Oracle® Fusion Applications

Extensibility Guide for Developers

11*g* Release 7 (11.1.7)

E41852-03

September 2013

Documentation for developers that describes how to use design-time tools to customize and extend the standard functionality provided by certain Oracle Fusion Applications.

ORACLE

Oracle Fusion Applications Extensibility Guide for Developers, 11*g* Release 7 (11.1.7)

E41852-03

Primary Author: Chris Kutler (lead), Shelly Butcher, Ralph Gordon, Peter Jew, Mark Kennedy, Steven Leslie, Landon Ott

Contents

Preface

Welcome to *Oracle Fusion Applications Extensibility Guide for Developers.*

Audience

This document is intended for developers who want to customize and extend the standard functionality provided by Oracle Fusion Applications. Developers should have a basic understanding of the Java programming language, web applications, Oracle JDeveloper, and Oracle Application Development Framework. This book gives an overview of the design-time customization and extension tasks and provides references to the books that contain more detailed documentation.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

You can also find information about Oracle Fusion Middleware and extending and customizing Oracle Fusion Applications in the following documents:

- *Oracle Database Security Guide*
- *Oracle Fusion Applications Administrator's Guide*
- *Oracle Fusion Applications Administrator's Troubleshooting Guide*
- *Oracle Fusion Applications Administrator and Implementor Roadmap*
- *Oracle Fusion Applications Common Implementation Guide*
- *Oracle Fusion Applications Concepts Guide*
- *Oracle Fusion Applications Extensibility Guide for Business Analysts*
- *Oracle Fusion Applications CRM Extensibility Guide*

- *Oracle Fusion Applications Enterprise Deployment Guide for Customer Relationship Management*
- *Oracle Fusion Applications Developer's Guide*
- *Oracle Fusion Functional Setup Manager User's Guide*
- *Oracle Fusion Functional Setup Manager Developer's Guide*
- *Oracle Fusion Applications Installation Guide*
- *Oracle Fusion Applications Post-Installation Guide*
- *Oracle Fusion Applications Master Glossary*
- *Oracle Fusion Applications Patching Guide*
- *Oracle Fusion Applications Security Guide*
- *Oracle Fusion Applications Security Hardening Guide*
- *Oracle Fusion Middleware Administrator's Guide*
- *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*
- *Oracle Fusion Middleware Application Security Guide*
- *Oracle Fusion Middleware Business Process Composer User's Guide for Oracle Business Process Management*
- *Oracle Fusion Middleware Developer's Guide for Oracle Enterprise Scheduler*
- *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*
- *Oracle Fusion Middleware Error Messages Reference*
- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*
- *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*
- *Oracle Fusion Middleware Developer's Guide for Oracle Business Intelligence Publisher (Oracle Fusion Applications Edition)*
- *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*
- *Oracle Fusion Middleware User's Guide for Oracle Business Rules*
- *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository*
- *Oracle Fusion Middleware User's Guide for Oracle Identity Manager*
- *Oracle Fusion Middleware User's Guide for Oracle WebCenter Portal: Spaces*
- *Oracle Fusion Middleware User's Guide for Technology Adapters*
- *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*

Conventions

The following text conventions are used in this document:

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

What's New in This Guide

The following topics introduce the new and changed design-time customization and extensibility features of Oracle Fusion Applications and other significant changes that are described in this guide, and provides pointers to additional information.

New and Changed Features for 11*g* Release 7 (11.1.7)

Oracle Fusion Applications 11*g* Release 7 (11.1.7) does not contain any new or changed features for this document.

Other Significant Changes in this Document for 11*g* Release 7 (11.1.7)

For 11*g* Release 7 (11.1.7), this guide has been updated in several ways. Following are the sections that have been added or changed.

