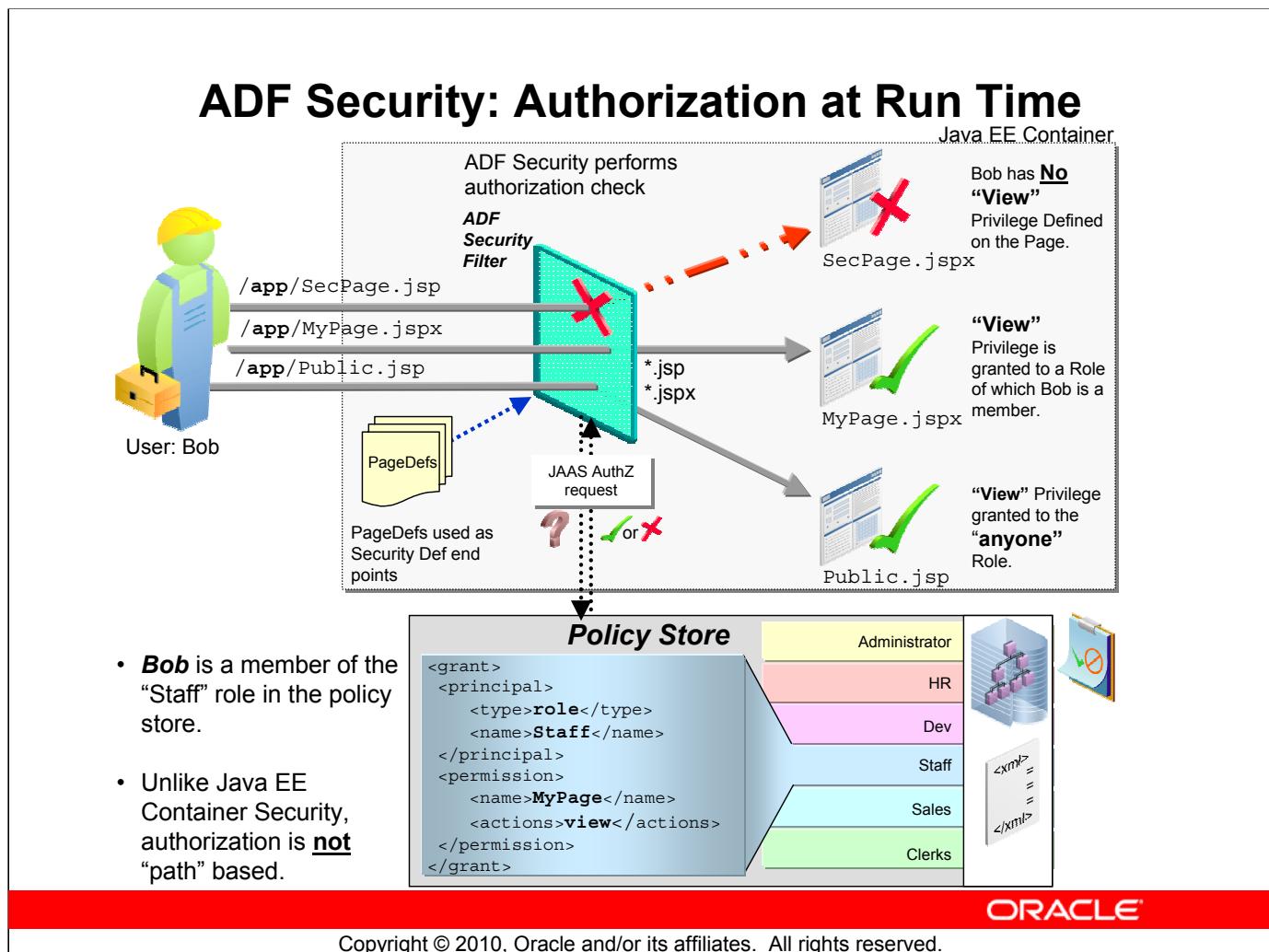
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

ADF Security: Explicit Authentication

Because Fusion applications have the notion of a public page, the following mechanism enables the user to explicitly authenticate from within that public page:

1. The unauthenticated user (with only the anonymous user principal and the anonymous-role role) clicks the Login link on the public page. The Login link is a direct request to the adfAuthentication servlet, which is secured through a Java EE security constraint.
2. The current page is passed as a parameter to adfAuthentication servlet. As with the implicit case, the security constraint redirects the user to the container's login component.
3. After the container authenticates the user, the request is returned to the adfAuthentication servlet, which subsequently returns the user to the original public page, but now with the new user and role principal.



ADF Security: Authorization at Run Time

The security implementation within a Fusion application is based around JAAS policies, which are held in a policy store (independent of the application) and accessed at run time. The policy determines if the user has access to the defined Web resource and what granular permissions (actions) have been granted to the user.

In this case, the user, Bob, is a member of the enterprise role Staff in the identity management solution.

Assuming that Bob is already logged in and successfully authenticated, when he tries to access mypage.jsp, the Oracle ADF Security enforcement logic intercepts the request and checks the page definition of that page to see whether permission is required. In the example in the slide, the View privilege on the requested page has been granted to the Staff role, and, therefore, for Bob the mypage.jsp appears.

However, when Bob tries to access SecPage.jsp later, permission is required and Bob does not belong to a role that has access to the resource; therefore, a security error appears.

Programmatically Accessing ADF Security Context

Is ADF security turned on?

```
if (ADFContext.getCurrent().getSecurityContext().isAuthorizationEnabled())
{ ... }
```

Is the user logged on?

```
public boolean isAuthenticated() {
    return ADFContext.getCurrent().getSecurityContext().isAuthenticated(); }
```

Who is the user?

```
public String getCurrentUser() {
    return ADFContext.getCurrent().getSecurityContext().getUserName(); }
```

Is the user in a specified role?

```
public boolean isUserInRole(String role) {
    return ADFContext.getCurrent().getSecurityContext().isUserInRole(role); }
```

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Programmatically Accessing ADF Security Context

You can use methods of the ADF Security context to obtain information about users and roles.

Because the enforcement of Oracle ADF Security can be turned on and off at the container level independent from the application, you should determine whether Oracle ADF Security is enabled before making permission checks. You can evaluate the `isAuthorizationEnabled()` method to achieve this.

It is not possible to check whether the user principal is null to determine whether the user has logged on or not, because it is either anonymous for unauthenticated users or the actual username for authenticated users. You can use the `isAuthenticated()` method to determine whether the user has authenticated.

You can determine the current username (either anonymous for unauthenticated users or the actual username for authenticated users) with the `getUserName()` method.

You can use the `isUserInRole()` method to determine whether the user is a member of a specified role.

It is a good idea to make user and role information available throughout the application by using session-scoped beans.

Using Expression Language to Extend Security Capabilities

You can integrate Expression Language in two ways:

- Using built-in global security expressions:

```
<af:commandLink action="accounts"
    rendered="#{securityContext.userInRole['admin']}"
    text="Manage Accounts"/>
```

- Using a security proxy bean:

```
<af:commandLink action="accounts"
    rendered="#{userInfo.admin}"
    text="Manage Accounts"/>
```

See next slide for source of this expression.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Expression Language to Extend Security Capabilities

You can use expressions to resolve properties such as Rendered, ReadOnly, and Disabled at run time. The built-in global security expressions, shown in the first example in the slide, enable such access. Alternatively, you could write your own beans, as in the second example.

Using Global Security Expressions

| Expression | Purpose |
|---|---|
| <code>#{{securityContext.userName}}</code> | Username of the authenticated user |
| <code>#{{securityContext.userInRole ['role list'] }}</code> | Is the user in <u>any</u> of these roles? |
| <code>#{{securityContext.userInAllRoles ['role list'] }}</code> | Is the user in <u>all</u> of these roles? |
| <code>#{{securityContext.userGrantedPermission ['permission'] }}</code> | Does the user have this permission granted? |
| <code>#{{securityContext.taskflowViewable ['target'] }}</code> | Does the user have view permission on the target task flow? |
| <code>#{{securityContext.regionViewable ['target'] }}</code> | Does the user have view permission on the target region? |

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Global Security Expressions

Global security expressions are available without installing any extensions. You can use them on any page, even if the page does not have a page definition file.

Using a Security Proxy Bean

A managed bean can expose a Boolean property that the UI expressions can consume.

Example: UserInfo bean:

```
public boolean isAdmin() {  
    return (ADFContext.getCurrent().  
        getSecurityContext().isUserInRole("admin"));  
}
```

Example: UI expression:

```
#{{userInfo.admin}}
```

Checking for multiple roles is a problem; you could end up writing many convenience methods.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using a Security Proxy Bean

A security proxy bean is a Java class that is usually stored in the session as a managed bean. This bean exposes security information in a form that Expression Language can process, usually as a method called `is<something>`, which translates to a simple expression as in the example shown in the slide.

However, when you need to check for multiple combinations, the Expression Language becomes complex or you need many convenience methods. Additionally, you need to generate accessors for each role.

Summary

In this lesson, you should have learned how to:

- Explain the need to secure Web applications
- Describe security aspects of a Web application
- Implement ADF security:
 - Authentication
 - Authorization:
 - In the data model
 - In the UI for task flows and pages
- Access security information programmatically
- Use Expression Language to extend security capabilities



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 20 Overview: Implementing ADF Security

This practice covers the following topics:

- Configuring the Application to Use ADF Security
- Defining Users in the Identity Store
- Defining Enterprise Roles
- Implementing Security on Task Flows and Pages
- Testing Application Authentication and Authorization
- Implementing Entity Object Security
- Testing Entity Object Security
- Accessing Security Context Programmatically
- Conditionally Displaying a Component Based on User Role
- Modifying the Hard-Coded Username to Use the API



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice 20 Overview: Implementing ADF Security

In the practices for this lesson, you set up the application to use security. You create users and roles in an identity store, and you assign permissions to roles. You also create login and error pages. You then use ADF security to manage access to pages, task flows, and entity objects, and you programmatically access the security context by using global security expressions and a security proxy bean.

Modeling the Database Schema

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Describe JDeveloper's tool for database modeling
- Create a schema model
- Modify the schema model
- Reconcile the modifications to the database
- Import database object definitions to an offline database
- Reconcile offline database objects with the database



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

When designing an application to interact with the database, an understanding of the database schema is essential, along with the ability to modify the schema as needed. This lesson shows you how to use JDeveloper tools to define a model of a database schema and to use the model to modify the schema.

Modeling Database Schemas

With JDeveloper's database modeler, you can:

- Create a schema model
- Modify the schema model
- Reconcile changes to the database



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Modeling Database Schemas

Offline database object modeling in projects has been extended to provide for:

- Index-organized tables
- External tables
- Partitioned tables and indexes
- Materialized (including partitioned and indexed) views and view logs
- Object Collections
- Triggers
- Domain indexes
- Storage properties (tablespace and so on)
- Auto-generated column values
- SXML definition generation
- Cross-referencing multiple database objects within or between projects

Goals of Database Modeling

Database modeling can help you achieve the following goals:

- For existing database objects:
 - Visualize the schema to help analyze whether it meets requirements.
 - Make any necessary modifications.
- For new database objects:
 - Design schema to support business requirements.
 - Generate DDL to create the required database objects.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Why Perform Database Modeling?

Visualizing existing tables in a data diagram can help you understand whether the schema might need modifications to support the business requirements.

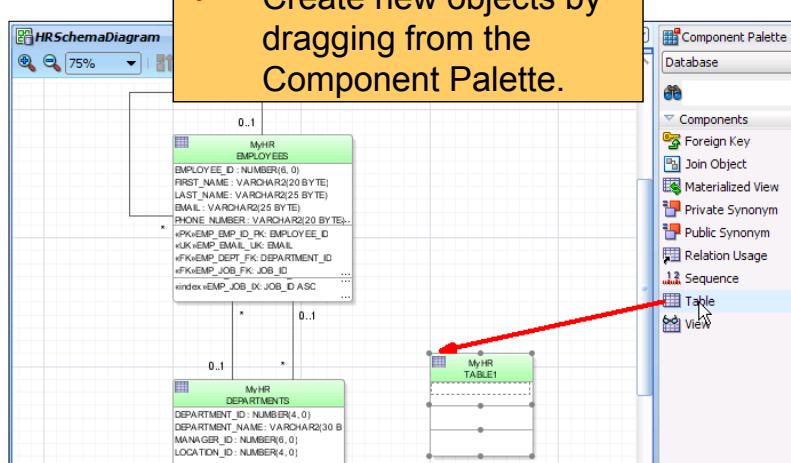
If the tables do not already exist, you can use the same diagrammer to design them and generate the DDL scripts to create the tables.

Characteristics of the JDeveloper Database Modeler

With the JDeveloper Database Modeler, you can:

- Model existing objects, such as tables and foreign keys, by dragging from Database Navigator.
- Use the Property Inspector to set properties on diagram and its objects.

- Create new objects by dragging from the Component Palette.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Characteristics of the JDeveloper Database Modeler

Although this course concentrates on working with an existing database schema, it is important to understand that you also can use JDeveloper's database diagrammer to design and create new tables.

Database Modeling Tools in JDeveloper

You can work with database objects in the following ways:

- Online: Create and modify schema objects directly in a database through a JDeveloper connection.
- Offline:
 - Model metadata for database objects outside the context of a database schema.
 - Import objects from online database.
 - Generate changes to online schemas.
- Diagrammer:
 - Create and modify objects by dragging from Component Palette and Database Navigator.
 - Import objects from offline database model.
 - Generate changes to online schemas.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Database Modeling in JDeveloper

You can use JDeveloper to work directly with database objects through a database connection. Alternatively, you can work with offline database definitions, which you can subsequently generate to a database schema.