- Design-time information for on-premises installations of Oracle Fusion Applications was moved from the *Oracle Fusion Applications Extensibility Guide for Business Analysts* to this book.
- Added customization guidelines for working in teams. See Section 3.2.4, "Avoiding Conflicts Among Multiple Customization Developers."
- Added guidelines for testing customized pages locally from Oracle JDeveloper. See Section 3.2.5, "Running Customizations Locally."
- Added instructions for implementing role-based customizations when customizing and extending Oracle Application Development Framework application artifacts. See Section 4.1.2, "Customizing at the Role Level."

Part I

Introduction to Customizing and Extending Oracle Fusion Applications

Part I contains the following chapters:

1

Customizing and Extending Oracle Fusion Applications

This chapter provides an overview of how to customize and extend on-premises installations of Oracle Fusion Applications and introduces the design time and runtime tools that are used in the process. The remainder of this book describes how to use the following tools to customize and extend on-premises installations at design time:

- Oracle JDeveloper
- Oracle Business Process Management Worklist (Oracle BPM Worklist)
- Oracle SOA Composer
- Oracle Enterprise Manager Fusion Applications Control (Fusion Applications Control)
- Oracle Application Development Framework (Oracle ADF) Skin Editor

This chapter includes the following sections:

1.1 Understanding Customizing and Extending Oracle Fusion Applications

While Oracle Fusion applications provide robust out-of-the-box functionality, there may be areas of one of the applications that you must change to meet your business needs. On-premises installations of Oracle Fusion Applications provide runtime and design time tools to customize and extend Oracle Fusion applications. This book gives an overview of both the runtime and design time tools, and then guides you through the process of using the design time tools that are available for on-premises installations. For further information about using runtime tools, see the *Oracle Fusion Applications CRM Extensibility Guide*, the *Oracle Fusion Applications Common Implementation Guide*, and the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

Most customizations made to an Oracle Fusion application, whether a personalization an end user makes, a change a business user makes using a runtime composer tool, or a change a developer makes using JDeveloper to create new source code, are stored in a metadata repository. Because these customizations are kept separate from the base

code, you can safely upgrade your Oracle Fusion application without losing your changes.

Customizations made at runtime can be saved in a **sandbox** so that the changes can be isolated and validated before being published into a full test environment. Changes done at design time are done in a development environment, and can also be deployed to a sandbox before being deployed into the full test environment.

The Manage Customizations dialog enables you to identify and examine where customizations have been made and for which layer, even when a page consists of several different **components** (some of them actually being another page). You can also use the Manage Customizations dialog to import customizations that others have done, or you can export your own customizations.

For more information about using the Manage Customizations dialog and sandboxes, see Chapter 2, "Understanding the Customization Development Lifecycle."

Note: You can also create a complete Java EE application to supplement your Oracle Fusion applications. See the *Oracle Fusion Applications Developer's Guide* for more information.

All Oracle Fusion applications are based on Oracle Fusion Middleware. Most user interfaces are implemented using Oracle Application Development Framework (Oracle ADF) and standard Java technologies, such as the JavaServer Faces technology. The foundation of the applications are the service-oriented architecture (SOA) business processes. Business intelligence frameworks provide several reporting capabilities. Identity management works at every level to control access. Each of these areas of an application can be customized and extended to suit your business needs.

Additionally, Oracle Fusion applications are built using a common data model. Because of this commonality, when you make a customization in one area, that customization will be available to all objects in the application. For example, if you add an attribute to an object, you can easily add that attribute to the web-based view page, to an associated mobile page, and to any associated reports.

Within this guide, the term **customize** means to change a standard (existing) artifact. For example, you can add an attribute to an existing object or you can change what is displayed on a standard page. The term **extend** means to create a completely new artifact, such as a custom object.

For customizations and extensions, there are three basic scenarios:

- Personalization
- Runtime customizations and extensions
- Design time customizations and extensions

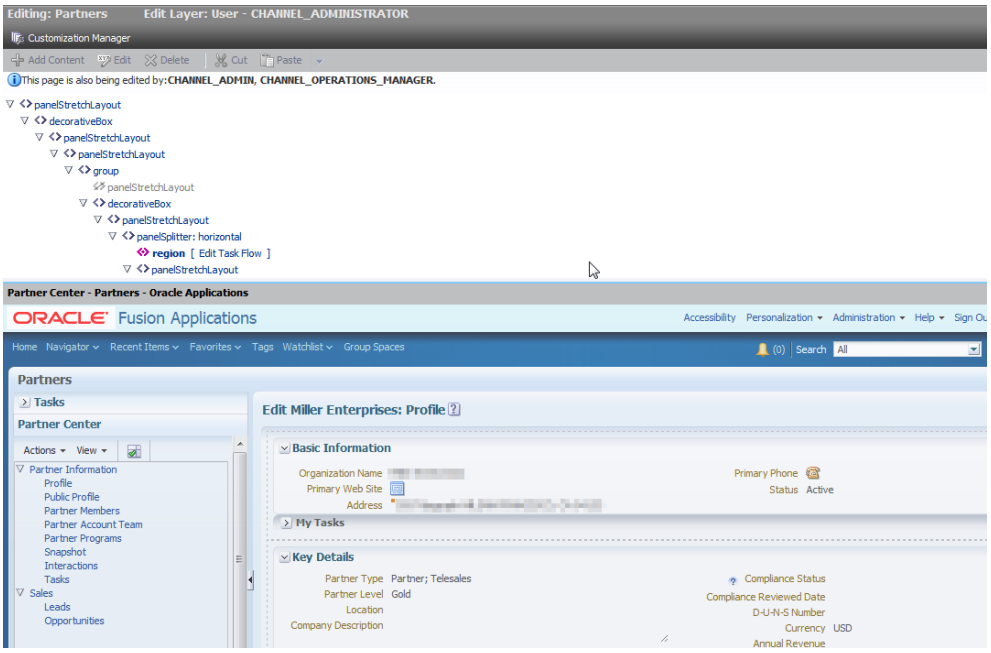
1.1.1 Personalization

The term personalization refers to the changes that every end user of the Oracle Fusion Applications product suite can make to certain artifacts in the user interface (UI) at runtime. These changes remain for that user each time that user logs in to the application. Personalization includes changes based on user behavior (such as changing the width of a column in a table), changes the user elects to save, such as search parameters, or composer-based personalizations, where an end user can redesign aspects of a page.

For composer-based personalizations, Oracle Fusion Applications includes Page Composer, which allows end users to change certain UI pages to suit their needs. For example, they can rearrange certain objects on a page, add and remove designated content, and save queries. Figure 1–1 shows the Partner Profile page in Page Composer. An end user can add other content to this page, or change the order of the current content.

> **Note:** By default, only certain personalizations are allowed. You can customize what can be personalized. For more information, see Chapter 10, "Configuring End-User Personalization."