JDeveloper provides the tools that you need to create and edit database objects such as tables and constraints outside the context of a database. You can create new tables and views and generate the information to a database, or you can import database objects from a database schema, make the changes that you want, and then generate the changes back to the same database schema, to a new database schema, or to a file that you can run against a database at a later date. Alternatively, you can use JDeveloper's modeling tools to visualize your offline database objects on a diagram.

You can use JDeveloper's database diagrammer to design and create new tables as well. You can import database objects from the offline database into the diagrammer. The diagrammer allows you to import database objects from online schemas and also generate changes to online schemas.

Modeling Database Objects Offline

You can create offline database definitions in the following ways:

- Create new offline database objects.
- Copy from database with filters.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Modeling Database Objects Offline

You can create offline database definitions based on objects that exist in a database schema in one of the following ways:

- **Using Drag and Drop:** You can drag tables, views, synonyms and sequences from a database schema onto a database diagram, where they become accessible as offline database objects.
- **Using Import with Filters:** You can import tables, views, sequences, and synonyms from a database schema to an offline schema where they become available as offline database objects. You can apply filters in the wizard to display only the objects that you are interested in, and when there are a large number of objects in the schema, you can turn off auto-query so that the wizard does not refresh every time you enter a filter character.

Creating a New Offline Database

- To create a definition for an offline database in which you can model database objects, perform the following steps:
 1. In the Application Navigator, right-click a project and select New to display the New Gallery.
 2. In the New Gallery, expand Database Tier, and select Offline Database Objects.
 3. Select Offline Database to launch the Create Offline Database Wizard.
 4. In the pages of the wizard, enter properties for the database, such as its name.
- After an offline database is created, you can create various database object definitions within it.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a New Offline Database

To create an offline database definition, perform the following steps:

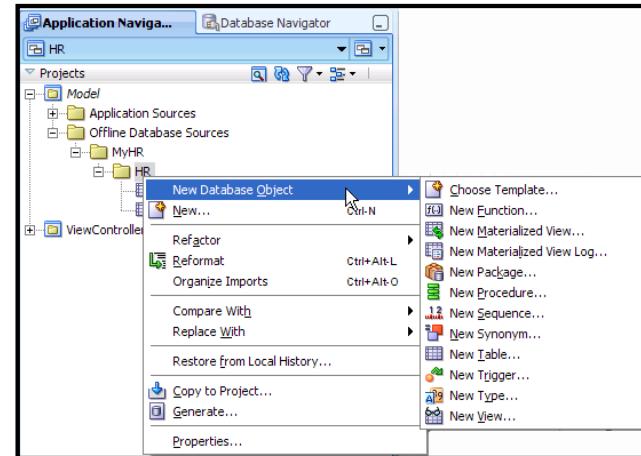
1. In the Application Navigator, expand the application and the project that you want to work in. Right-click a project and select New to display the New Gallery.
2. From the New Gallery, expand Database Tier, and select Offline Database Objects.
3. Select Offline Database to launch the Create Offline Database Wizard.
4. On the pages of the wizard, enter properties for the database, such as its name.

After an offline database is created, you can create various database object definitions within this database.

Creating New Schema Objects in an Offline Database

Create new schema objects in an offline database by using wizards to define:

- Functions
- Materialized views and view logs
- Packages
- Procedures
- Sequences
- Synonyms
- Tables
- Triggers
- Types
- Views



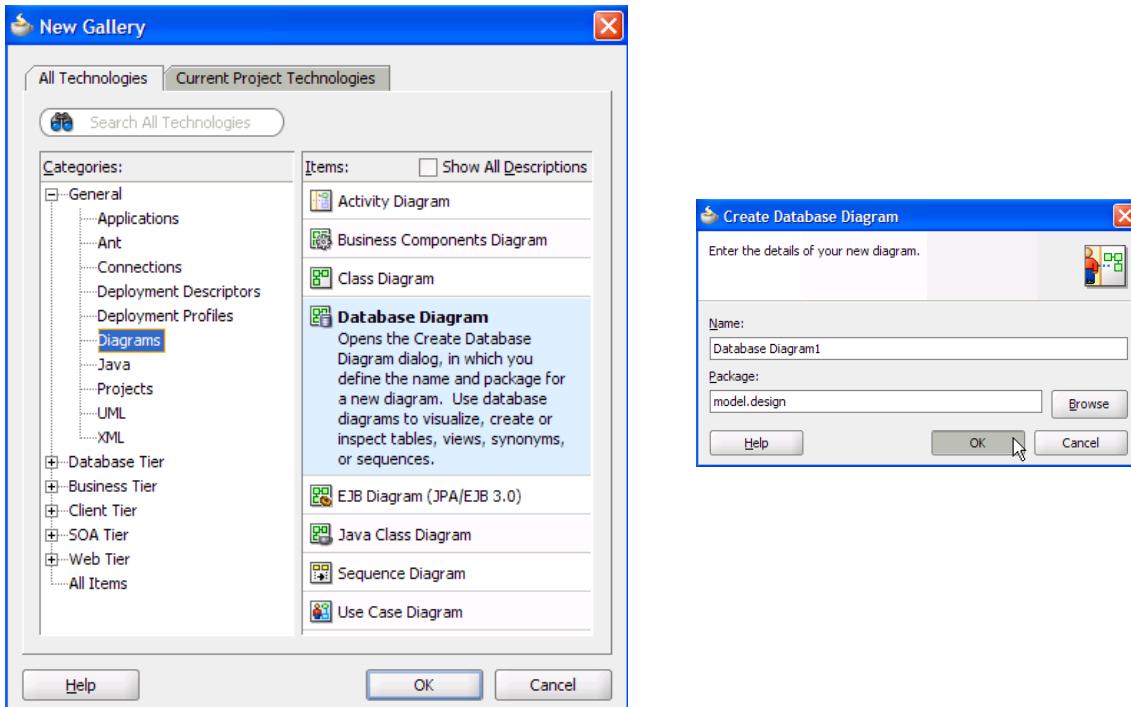
ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating New Schema Objects in an Offline Database

You can create new schema objects in an offline database by invoking one of the New Database Object wizards and entering the required information.

Creating a Database Diagram



ORACLE

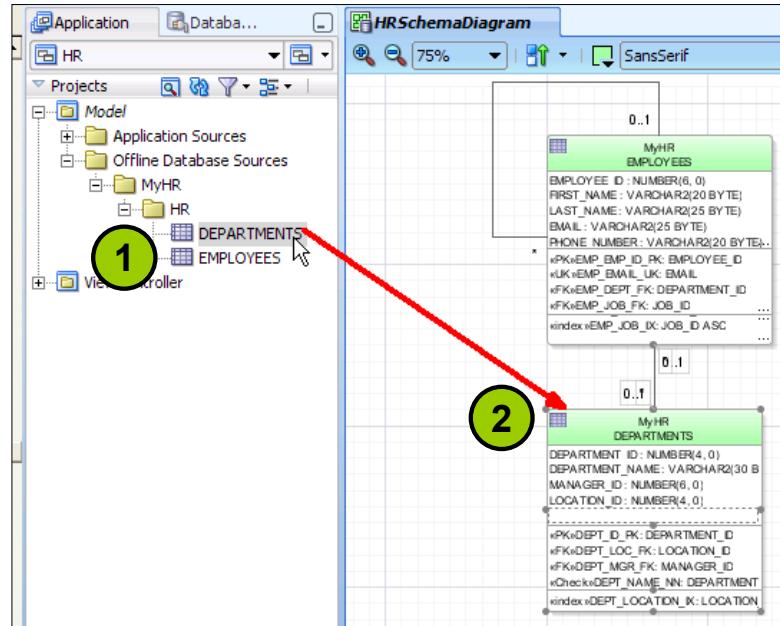
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Database Diagram

After creating a named connection in the Database Navigator to browse database schema objects, you can create a database diagram in any JDeveloper project. To create a new diagram, perform the following steps:

1. Select the project in which you want to create the diagram.
2. Select File > New to invoke the New Gallery.
3. In the New Gallery, expand the General category and select Diagrams.
4. Select Database Diagram from the Items list, and then click OK.
5. In the Create Database Diagram dialog box, specify a name for the diagram and, optionally, a package name, and then click OK.

Importing Tables to the Diagram from an Offline Database



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Importing Tables to the Diagram from an Offline Database

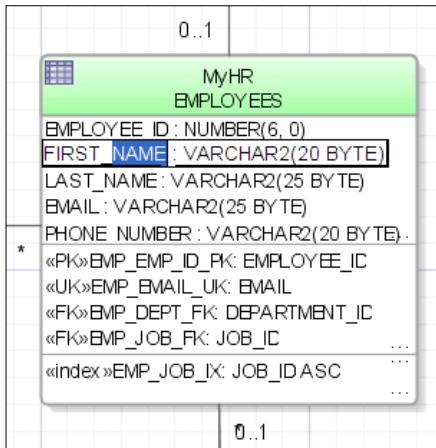
You can import tables from an offline database as follows:

1. In the Offline Database Navigator, select the desired offline object and drag it to the diagram.
2. A representation of the offline object appears on the diagram.

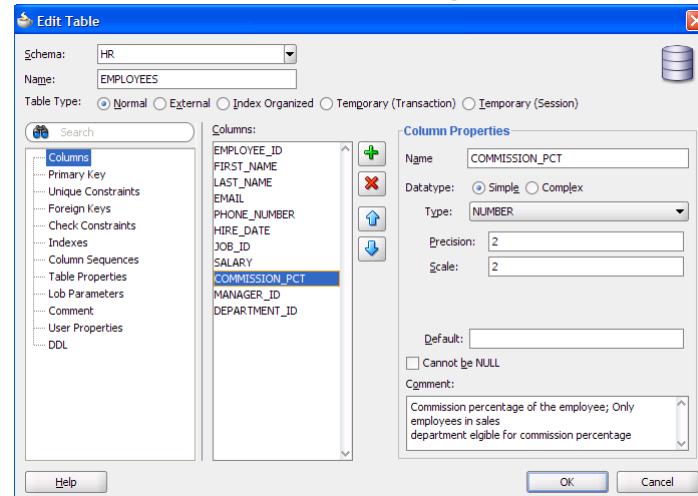
When you drag related tables to the diagram, the foreign key relations are also depicted visually. Offline objects' attributes can be modified in the Offline Database Navigator or on the diagram, and are kept in synch.

Editing Objects on the Diagram

In-place editing:



The Edit Table dialog



ORACLE®

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

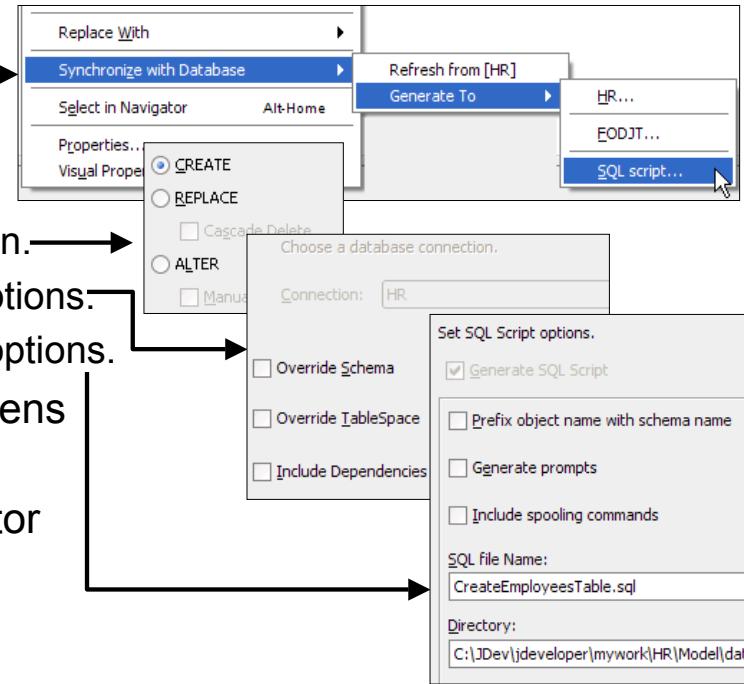
Editing Objects on the Diagram

There are two ways to edit objects on the diagram:

- In-place editing: Click inside the object, such as an attribute, and then click it again to put the text in edit mode.
- Double-click the object to invoke the editor, such as the table editor shown in the slide.