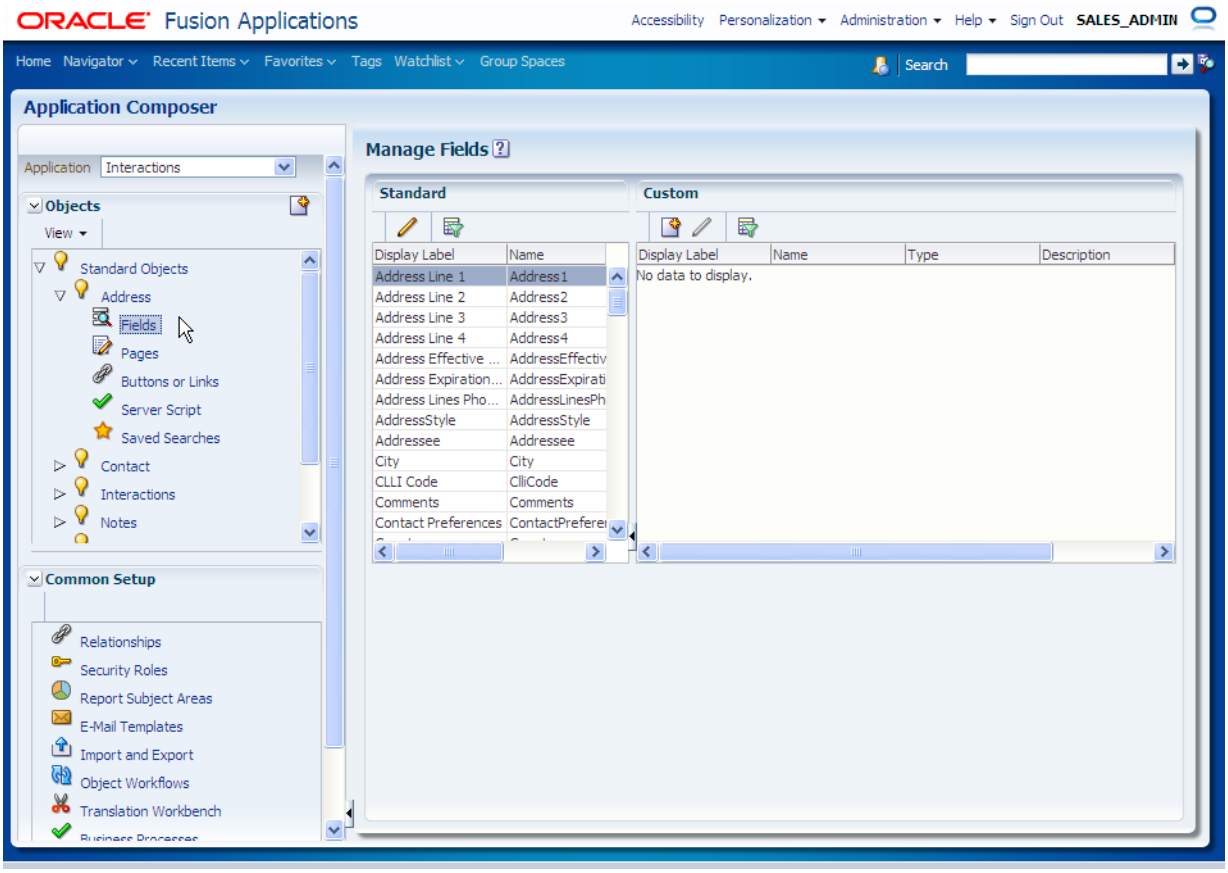
Figure 1–1 End Users Can Personalize UIs with Page Composer



1.1.2 Runtime Customizations and Extensions

Runtime customizations and extensions include those that a business analyst can make to an Oracle Fusion application at runtime using browser-based composers. These customizations and extensions are visible and usable by all or by a subset of Oracle Fusion Applications users. The types of runtime customizations and extensions range from changing the look and feel of a page, to customizing standard objects, adding a new object and associated pages and application functionality, changing workflows, defining security for new objects, and customizing reports. Figure 1–2 shows how you can customize the fields on a standard object using Application Composer, which is a runtime tool used to customize and extend certain Oracle Fusion Customer Relationship Management (Oracle Fusion CRM) applications.

Figure 1–2 Application Composer Allows You to Customize Objects at Runtime

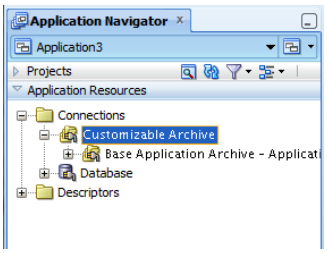


For information about customizing and extending Oracle Fusion applications using runtime tools, see the *Oracle Fusion Applications CRM Extensibility Guide*, the *Oracle Fusion Applications Common Implementation Guide*, and the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

1.1.3 Design Time Customizations and Extensions

Design time customizations and extensions include more complex changes, such as creating a **SOA composite application** or creating a new batch job, and they require deployment into a runtime environment. These **design time customizations and extensions** are most often done by Java developers using Oracle JDeveloper (a comprehensive integrated development environment), as shown in Figure 1–3, or they may be done in other tools, such as Oracle SOA Composer. The customizations are then uploaded or deployed to a running instance of Oracle Fusion Applications. Developer-level extensions are covered in this book.