Generating Changes from the Diagram to the Database

1. Invoke the wizard.
2. In the wizard:
 - Select objects.
 - Specify operation.
 - Set database options.
 - Set SQL script options.
3. Right-click the editor to run it.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

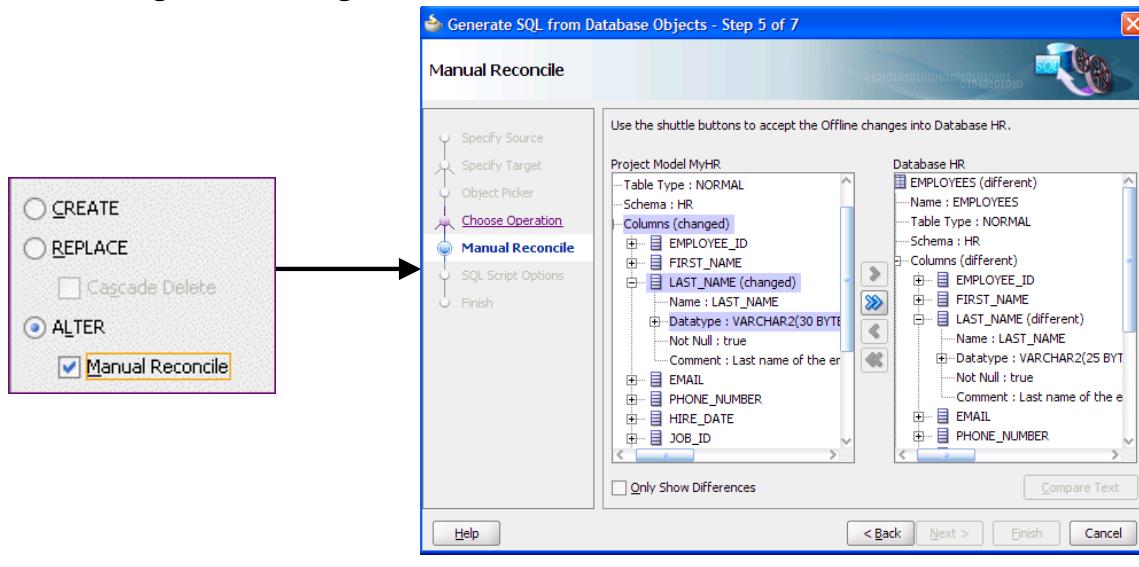
ORACLE

Generating Changes to the Database

If you created a database diagram, you can right-click the object or objects on the diagram to generate the DDL to create, replace, or alter those objects. From the context menu, select Synchronize with Database > Generate To > SQL Script. This invokes a wizard where you specify the object, operations, and SQL script options that you want to use. When you complete the wizard, the script that it generates opens in the editor, where you can run it.

Reconciling Changes to the Database

Selecting ALTER with Manual Reconcile enables you to choose the changes to be generated to the database.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

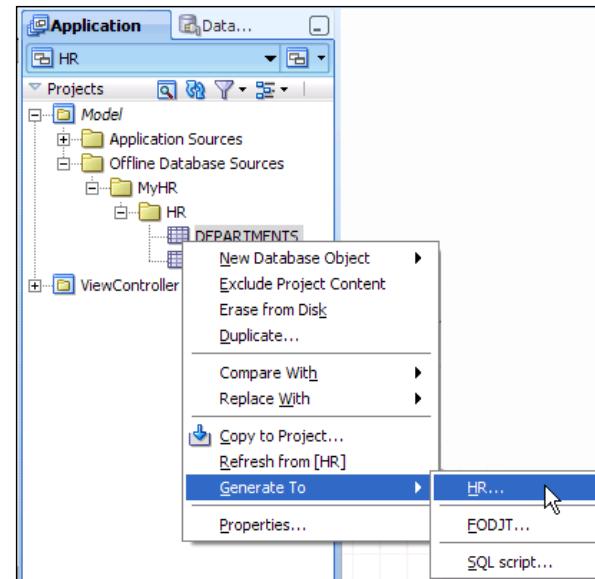
Reconciling Changes to the Database

If you choose the ALTER operation, the offline objects are compared with the online objects and DDL is generated to make the online objects match the offline objects. You can optionally specify that you want to manually reconcile the objects. After the wizard is completed, if you specified manual reconciliation, an additional dialog box is displayed that enables you to compare the objects to be generated to the database with the objects in the database, so that you can choose which changes to generate to the database.

Generating Changes from the Offline Database Object to the Database

To generate imported offline database objects to the database:

1. Right-click the object in the Applications Navigator.
2. Select Generate To.
3. Select the JDeveloper connection, or create a SQL script or SXML.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

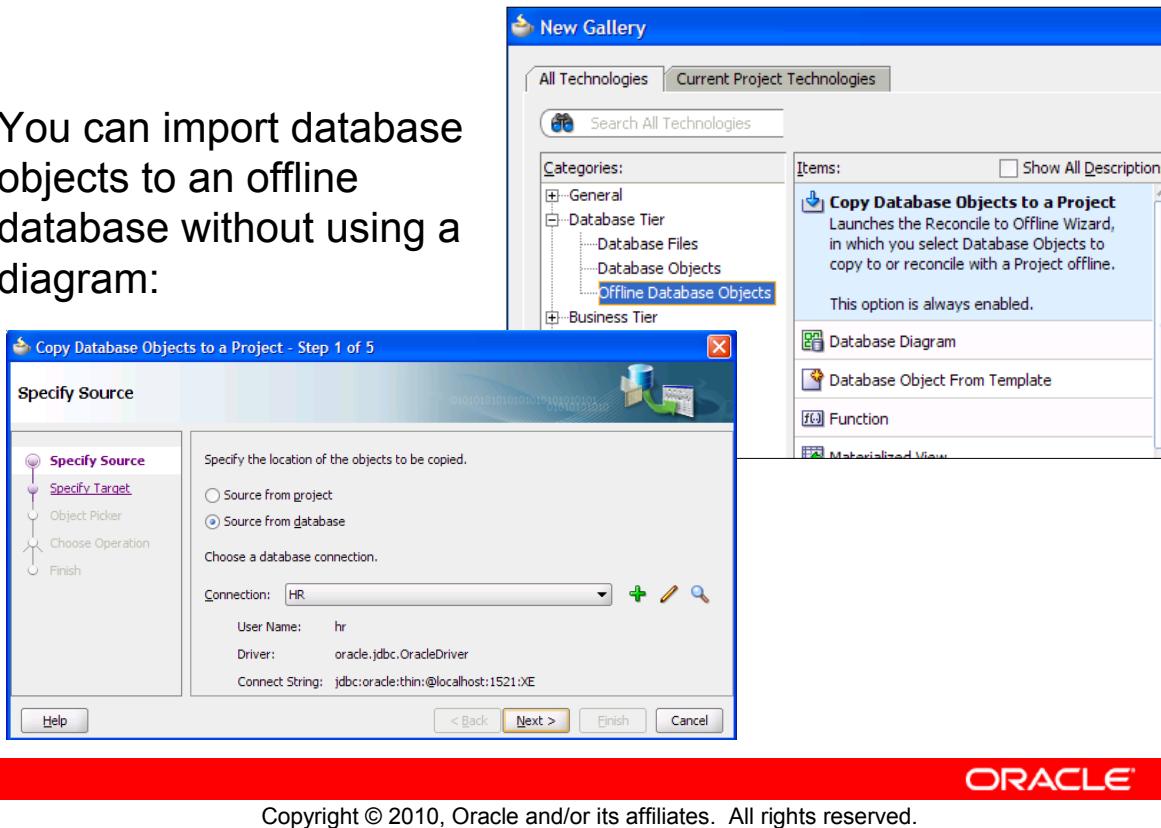
Generating Changes from the Offline Database Object to the Database

Whether or not you have created a database diagram, you can generate changes to the database by right-clicking the database object or objects in the Applications Navigator and selecting Generate To. You can then select a JDeveloper connection, or you can generate a SQL script or SXML.

When you choose to generate to a JDeveloper connection as depicted in the slide, or to SQL scripts, this also invokes the “Generate SQL from Offline Database Objects” Wizard. Its pages enable you to specify whether to create, replace, or alter, and whether to generate a SQL script, along with script options. If you choose to generate to SXML, an XML file is created and opens in the editor.

Importing Database Objects Without a Diagram

You can import database objects to an offline database without using a diagram:



ORACLE

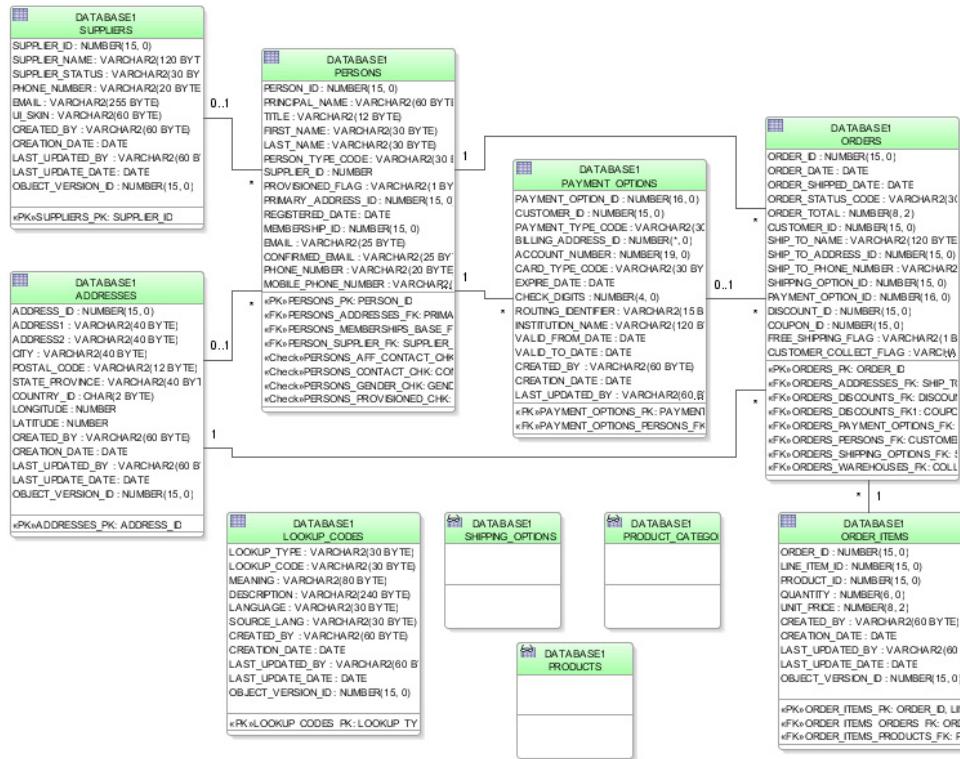
Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Importing Database Objects Without a Diagram

If you do not want to create a database diagram, you can use the “Copy Database Objects to a Project” item in the New Gallery by expanding Database Tier and selecting Offline Database Objects in the categories list. This invokes the “Copy Database Objects to a Project” Wizard, which has the following pages:

- Specify Source (shown in the slide): Choose the JDeveloper connection from which to import.
- Specify Target: Select the application and project to import to, and select or create a new offline database to contain the imported objects.
- Object Picker: Select the database objects to import.
- Choose Operation: Choose whether to create, replace, or alter offline database objects.
- Finish: A summary of the specified import. Click Finish to start it.

Presenting the Storefront Schema



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Presenting the Storefront Schema

The schema used for the course application is a subset of the Fusion Order Demo (FOD) schema. You can read about how to obtain the Fusion Order Demo in Section 2.2 of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework 11g Release 1 (11.1.1.2.0)*.

The main tables that are used in the course application are Persons, Suppliers, Orders, and Order Items. The application also uses some views, such as Products and ProductCategories.

Summary

In this lesson, you should have learned how to:

- Describe JDeveloper's tool for database modeling
- Create a schema model
- Modify the schema model
- Reconcile the modifications to the database
- Import database object definitions to an offline database
- Reconcile offline database objects with the database



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice Overview: Modeling the Schema for the Course Application

This practice covers the following topics:

- Creating a Database Diagram
- Adding Existing Schema Objects to the Diagram
- Adding a New Database Object to the Diagram
- Creating the DDL to Add the New Object to the Database



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice Overview: Modeling the Schema for the Course Application (optional)

In this practice, you use the database diagrammer to create a visual representation of the database schema. You also create a new database component (a table) and create the DDL to add it to the schema.

This practice can be found in the Practices document in the Practices for Appendix A section.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Osi S.R.L. use only

Deploying ADF Applications

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Objectives

After completing this lesson, you should be able to do the following:

- Create deployment profiles
- Configure deployment options
- Use JDeveloper to deploy an application
- Use the WebLogic Server Administration Console to deploy an application
- Use Ant to deploy an application
- Test the deployed application



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Lesson Aim

This lesson teaches you how to deploy an ADF BC application by using JDeveloper or by using the WebLogic Server Administration Console. You also learn how to use Ant to automate the deployment process.

Steps in the Deployment Process

Deployment tasks:

- Prepare an application or a project for deployment.
- Prepare Oracle WebLogic Server (WLS) for ADF deployment.
- Deploy the application with one of the following techniques:
 - Deploy directly from JDeveloper to WLS.
 - Deploy to an EAR file and use the application server tool for deploying to the application server.
 - Use a script to deploy.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Steps in the Deployment Process

Deployment is the process of packaging application files as an archive file and transferring it to a target application server.

The following are some common deployment techniques that you can use during the application development and deployment cycle:

- Deploy directly from JDeveloper to an application server. You would most likely do this while developing the application so that you can quickly deploy it for frequent testing (see later slides in this lesson regarding testing), but you also can use JDeveloper to deploy an application to the production server.
- Deploy from JDeveloper to an EAR file, and then use the tools of the application server to deploy the EAR file to the target server. You can do this when you are ready to test features such as Lightweight Directory Access Protocol (LDAP) and single sign-on (SSO) that may not be available on the development server. You can also use this technique to deploy the application for production.
- Use an automated script. Scripts are used in testing, and system administrators typically run scripts to deploy applications in production environments.

Whether you use JDeveloper to deploy directly to the application server or to an EAR file, you must first create the appropriate deployment profile to define how the application is packaged. If deploying to Oracle WebLogic Server (WLS), you also must prepare WLS for ADF application deployment. These tasks are discussed in subsequent slides.