Figure 1–3 Oracle JDeveloper



1.2 Understanding Customization Layers

Oracle Fusion applications contain built-in customization layers that allow you to make customizations that affect only certain instances of an application. For example, the Sales application has a layer for job **role**. When you customize an artifact, you can choose to make that customization available only to users of a specific job role, for example, a sales representative.

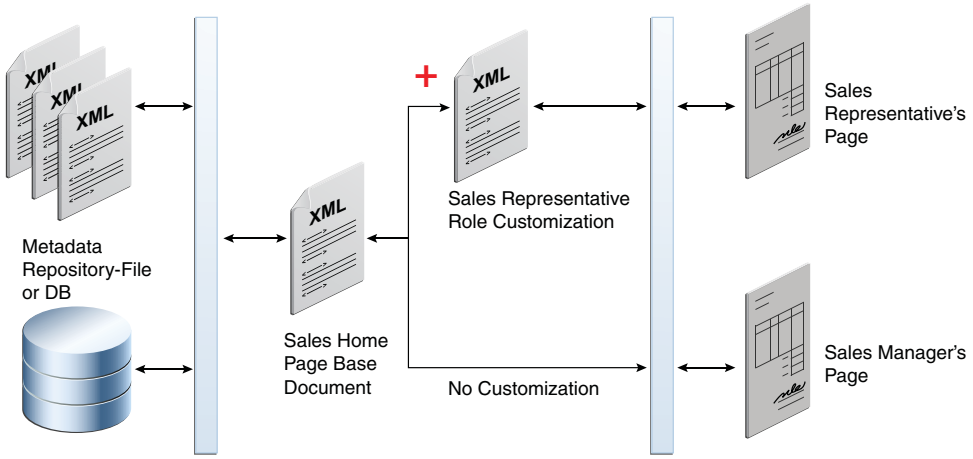
Customizations you make are not saved to the base standard artifact. Instead, they are saved to an Extensible Markup Language (XML) file that is stored in an Oracle Metadata Services (MDS) repository. This XML file acts like a list of instructions that determines how the artifact looks or behaves in the application, based on the layer that is controlling the current context. The customization engine in MDS manages this process.

For example, say you want to customize the Sales home page by removing the Quick Create panel, but only for users with the Sales Representative role. Before you make your customization, you first select the layer in which to make your customization, in this case the role layer whose value is `Sales Representative`. When you make your customization by removing that pane from the page, an XML file is generated with the instructions to remove the pane, but only in the role layer, and only when the value is `Sales Representative`. The original page file remains untouched. The customization engine in MDS then stores the XML file in an MDS repository.

Now, whenever someone logs in to the application and requests an artifact, the customization engine in MDS checks the repository for XML files that match the requested artifact and the given context, and if there is a match, it layers the instructions on top of the base artifact. In this example, whenever the Sales home page is requested (the artifact) by someone who is assigned the role of Sales Representative (the context), before the page is rendered, the customization engine in MDS pulls the corresponding XML file from the repository, layers it on top of the standard Sales home page, and removes that pane. Whenever someone who is not a Sales Representative logs in (for example, someone with the role of Sales Manager), the XML file with your changes is not layered on top, and so the Quick Create panel is displayed.

Figure 1–4 shows how the customization XML file is applied to the base document and is visible only to a sales representative.