Steps in the Deployment Process

Deployment tasks:

- Prepare application or project for deployment.
- Prepare Oracle WebLogic Server (WLS) for ADF deployment.
- Deploy the application with one of the following techniques:
 - Deploy directly from JDeveloper to WLS.
 - Deploy to an EAR file and use application server tool for deploying to the application server.
 - Use a script to deploy.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Configuring Deployment Options

To configure an application for deployment, you create:

- Deployment profile, which can be one of three types:
 - ADF Business Components: Business Component Java ARchive (JAR)
 - Fusion Web application: Enterprise ARchive (EAR) and Web ARchive (WAR) deployment profiles
 - Fusion Web application with customizations: Oracle ARchive (OAR) and Metadata ARchive (MAR)
- Deployment descriptors



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Configuring Deployment Options

Deployment is a two-step process. First, you package the application into deployable artifacts by using deployment profiles and deployment descriptors. Then you deploy these artifacts to the target environment.

There are different types of deployment, whose options are configured in deployment profiles:

- Applications based on ADF Business Components use a Business Component archive (JAR).
- Fusion Web applications typically use EAR and WAR deployment profiles.
- Fusion Web applications that include customizations deploy to a target repository using OAR, a type of deployment that allows for the packaging of metadata customizations. An OAR can contain a MAR packaged inside it—a MAR is a compressed archive of selected metadata.

This lesson discusses the first two types of deployment. For further information about deploying Fusion Web applications with MDS customizations, see the *Fusion Developer's Guide for ADF*.

Creating Deployment Profiles

Deployment profiles:

- Specify how application is packaged for deployment
- Can be application level or project level

The screenshot shows three panels from the Oracle Fusion Middleware New Gallery:

- Creating an application deployment profile:** Shows categories like General, Applications, and Deployment Descriptors.
- For the Model project:** Shows options for Business Components Archive, EAR File, and WAR File.
- For the View-Controller project:** Shows options for Business Components EJB Session Bean, Client JAR File, and WAR File.

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating Deployment Profiles

A deployment profile defines the way the application is packaged into the archives that are deployed into the target environment. The deployment profile:

- Specifies the format and contents of the archive file to be created
- Lists the source files, deployment descriptors, and other auxiliary files to be packaged
- Describes the type and name of the archive file to be created
- Highlights dependency information, platform-specific instructions, and other information

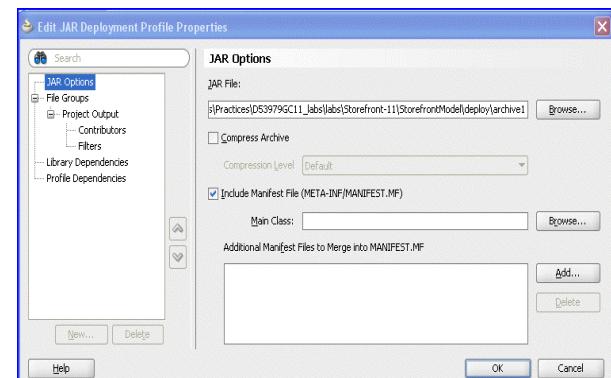
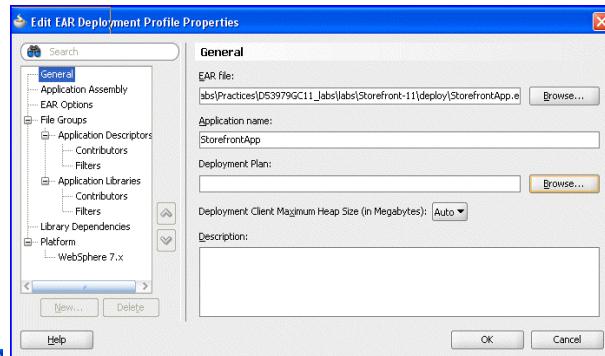
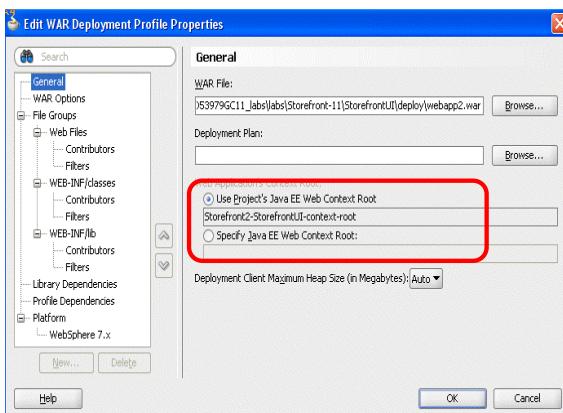
You can define and store deployment profiles at the application or project level, giving you more flexibility and enabling direct referencing and sharing. For example, you could create a WAR profile for projects associated with the View and Controller layers and a Business Components JAR for the model project. You could then create an application-level EAR profile that assembles the model and view-controller profiles for deployment.

You can create a deployment profile by using the New Gallery, or by using application properties or project properties. The types of profiles that are available for you to choose depend on the type of project or application for which you are creating a profile. For example, as the slide shows, application deployment profiles are EAR files, whereas project deployment profiles offer different choices depending on the type of project.

Specifying Deployment Profile Options

Deployment profile options:

- Specified in profile properties
- Different depending on the type of profile



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Specifying Deployment Profile Options

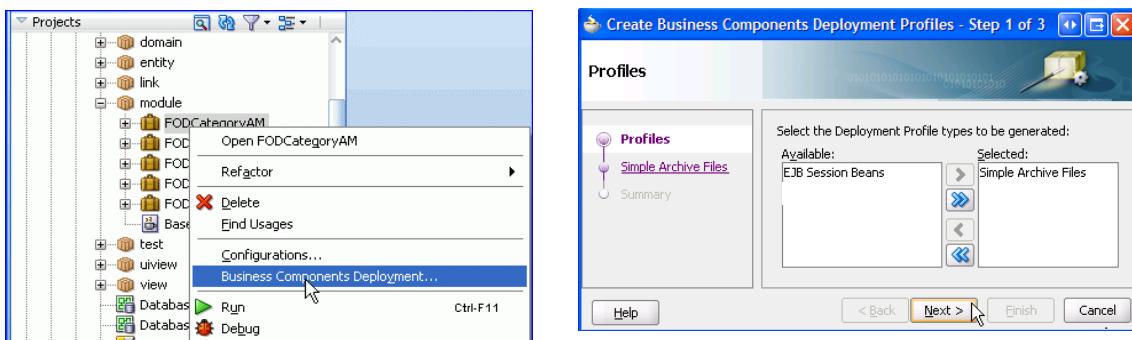
After you choose the type of deployment profile to create and give it a name, you are presented with a Deployment Profile Properties dialog box. There is a tree in the left pane of the dialog box, and the options in that tree are different depending on the type of deployment profile you are creating. You can get more information about the choices that are available from online Help.

Subsequent slides discuss the different types of deployment profiles in greater detail.

Creating a Business Components Deployment Profile

You can create business components deployment profiles by using one of the following:

- Context menu
- Project Properties dialog box
- New Gallery



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Business Components Deployment Profile

To create a deployment profile for the ADF BC library, perform the following steps:

1. In the Application Navigator, right-click the application module and select Business Components Deployment from the context menu.
2. In the Create Business Components Deployment Profiles Wizard, ensure that Simple Archive Files is in the Selected list. Click Next.
3. On the second page of the wizard, you can name the profile, and then click Finish.

Another way to create a deployment profile is to invoke the Project Properties dialog box, select Deployment, and then click New.

These methods of creating a deployment profile do not give you the ability to set a path for deploying the JAR file, but set it to deploy to a default path, which is a \deploy directory under your project. If you want to specify where to deploy the JAR file, you can create the deployment profile by using the New Gallery instead (General > Deployment Profiles > ADF Library JAR File). This invokes a dialog box where you can specify a path to the JAR file, which is important if you want to deploy it to a central location for sharing with other developers.

After you create the deployment profile, you can use it to create the JAR. Right-click the project, select Deploy, select the JAR to create, and select Deploy again.

Web Module Deployment

- It is used for deploying ADF BC applications to a Web tier.
- It contains ADF BC components and JSPs and servlets.
- It is used for deploying JSP, JSF, and servlet applications.
- Deployment is to a Java EE archive (.ear or .war) file.
- JDeveloper creates the archive file and deployment descriptors.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Web Module Deployment

A Web module deployment is a standard Java EE deployment model designed to be deployed to an application server such as Oracle Application Server. It contains all the components of the business logic layer and JSPs or servlets. The deployment is created as a WAR file or an EAR file that contains all the ADF Business Components code and JSP or servlet code.

JDeveloper provides complete deployment in this model and creates all the parts necessary for Web module deployment automatically.

This deployment model has the business logic tier and client tier sharing the same JVM, which is hosted by an application server.

The next few pages outline the steps in creating and deploying a Web module.

Typical Web Application Deployment Example

To deploy a typical Web application that uses ADF Business Components:

- Create a WAR deployment profile for the UI project
- Create an EAR deployment profile for the application
- Deploy the application to the EAR file or directly to an application server

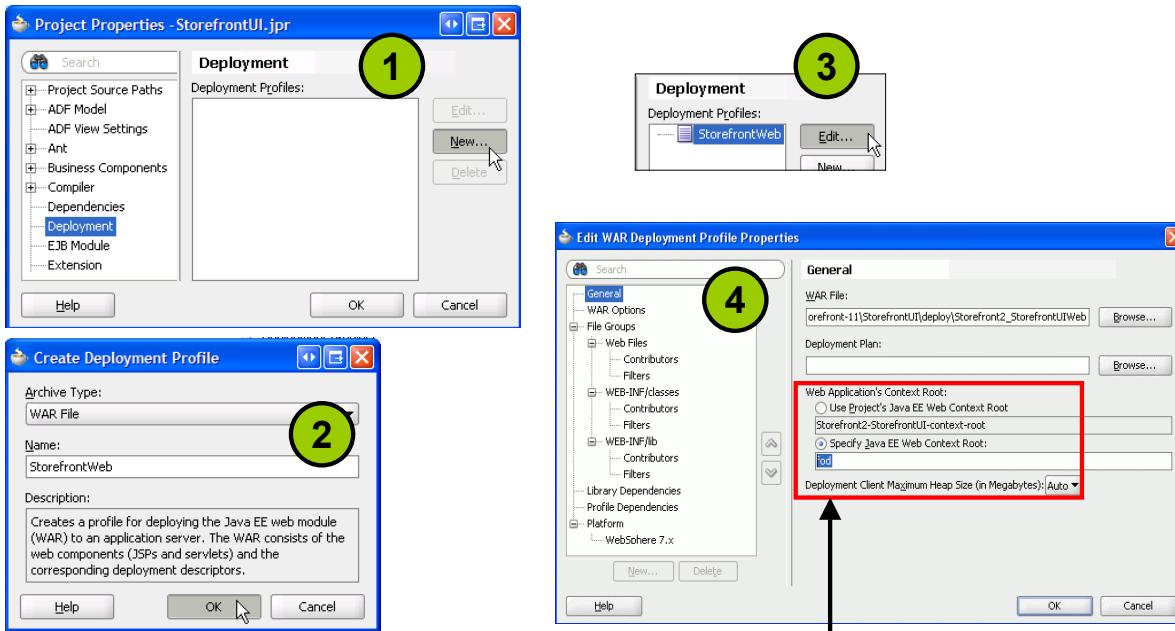


Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Typical Web Application Deployment Example

The following slides illustrate the process of deploying a Web application to an application server.

Example: Creating a WAR Deployment Profile for a UI Project



Context root becomes part of the URL to access the Web application.

ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a WAR Deployment Profile for a UI Project

You create a new deployment profile for a UI project by selecting Deployment Profiles and WAR File from General category of the New Gallery, or by modifying project properties and creating a new deployment profile. After you give the profile a name, JDeveloper launches the WAR Deployment Profile Properties dialog box.

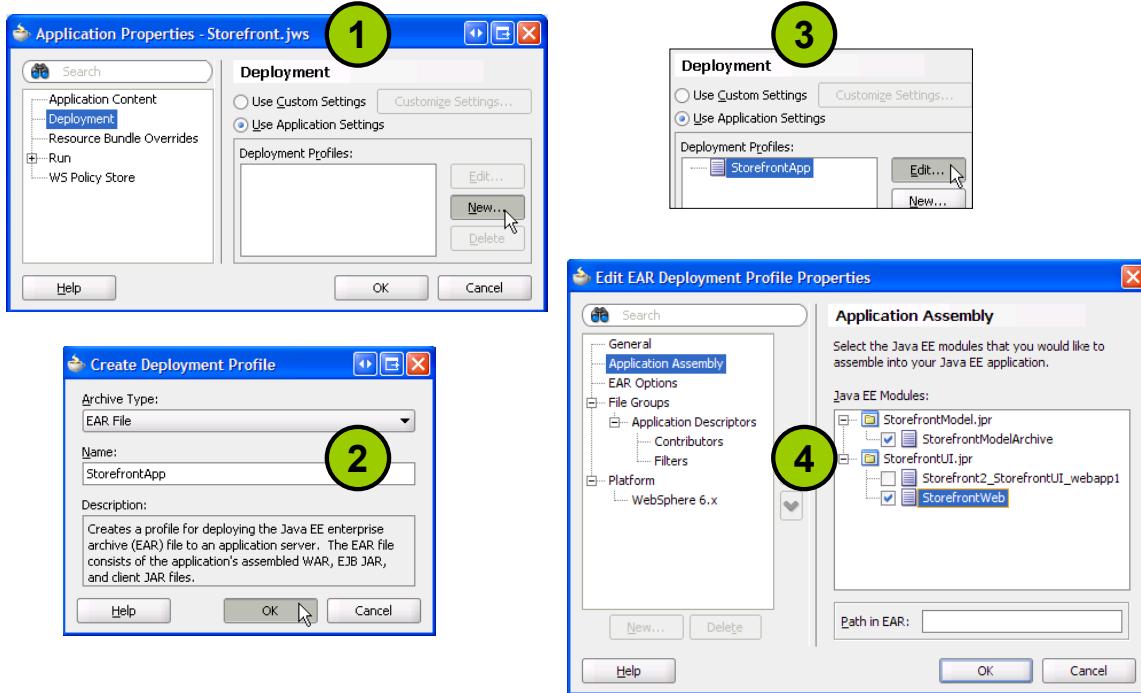
The default setting for the profile selects all the client JSP code, in the example, and all the ADF BC components. When deployed, the profile creates both .war and .ear files. The .war file contains the JSPs and the ADF BC components, whereas the .ear file contains the deployment descriptor files.

You should specify a context root for the application, because the one supplied by default is rather lengthy, and the component root becomes part of the URL for accessing the application.

To create a WAR deployment profile, perform the following steps:

1. For the project that contains your user interface, invoke the Project Properties dialog box, select Deployment, and click New (or select the project and use the New Gallery > General > Deployment Profiles > WAR File).
2. Select the archive type (WAR) and give the profile a name.
3. Edit the profile that you just created.
4. Specify a short and descriptive Web Context Root.

Example: Creating an EAR Deployment Profile for the Application



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating an EAR Deployment Profile for an Application

To create the application-level EAR deployment profile, perform the following steps:

1. Invoke the Application Properties dialog box, select Deployment, and click New (or from the Application menu, select New and use the New Gallery > General > Deployment Profiles > EAR File).
2. Choose the archive type (EAR) and give the profile a name.
3. Edit the profile that you just created.
4. In the EAR Deployment Profile Properties dialog box, there are several options that you can specify for the EAR file. At a minimum, you should select the Application Assembly node from the tree, and then select the modules to include in the EAR. The example in the slide shows the selection of the WAR file that was previously created for the UI project, but you typically would be deploying multiple modules that compose an application.

Using Deployment Descriptors

Deployment descriptors are:

- Project configuration files deployed with the application
- XML files of several types depending on project technologies and deployment target
- Created in various ways:
 - By JDeveloper wizards
 - As XML source files
 - By using the New Gallery



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Deployment Descriptors

Deployment descriptors are configuration files associated with projects and deployed with Java EE applications and modules. Deployment descriptors contain the declarative data required to deploy the components as well as the assembly instructions that describe how the components are composed into an application. Projects require different deployment descriptors depending on the technologies the project uses and on the type of the target application server. In addition to the standard Java EE deployment descriptors such as `application.xml` and `web.xml`, you can also have deployment descriptors that are specific to your target application server. For example, if you are deploying to Oracle WebLogic Server, you can also have `weblogic.xml` and `weblogic-application.xml`.

JDeveloper provides dialog boxes to inspect and set the properties of many deployment descriptors. You also can create and edit them as XML source files. The essential descriptors are created by the wizards that create deployment profiles. Add other descriptors only if you want to override default behavior. New deployment descriptors are placed in a `META-INF` subfolder of a project's Application Sources or Web Contents folders.

Warning: Avoid creating any deployment descriptors that apply to EAR files for more than one project in an application or a workspace. These files are assigned to projects, but have workspace scope. If multiple projects in an application or workspace have the same deployment descriptor, the one belonging to the launched project will supersede the others. This restriction applies to `application.xml`, `weblogic-jdbc.xml`, `jazn-data.xml`, and `weblogic.xml`.

Steps in the Deployment Process

Deployment tasks:

- Prepare application or project for deployment.
- Prepare Oracle WebLogic Server (WLS) for ADF deployment.
- Deploy the application with one of the following techniques:
 - Deploy directly from JDeveloper to WLS.
 - Deploy to an EAR file and use application server tool for deploying to the application server.
 - Use a script to deploy.

 ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Preparing the Oracle WebLogic Server

To deploy an application to Oracle WebLogic Server:

- Add the ADF Runtime to the WebLogic installation
- Create and configure the WebLogic domain
- Create a JDBC data source

Additionally, you must configure the application to use the WLS data source that you created.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Preparing the Oracle WebLogic Server

There are three basic steps that must be performed to prepare the WebLogic Server for an ADF application. Subsequent slides describe these steps.

Installing the ADF Runtime to the WebLogic Installation

- If you need to, use the Oracle Installer to add ADF Runtime to the existing WebLogic installation.
- Not needed if you:
 - Are using integrated WLS
 - Have installed WLS with the JDeveloper installer and selected the ADF Runtime option when installing



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Installing the ADF Runtime to the WebLogic Installation

Oracle WebLogic Server requires the ADF Runtime to run Oracle ADF applications. Installing the ADF Runtime is not required if you are using JDeveloper to run applications in the Integrated WLS. If you installed the Oracle WebLogic Server 10gR3 together with JDeveloper 11g by using the Oracle JDeveloper 11g Installer with the Application Development Framework Runtime check box selected, then the ADF Runtime is already installed.

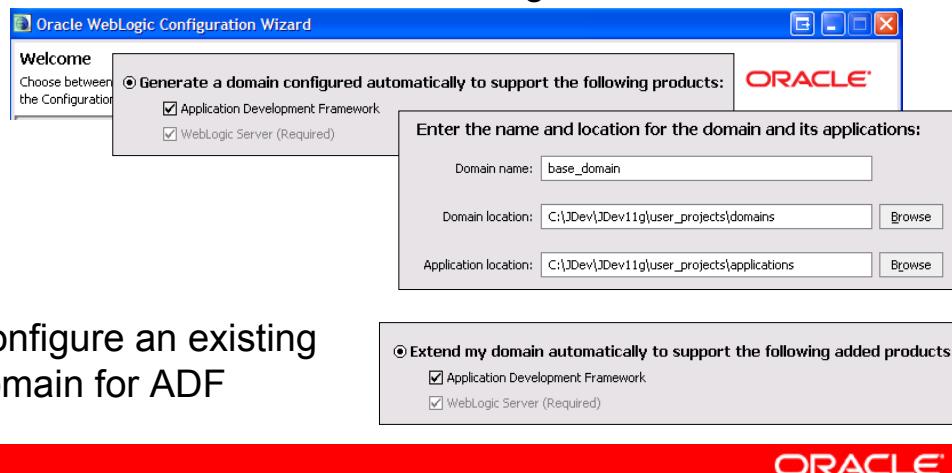
To install the ADF Runtime into an existing WebLogic Server, perform the following steps:

1. Start the Oracle JDeveloper 11g Installer as described in the installation guide. You can download the installer from Oracle Technology Network (OTN).
2. On the Choose Middleware Home directory page, select “Use an existing Middleware Home,” select the directory in which your Oracle WebLogic Server 10gR3 resides, and click Next.
3. On the “Choose Products and Components” page, choose Application Development Framework Runtime, deselect JDeveloper Studio if you do not want to install the JDeveloper IDE, and click Next. The Oracle WebLogic Server product and components should be disabled. If it is enabled and selected, you must check whether you have selected the correct Oracle WebLogic Server installation directory.
4. Follow the instructions in the *Oracle JDeveloper 11g Installation Guide* to complete the installation.
5. When the installation is complete, click Done.

Creating and Configuring the WebLogic Domain

- You need to have a WebLogic domain that is configured to accept ADF applications.
- Use the Oracle WebLogic Configuration Wizard to:
 - Create a new domain that is configured for ADF

OR



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating and Configuring a WebLogic Domain

You must create and configure a WebLogic domain to accept ADF applications. If you do not already have a domain, you must create a new domain. If you already have a domain, you must extend the domain before it can run ADF applications.

To create a new WebLogic domain configured for ADF, perform the following steps:

1. Start the configuration by selecting Oracle WebLogic > WebLogic Server 10.3 > Tools > Configuration Wizard from the Windows Start menu.
2. On the Welcome page, select “Create a New WebLogic Domain” and click Next.
3. On the Select Domain Source page, select “Generate a domain configured automatically to support the following products:”, and then select Application Development Framework and click Next. The WebLogic Server (Required) option is already selected.
4. In the Configure Administrator Username and Password page, enter the username and password, and then click Next.
5. On the Configure Server Start Mode and JDK page, select whether the domain is for a development or a production system, select the JDK to use, and click Next.
6. On the Customize Environment and Services Setting page, select No when asked to customize further domain options and click Next.
7. On the Create WebLogic Domain page, enter the name and location of the new domain and click Create.
8. Click Done when the creation process has finished.

Creating and Configuring a WebLogic Domain (continued)

If you are extending an existing WebLogic domain for Oracle ADF, rather than creating a new domain, you must perform the following steps:

1. Start the Configuration Wizard by selecting Oracle WebLogic > WebLogic Server 10.3 > Tools > Configuration Wizard from the Windows Start menu.
2. On the Welcome page, select “Extend an existing WebLogic domain” and click Next.
3. On the “Select a WebLogic Domain Directory” page, select the location of the domain that you want to configure for ADF, and click Next.
4. On the Select Extension Source page, select “Extend my domain automatically to support the following added products:” and then select Application Development Framework and click Next.
5. On the “Customize JDBC and JMS File Store Settings” page, leave No selected to update the domain as specified in the template and click Next.
6. On the Extend WebLogic Domain page, enter the domain name and location, and then click Extend.
7. Click Done when the configuration has finished.
8. After you have extended the domain, check the POST_CLASSPATH entry to make sure that setDomainEnv is set to the correct location, which should be
`ORACLE_HOME\jdeveloper\modules\features\adf.share_11.1.1.jar.`
This configures the rest of the run-time .jar files using the manifest file.

Note: Your application’s EAR file must have a `weblogic-application.xml` file containing a reference to the `adf.oracle.domain` shared library.

You can now start the Oracle WebLogic Server by running the command-line script
`<ORACLE_HOME>\user_projects\domains\domain_name\bin\startWebLogic.cmd` and stop the server using the `stopWebLogic.cmd` script in the same directory.

Access the Oracle WebLogic Server Administration Console by using the URL
`http://localhost:7001/console`.

Creating a JDBC Data Source

The screenshot shows the Oracle WebLogic Server Administration Console interface. It is divided into several sections:

- Top Left:** Shows the "ORACLE® WebLogic Server® Administration Console" logo and a "Welcome" message.
- Top Right:** A login form with fields for "Username" (weblogic) and "Password" (*****).
- Middle Left:** A "Domain Structure" tree view under "base_domain". The "Data Sources" node is highlighted with a red box and circled with a green number 2.
- Middle Right:** A "Summary" section showing a summary of the JDBC data source configuration.
- Bottom:** A red footer bar with the text "Copyright © 2010, Oracle and/or its affiliates. All rights reserved." and the "ORACLE" logo.

Four numbered circles (1, 2, 3, 4) highlight specific areas of the interface:

- Circle 1:** Points to the "Welcome" message at the top of the console.
- Circle 2:** Points to the "Data Sources" node in the Domain Structure tree.
- Circle 3:** Points to the "Name:" field in the "Create a New JDBC Data Source" wizard, which contains the value "FODlocal".
- Circle 4:** Points to the "Next" button in the wizard, which is highlighted with a mouse cursor.

Creating a JDBC Data Source

To set up a data source in the WebLogic server, perform the following steps:

1. With the server started, invoke the WebLogic Server Administration Console:
 - Enter the URL for the administration console, such as `http://localhost:7101/console`, or use the Start menu.
 - When the console application is deployed, enter credentials, such as `weblogic/weblogic`, and click Log In.
2. Invoke the Data Sources Wizard:
 - In the Domain Structure section at the left of the page, click Services > JDBC > Data Sources.
 - In the Summary of JDBC Data Sources section of the page, click New.