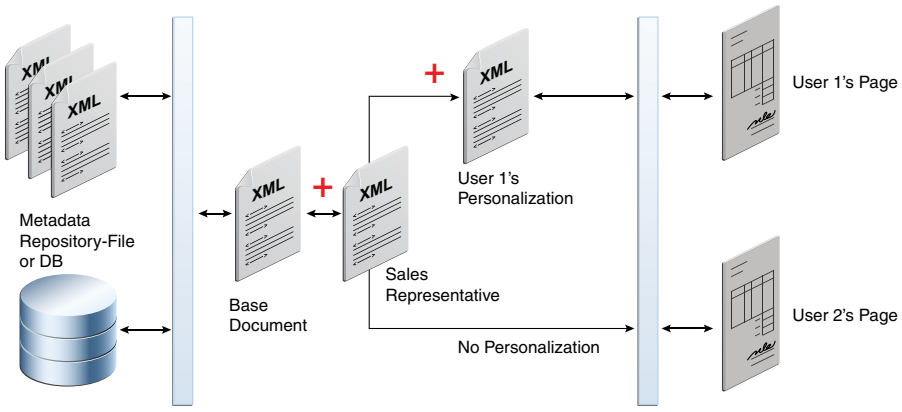
Figure 1–4 One Customization Layer Handled by the Customization Engine in MDS



All users of Oracle Fusion applications can personalize certain pages using the Personalization menu. Users can move elements around on a page, hide elements, and even add available elements to their page. When they do this personalization, the customization engine in MDS creates an XML file specific to that user.

For example, say User 1 (who has the role of Sales Representative) personalizes the Sales home page. There will then be an XML file stored in the repository, noting the changes that user made. When User 1 logs in, as in the previous example, the customization engine in MDS pulls the XML file with the sales representative customizations from the repository and layers it on top of the standard Sales home page. In addition, the engine pulls the XML file with the User 1personalizations, allowing the user to see the personalization changes along with the Sales Representative changes. When other Sales Representatives log in, they do not see the User 1 personalization changes, as shown in Figure 1–5.

Figure 1–5 Personalizations Are Also Handled by the Customization Engine in MDS



The exact customization layers available for an application depend on that application family (see the product-specific documentation from Oracle Enterprise Repository for Oracle Fusion Applications for details). However, all Oracle Fusion applications have the following customization layers:

- Global layer: When customizations are made in the **global layer**, they affect all users of the application. This layer's XML files are added for everyone, whenever

the artifact is requested. Customizations made to ADF Business Components in JDeveloper must be made in the global layer.

- Site layer: Customizations made in the **site layer** affect users at a particular location.
- User layer: The **user layer** is where all personalizations are made. Users do not have to explicitly select this layer. It is automatically selected when you use the Personalization menu.

These layers are applied in a hierarchy, and the highest layer in that hierarchy in the current context is considered the **tip**. Within the default customization layers, the global layer is the base layer, and the user layer is the tip. If customizations are done to the same object, but in different layers, at runtime, the tip layer customizations take precedence. For example, if you customize the label for a field in the site layer using Page Composer and customize the same label in the global layer using JDeveloper, the site layer customization will be displayed at runtime.

Because customizations are saved in these XML files, when you patch or upgrade your Oracle Fusion applications, the base artifacts can be updated without touching your changes. The base artifact is replaced, and when the application is run after the patch or upgrade, the XML files are simply layered on top of the new version. You do not need to redo your customizations.

Before you create customizations, you must select the layer to which you want your customizations to be applied. Most of the tools that you use to create your customizations provide a dialog where you can pick the layer for your customizations.

1.3 Understanding the Tools

Oracle Fusion Applications provides several tools to enable you to customize and extend Oracle Fusion applications. With these tools, you can perform the following tasks:

- Personalize and customize pages using Page Composer
- Customize pages using Application Composer
- Create and customize objects using Application Composer
- Create business process flows for custom objects
- Define security policies for custom objects
- Add custom attributes to a **business object**
- Customize reports and analytics
- Perform design time customizations using JDeveloper
- Customize and extend Oracle BPM Project Templates
- Configure end-user personalization
- Customize help
- Customize the Oracle Fusion Applications **skin**
- Translate custom text

For a more detailed description of the workflow you must follow when customizing and extending Oracle Fusion applications, see Chapter 2, "Understanding the Customization Development Lifecycle."