Creating a JDBC Data Source (continued)

3. Define the new data source:
 - On the JDBC Data Source Properties page of the wizard:

| | |
|------------------------|---|
| Name | Any name, such as FODLocal, to denote that you are using the FOD schema and a local database |
| JNDI Name | <code>jdbc/<connection name>DS</code> : This is the name that you will use to configure the application module, such as <code>jdbc/FODDS</code> . |
| Database Type | Oracle |
| Database Driver | Oracle's Driver (Thin) Versions: 9.0.1, 9.2.0, 10, 11 |

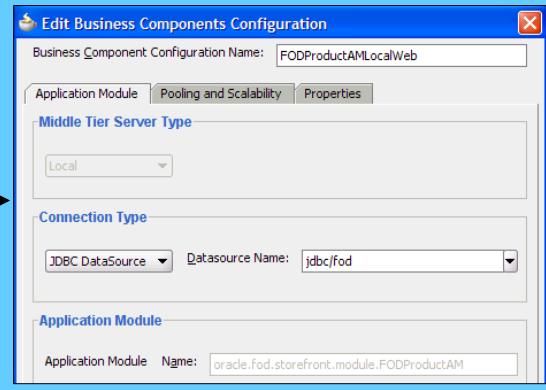
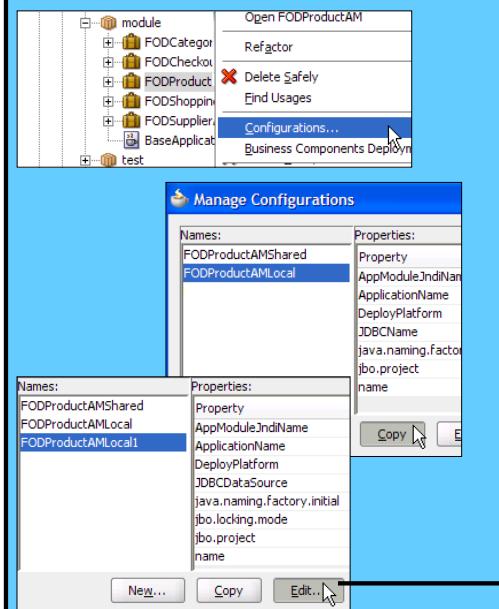
 - On the Transaction Options page of the wizard, accept the defaults.
 - On the Connection Properties page of the wizard:

| | |
|--|---|
| Database Name | SID of the database in your JDev connection, such as XE |
| Host Name | Host of the database in your JDev connection, such as localhost |
| Port | Port of the database in your JDev connection, such as 1521 |
| Database User Name and Password | Username and password of the schema in your JDev connection, such as fod/fusion |

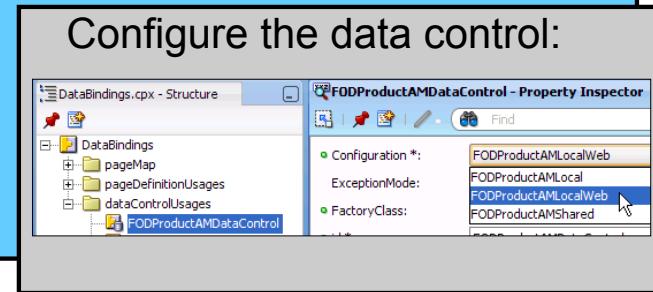
 - On the Test Database Connection page of the wizard, review the values and click Test Configuration, which should return a message “Connection test succeeded.” Click Next. (Do not click Finish until you complete the next page.)
 - On the Select Targets page of the wizard, select the DefaultServer check box and click Finish.
4. So that the configuration takes effect, stop and then restart the WLS Server instance by using the Start menu (or JDeveloper if stopping the integrated server).

Configuring the Data Control to Use the Data Source

Define a new AM configuration:



Configure the data control:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Configuring the Data Control to Use the Data Source

Each application module that is used as a data control in the application must use the WLS data source when running a page in a browser. To configure an application module to use the data source, perform the following steps:

1. In the Application Navigator, right-click the application module and select Configurations.
2. In the Manage Configurations dialog box:
 - a. Select the Local configuration and click Copy.
 - b. Select the new configuration and click Edit.
 - c. In the Edit Business Components dialog box:
Set the following properties on the Application Module tab:

| | |
|--|---|
| Business Component Configuration Name | Any unique name, such as FODProductAMLocalWeb |
| Connection Type | JDBC DataSource |
| Datasource Name | The JNDI name of your WLS data source, such as jdbc/fod |

On the Properties tab, set jbo.locking.mode to optimistic, the preferred locking mode for Web applications.

Configuring the Data Control to Use the Data Source (continued)

Next you set the data control to use the new configuration:

1. In the Application Navigator, select DataBindings.cpx.
2. In the Structure window, expand dataControlUsages and select the data control.
3. In the Property Inspector, select the new configuration from the Configuration drop-down list.

Steps in the Deployment Process

Deployment tasks:

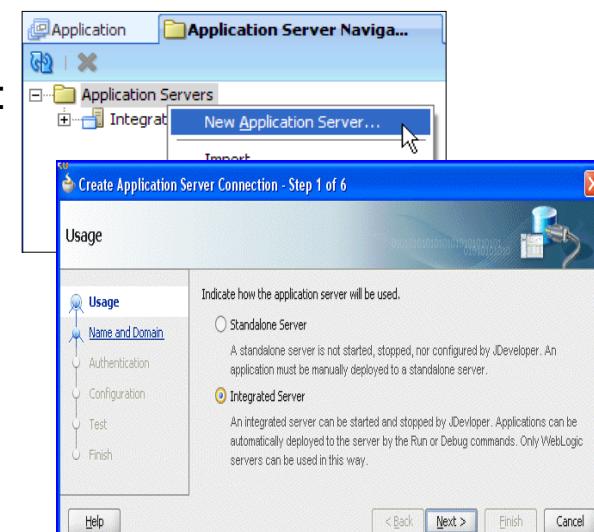
- Prepare application or project for deployment.
- Prepare Oracle WebLogic Server (WLS) for ADF deployment.
- Deploy the application with one of the following techniques:
 - Deploy directly from JDeveloper to WLS.
 - Deploy to an EAR file and use application server tool for deploying to the application server.
 - Use a script to deploy.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Connection to an Application Server

- Start the Application Server Connection Wizard: Right-click IDE Connections > New Application Server Connection.
- On the Usage page, indicate whether the connection is for:
 - A stand-alone server
 - An integrated WebLogic server



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Connection to an Application Server

Before you can use the JDeveloper built-in deployment utility, you must create a connection object. The deployment wizards use this connection to build the deployment archives and physically copy the deployment files to the target environment.

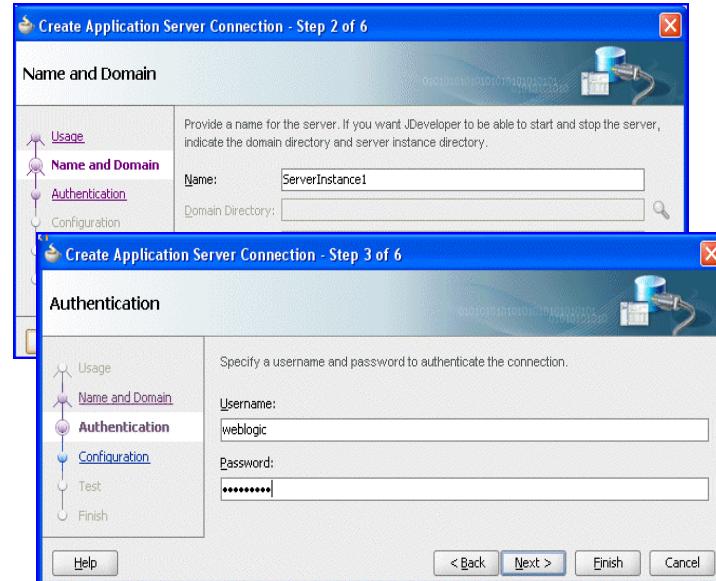
You start the process by right-clicking the IDE Connections node in the Application Server Navigator (View > Application Server Navigator) or in the Resource Palette and selecting New Application Server Connection. This launches the Create Application Server Connection Wizard. You can also launch the wizard from the New Gallery.

The first page of the wizard is the Usage page. You use this page to determine whether the connection is to a stand-alone server or the integrated WebLogic Server.

- A stand-alone server is not started, stopped, or configured by JDeveloper. An application must be manually deployed to a stand-alone server.
- The integrated WebLogic Server can be started and stopped by JDeveloper. Applications can be automatically deployed to the server by the Run or Debug command. Only Oracle WebLogic Server instances can be used in this way.

Creating a Connection to an Application Server

- On subsequent pages of the wizard, enter the following:
 - Connection name
(for a stand-alone server you also specify the type of connection)
 - Username and password
 - Host name
 - Port and SSL Port
 - WebLogic Domain
- Test the connection.



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating a Connection to an Application Server

The next page of the wizard requires you to provide a name for the server connection. If you select the “Let JDeveloper manage the lifecycle for this Server Instance” check box, you can start the Integrated Server in Run and Debug mode, and launch the Administration Console from the Application Server Navigator. However, in this case, you also need to specify a domain directory and a server instance.

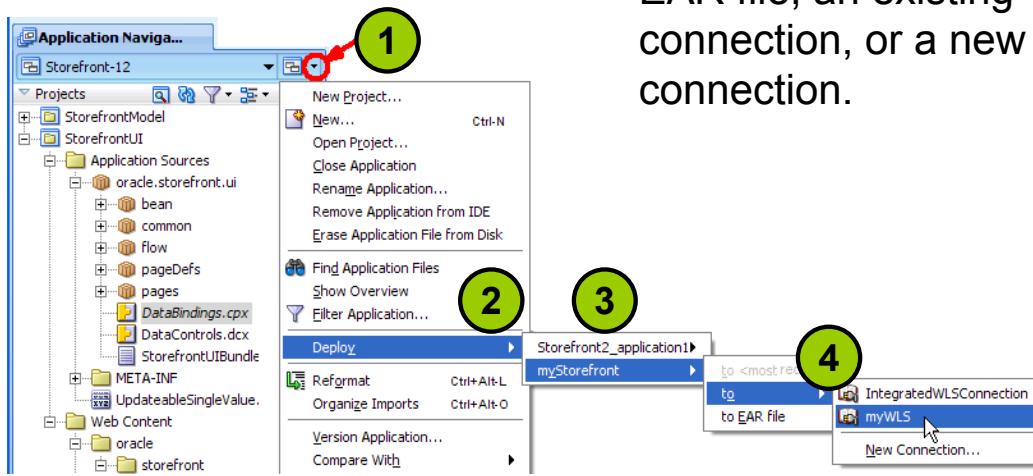
Alternatively, if you selected the stand-alone server option on the previous page, you need to provide a connection type as well as a name. The available types are WebLogic 10.3, JBoss 5.x, Tomcat 6.x, and WebSphere Server 7.x.

On the Authentication page, enter the administrative username and password for the application server and on the Configuration page, the host name for the server, the port and SSL port numbers, and the WebLogic domain.

Before leaving the wizard, be sure to visit the Test page of the wizard and click Test to ensure that the connection functions correctly. The application server must be running in order for the connection to be successful.

Example: Deploying the Application

1. Invoke the Application menu.
2. Select Deploy.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Deploying the Application

Now that you have created an application deployment profile, the rest of the process is almost automatic.

To actually deploy the application, right-click the deployment profile, click Deploy to, and select the connection you want to use. JDeveloper now uses the connection information to copy the required deployment files to the target environment.

Alternatively, you could deploy to an EAR file, and then use the tools of the application server to deploy the EAR file.

Steps in the Deployment Process

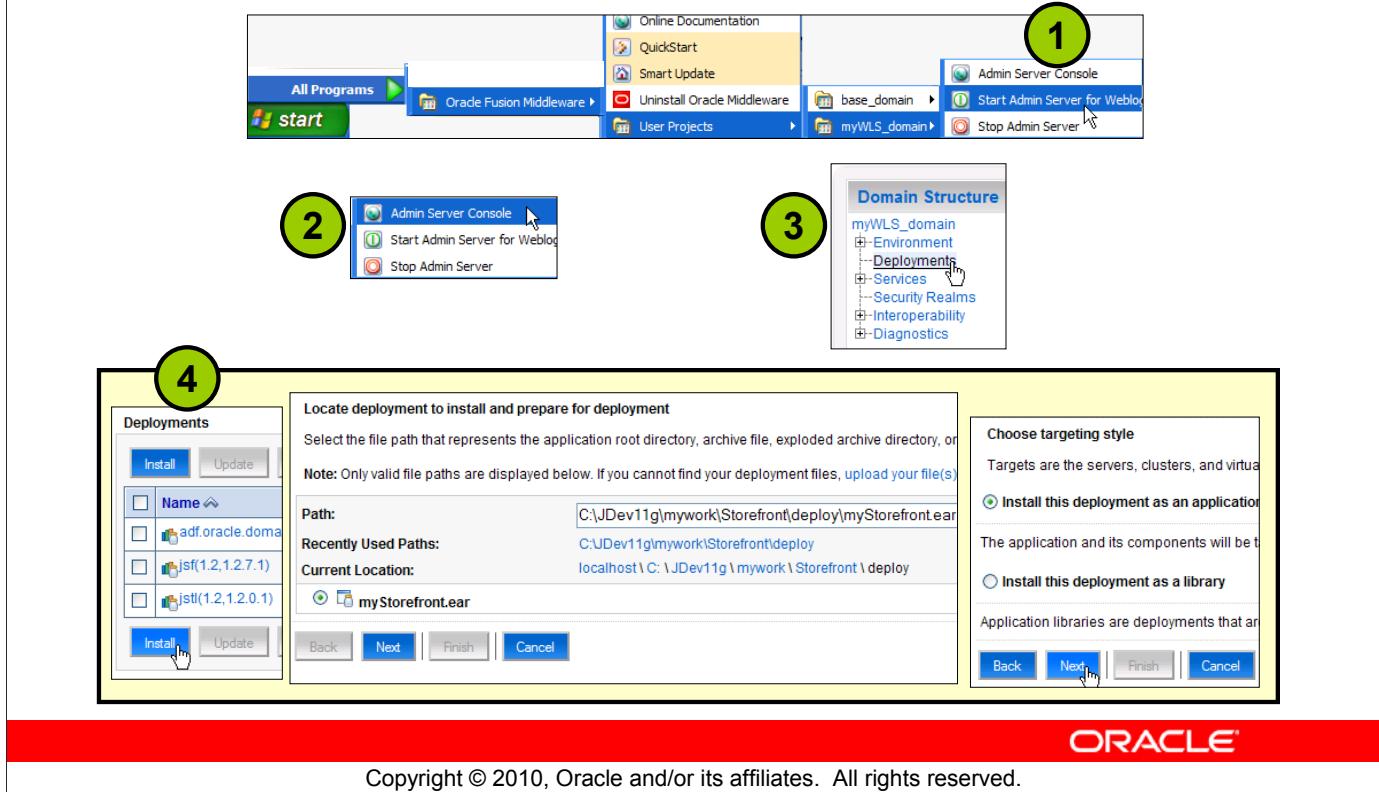
Deployment tasks:

- Prepare application or project for deployment.
- Prepare Oracle WebLogic Server (WLS) for ADF deployment.
- Deploy the application with one of the following techniques:
 - Deploy directly from JDeveloper to WLS.
 - Deploy to an EAR file and use application server tool for deploying to the application server.
 - Use a script to deploy.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Deploying the Application from the WebLogic Administration Server Console



Deploying the Application from the WebLogic Administration Server Console

To deploy on Oracle WebLogic Server, perform the following steps:

1. Start the WebLogic Server instance.
2. Invoke the Administration Console.
3. Navigate to the Deployments panel.
4. Install the EAR that you created in JDeveloper.

Steps in the Deployment Process

Deployment tasks:

- Prepare application or project for deployment.
- Prepare Oracle WebLogic Server (WLS) for ADF deployment.
- Deploy the application with one of the following techniques:
 - Deploy directly from JDeveloper to WLS.
 - Deploy to an EAR file and use application server tool for deploying to the application server.
 - Use a script to deploy.



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Using Ant to Automate the Deployment Process

- Ant is an XML scripting tool that enables you to build and deploy Java EE applications.
 - Define named series of tasks (called targets).
 - Define dependencies between targets.
- Following are the advantages of Ant:
 - Well-documented
 - Widely adopted
 - Easy to configure
 - Extensible
- You can get more information about Ant at:
<http://ant.apache.org>



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Deploying with Ant

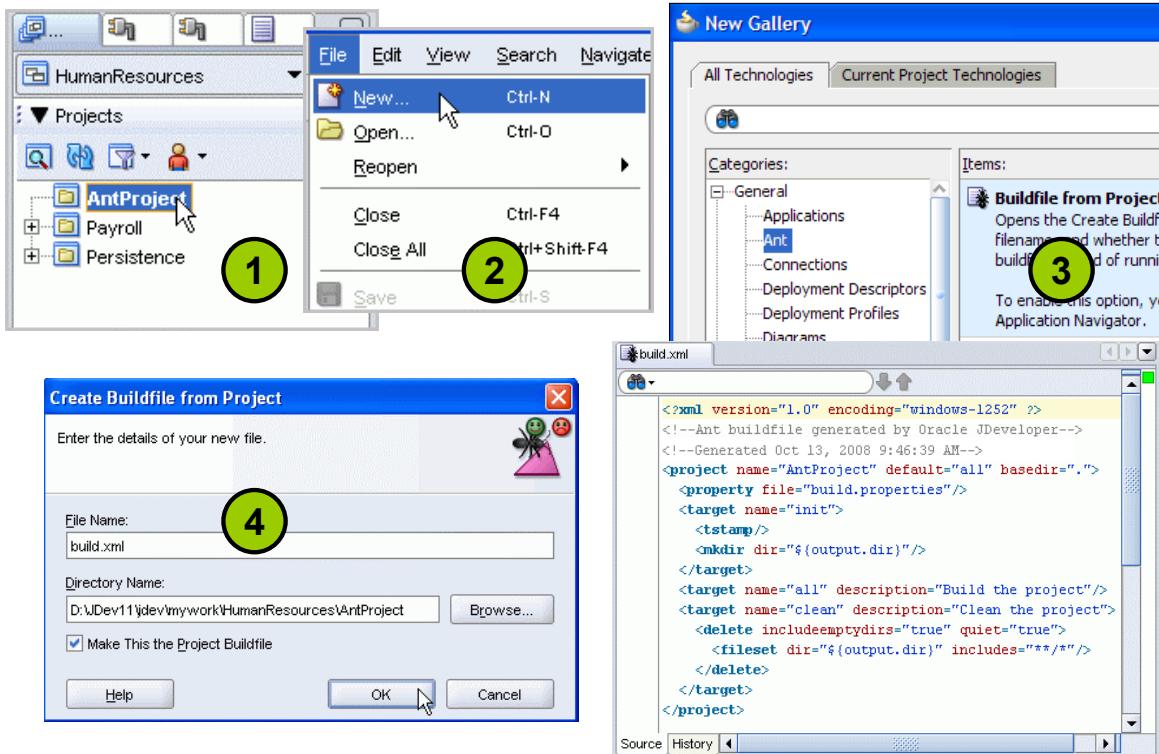
The Java-based Ant tool is a programmatic approach for building and deploying Java EE applications. It offers many advantages, such as an abundance of documentation and expertise, wide adoption of the tool, and the relative ease with which developers can configure this tool to help them automate the build-and-deploy cycle. In addition, because Ant is written in Java, developers can extend the functionality by writing their own custom tasks. Ant has built-in support for CVS, JUnit, and other tasks so that you can automate the entire build, deployment, and unit test process.

An Ant buildfile defines targets and dependencies between targets. A target is a sequence of programmatic tasks. When Ant is run to make a target, it first makes other targets on which it depends, and then executes the target's own tasks.

Oracle JDeveloper supports creating and running Ant tasks. Ant buildfiles can be added to or created for projects. Ant buildfiles can be edited with JDeveloper's XML Source Editor. Ant can be invoked from the user interface to make targets defined in buildfiles.

The buildfile may vary depending on the target application server. For deployment to other application servers, see the application server's documentation. If your application server does not provide specific Ant tasks, you may be able to use generic Ant tasks. For example, the generic `ear` task creates an EAR file for you.

Creating an Ant Buildfile in JDeveloper



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Creating an Ant Buildfile

Ant buildfiles are XML files, and as such can be maintained by using JDeveloper XML editing tools. Buildfiles use a target tree to define various tasks. A target is a set of tasks to be executed, such as compiling or packaging into JAR files for distribution. Targets may have dependencies on other targets as defined in the tree.

To create a buildfile, perform the following steps:

1. Select a project in the Applications Navigator.
2. Select File > New from the menu.
3. In the New Gallery, expand the General category and select Ant, and then select “Buildfile from Project” in the items list. Click OK.
4. In the “Create Buildfile from Project” dialog box, specify the file name and directory, and optionally indicate whether you want this to be the project buildfile. Click OK.
5. The buildfile opens in the editor with some default targets, along with a Component Palette of Ant-related elements that you can add to the XML file.

Defining Ant Deployment Tasks

Tasks for handling archive files:

- <jar>
- <ejbjar>
- <ear>
- <war>
- <unjar>
- <unwar>

Tasks for local server deployment:

- <copy>
- <delete>
- <java>

Tasks for remote deployment:

- <ftp>
- <condition>
- <sleep>
- <get>



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Defining Ant Deployment Tasks

The set of core Ant tasks that you can find in the Component Palette includes several that you can use in the deployment process. For example, to create and use archive files, you can employ the tasks <jar>, <ejbjar>, <ear>, <war>, <unjar>, and <unwar>.

You use the following tasks for local server deployment:

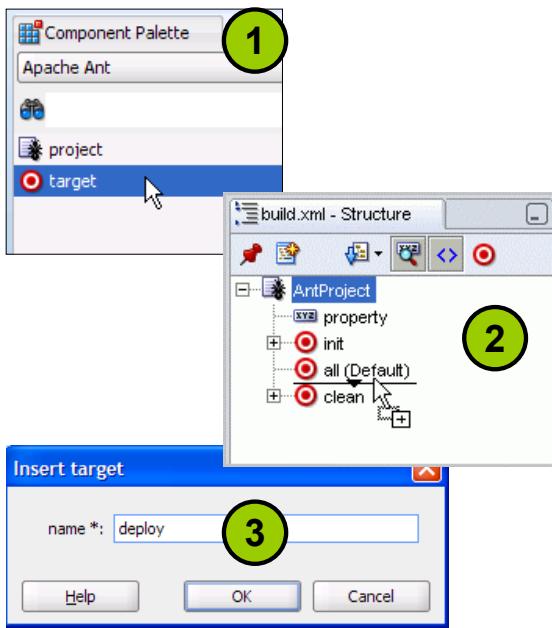
- <copy>: Copies a file or FileSet (group of files) to a new file or directory
- <delete>: Deletes a single file, a specified directory and all its files and subdirectories, or a set of files specified by one or more FileSets
- <java>: Executes a Java class within the running (Ant) VM or forks another VM if specified

The following core Ant tasks are useful for remote deployment:

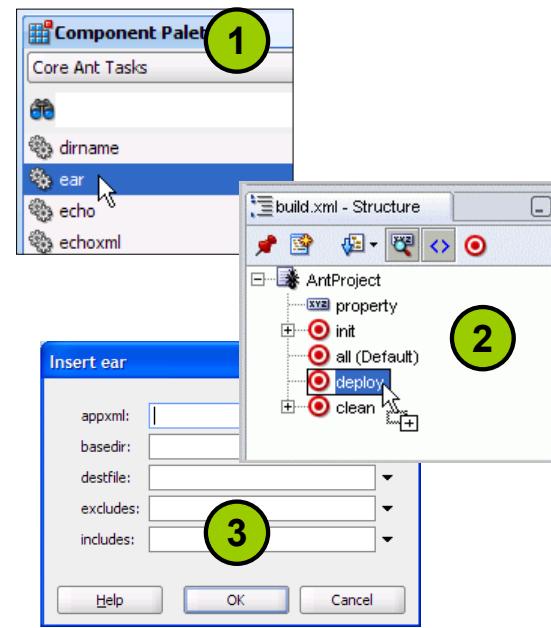
- <ftp>: Implements a basic FTP client that can send, receive, list, and delete files, and create directories
- <condition>: Sets a property to true by default if a certain condition holds true; otherwise, the property is not set. You can set the value to something other than the default by specifying the value attribute.
- <sleep>: A task for pausing for a short period of time. It is useful when a build or deployment process requires an interval between tasks.
- <waitfor>: Blocks execution until a set of specified conditions becomes true
- <get>: Gets a file from a URL

Adding Elements to the Buildfile

Adding a target:



Adding a task:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Adding Elements to the Buildfile

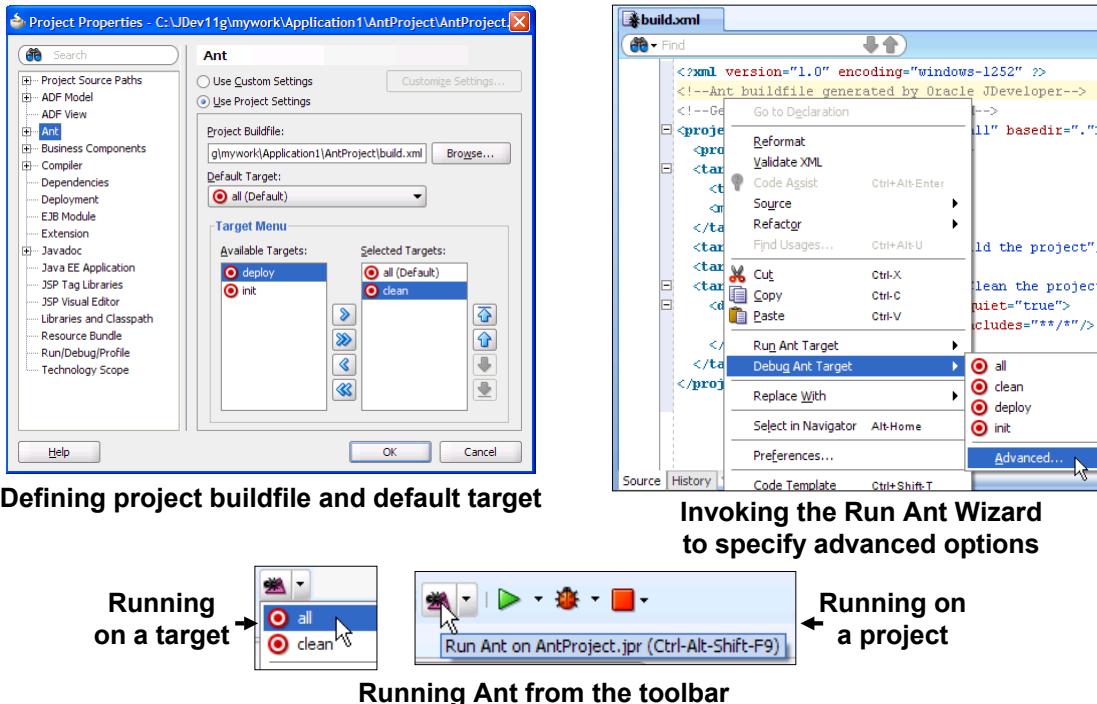
With the buildfile open in the editor, you can add elements to it by dragging them from the Component Palette to the Structure window. For example, to add a target to the buildfile, perform the following steps:

1. In the Component Palette, select Apache Ant from the drop-down list and click target.
2. Drag target to the desired location in the Structure window.
3. In the Insert target dialog box, name the target, and then click OK. You can optionally define advanced properties, such as dependencies, in the Property Inspector.

To add tasks to a target, perform the following steps:

1. In the Component Palette, select the desired task type, such as Core Ant Tasks, from the drop-down list and click the desired task, such as copy.
2. Drag the task to the desired target in the Structure window.
3. In the insert dialog box (if there is one) or in the Property Inspector, define the attributes of the task as needed.

Running Ant on Buildfile Targets



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

Running Ant on Buildfile Targets

You can run Ant on buildfile targets in one of the following ways:

- On targets in the project buildfile: A project can contain several Ant buildfiles, but one can be designated as the project buildfile. You can configure the Run Ant toolbar button and drop-down menu to give easy access to the project buildfile's targets.
- From the Structure window when editing an Ant buildfile: When an Ant buildfile is open in an XML source editor, its targets are listed in the Structure window and can be run.
- From external tools that you define: Use the Create External Tool Wizard to define menu items and toolbar buttons that make Ant targets. The projects that external tools act on need not be open in JDeveloper.

Running Ant on Project Buildfile Targets

To select and configure a project buildfile, go to the Ant project properties page (choose Tools > Project Properties).

You can run Ant on targets in the project buildfile in one of the following ways:

- From the toolbar Run Ant drop-down menu, select a target.
- From the toolbar, click Run Ant without invoking the drop-down menu.
- From the main menu, select Build > Run Ant on <Project>.

If you use one of the last two methods, Ant makes (compiles) the project's designated default target.

Running Ant on Buildfile Targets (continued)

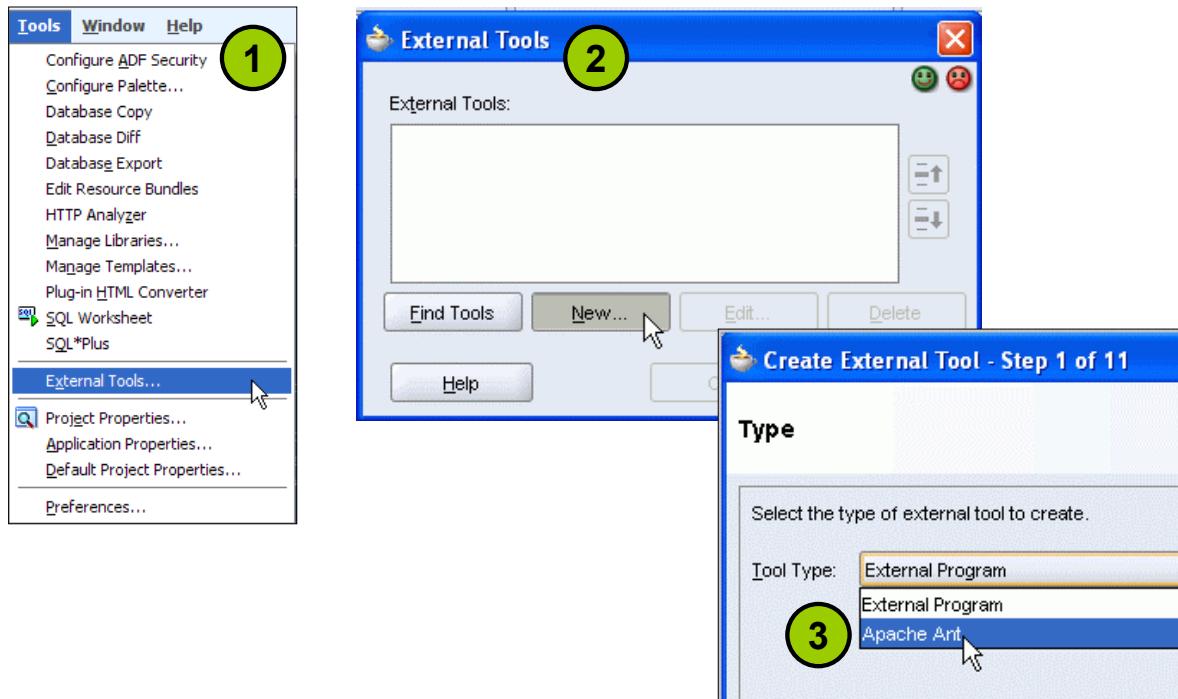
Running Ant from the Structure Window

From the Structure window, you can run Ant by using either the project configuration or a distinct buildfile configuration. The buildfile configuration is derived initially from the project configuration, and persists with the buildfile from session to session.

You can invoke a target from the Structure window in one of the following ways:

- Run Ant using the project configuration.
 1. Open an Ant buildfile or give focus to a buildfile that is already open. The Structure window displays the buildfile's elements.
 2. Right-click a target element, and select Run Target <*target*>. Ant runs to make the target, and the log window displays the results.
- Run Ant using a buildfile-specific configuration.
 1. Open an Ant buildfile or give focus to a buildfile that is already open. The Structure window displays the buildfile's elements.
 2. Right-click in the editor window and select Run Ant Target > <*target*> or select Run Ant Target > Advanced to open the Run Ant Wizard where you can configure advanced options.
 3. Ant runs to make (compile) the target, and the log window displays the results.

Creating an External Ant Tool



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

ORACLE

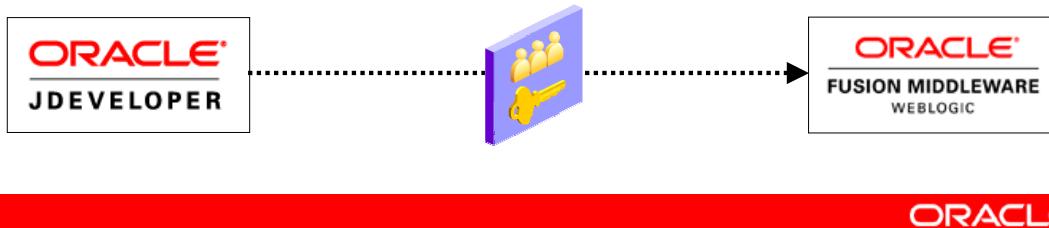
Creating an External Ant Tool

An external Ant tool enables you to build a project from a custom menu item or toolbar button without first opening the project in JDeveloper. To create an external Ant tool, perform the following steps:

1. From the main menu, select Tools > External Tools.
2. In the External Tools dialog box, click New to open the Create External Tool Wizard.
3. On the Type page of the wizard, select Apache Ant in the Tool Type drop-down menu and click Next.
4. On the remaining pages of the wizard, describe the Ant target, configure the build parameters, and select the UI items to represent the external Ant tool. Click Help on any page for more information.
5. On the last page of the wizard, click Finish to create the external Ant tool.

Implementing Security in Deployed Applications

- After deployment, you need to migrate to WLS the application's:
 - Database credential store
 - Security policies
- To help with this migration, you can:
 - Run the command-line script
 - OR
 - Use a simplified method with the supplied Ant script



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Implementing Security in Deployed Applications

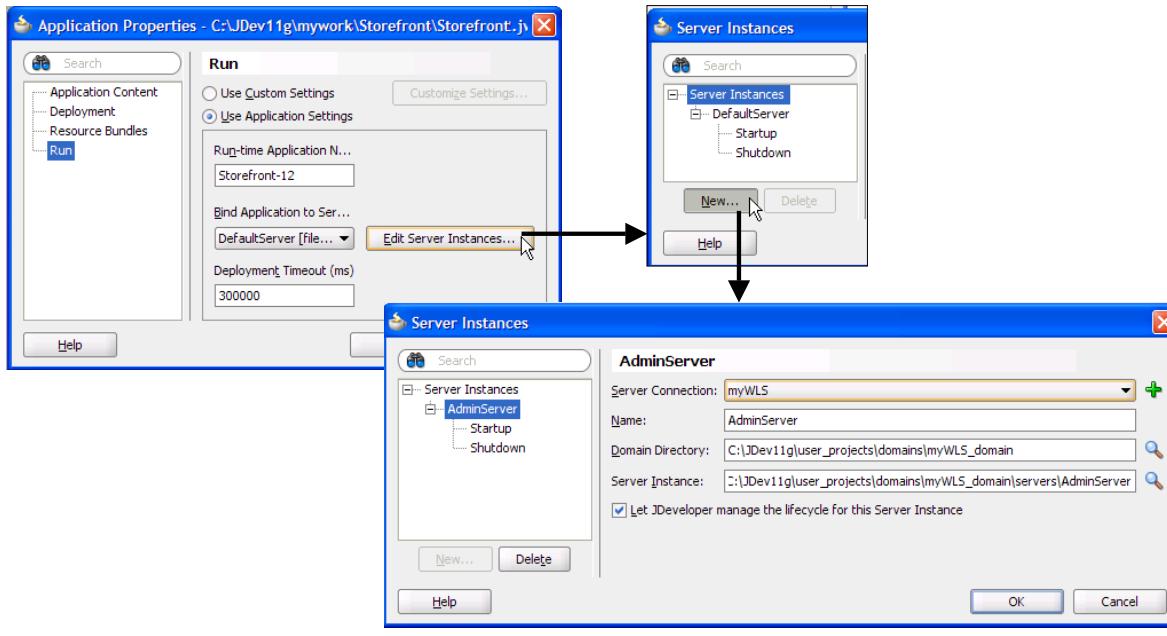
Although JDeveloper 11g migrates application identities to the integrated WebLogic Server for single-user testing and debugging, there is no automated migration of the identities to an external, stand-alone WLS. Typically the file-based identities in an application include only a few test usernames in the necessary roles for development testing purposes. The full set of usernames, roles, and role memberships needs to be defined on the stand-alone WebLogic Server as a separate step if these have not already been set up for other applications' environment.

After deploying to WLS, you must, therefore, migrate your application's credential store and any security policies outside of JDeveloper. There are some tools to assist you with this process:

- The command-line script
`<JDev_Home>/jdeveloper/modules/oracle.jps_11.1.1/scripts/migrateSecurityStore` can merge the credentials of your application with the existing data store, and can also merge application-level security policies with domain-level policies. Refer the JDeveloper online Help.
- For the most common use cases (migrating an application's credentials in `cwallet.sso` or the security policies in `jazn-data.xml` to the WLS domain's `cwallet.sso` or `system-jazn-data.xml` file), you can use a simplified method that uses an Ant script.

Deployment Testing During Development

Bind to an Oracle WebLogic Server instance:



Deployment Testing During Development

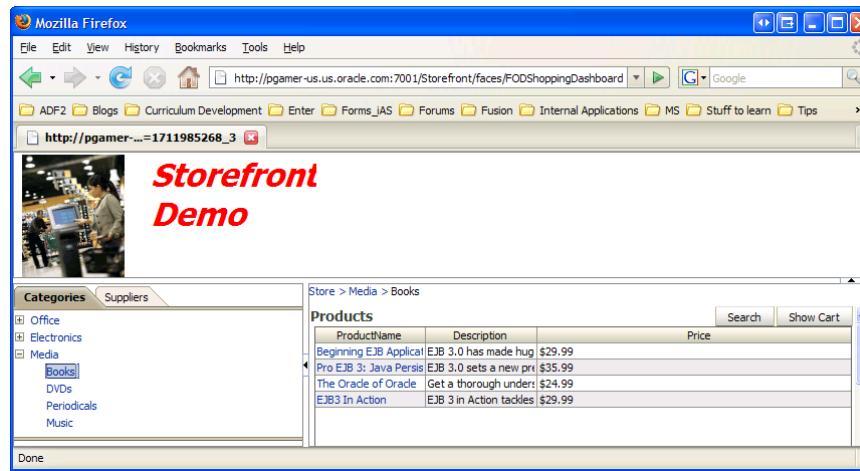
JDeveloper enables you to create an instance in any compatible Oracle WebLogic Server installation (which could be the integrated server) as a test environment to simulate the production server. This enables you to specify a full WebLogic Server instance to use as your primary run-time environment for development within JDeveloper. You can configure the environment with the services you are using, so that you can exactly simulate your production server for testing.

In JDeveloper, there is one Application Server instance that you can leave running, and each run operation deploys and runs the application in that Application Server instance. This mode allows for complete configuration of the server, and is recommended for developers who are building Service-Oriented Architecture (SOA) applications. At the same time, you retain the productivity features of the integrated server for application development, such as implicit starting/stopping, automatic deployment, in-place editing, and more.

You can configure WebLogic Server instances for development testing in the Run dialog box for Application Properties (Tools > Application Properties >Run).

Deployment Testing for Production

Use a URL for production testing:



ORACLE

Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Deployment Testing for Production

After you deploy the application, you can test it from the application server. To test run your application, open a browser window and enter a URL:

- For Oracle AS: `http://<host>:<port>/<context root>/<page>`
- For Faces pages: `http://<host>:<port>/<context root>/faces/<page>`

Note: /faces has to be in the URL for Faces pages. This is because JDeveloper configures your web.xml file to use the URL pattern of /faces to be associated with the Faces Servlet. The Faces Servlet does its per-request processing, strips out the /faces part in the URL, and then forwards to the JSP. If you do not include the /faces in the URL, the Faces Servlet is not engaged (because the URL pattern does not match). Your JSP is run without the necessary JSF per-request processing.

Summary

In this lesson, you should have learned how to:

- Create deployment profiles
- Configure deployment options
- Use JDeveloper to deploy an application
- Use the WebLogic Server Administration Console to deploy an application
- Use Ant to deploy an application
- Test the deployed application



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice Overview: Deploying the Web Application

This practice covers the following topics:

- Creating Deployment Profiles
- Using a WebLogic Server Data Source
- Creating an Application Server Connection
- Deploying the Application from JDeveloper
- Deploying the Application from the WLS Console
- Running the Deployed Application



Copyright © 2010, Oracle and/or its affiliates. All rights reserved.

Practice Overview: Deploying the Web Application

In this practice, you create a deployment profile and configure deployment options for the Storefront application that you have developed. You then configure a data source in the stand-alone Oracle WebLogic Server, and you configure the application to use a data source. Finally, you deploy the application to the stand-alone Oracle WebLogic Server, and you test the deployment.

This practice may be found in the Practices document in the Practices for Appendix B section.

THESE eKIT MATERIALS ARE FOR YOUR USE IN THIS CLASSROOM ONLY. COPYING eKIT MATERIALS FROM THIS COMPUTER IS STRICTLY PROHIBITED

Oracle University and Osi S.R.L. use only