Tip: When you extend Oracle Fusion applications, you may want those extensions to be configurable using Oracle Fusion Functional Setup Manager. For more information about creating setup flows for extensions, see the Oracle Fusion Functional Setup Manager User's Guide.

1.3.1 Understanding Role-Based Access to Tools

The user interfaces in Oracle Fusion applications are controlled by role-based authentication, meaning that the information presented in the UI, and what the user can do in the UI, depends on the role assigned to the currently logged-in user. For example, if you are assigned a role with an administrative **privilege**, when you log in to Oracle Fusion Applications, you will see an Administration menu, as shown in Figure 1–6. This menu allows you to do things such as customize a page for all users, or manage customizations.

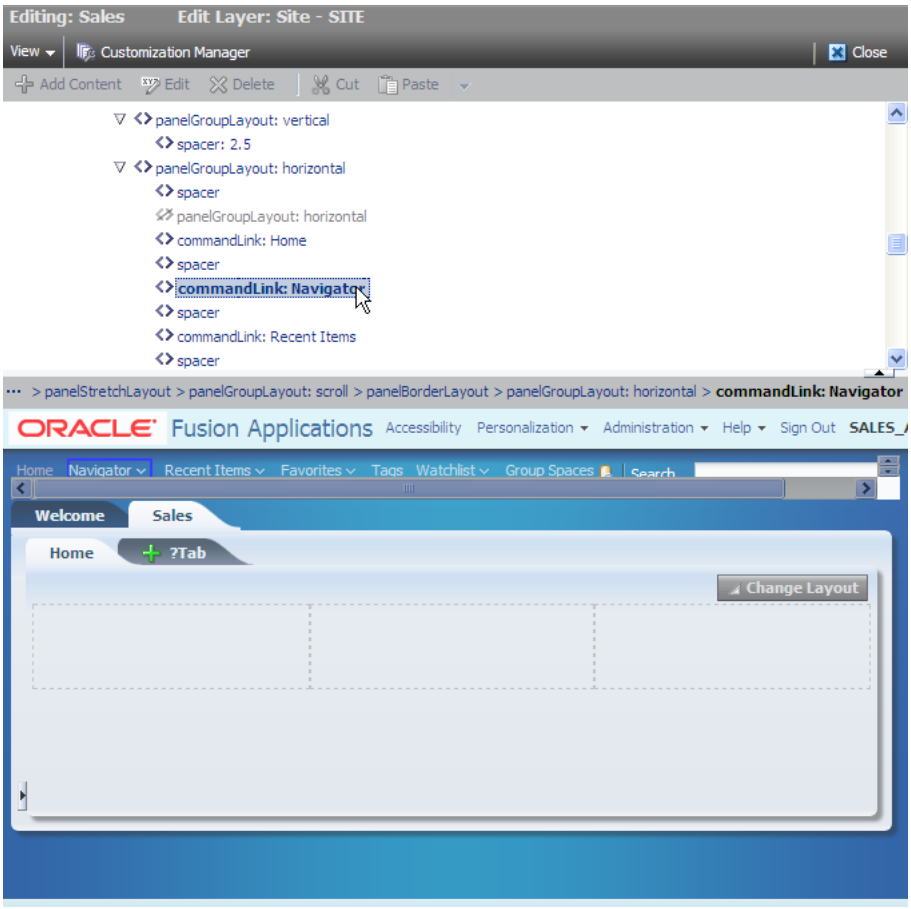
Figure 1–6 Oracle Fusion Applications Menu Bar

1.3.2 Personalizing and Customizing Pages Using Page Composer

Both personalization and customization use Page Composer to make changes to an application page. Using personalization, any user can drag and drop fields, rearrange regions, add approved external content, and save their favorite queries.

Using administration customization, you also use Page Composer to customize pages for other users. You can add fields, add validation, change defaults, rearrange regions, and add external content. Page Composer allows you to work in a WYSIWYG view, and, in some cases, Source view, as shown in Figure 1–7.

Figure 1–7 Page Composer



For more information about customizing pages, see the "Page Composer: Customizing Oracle Fusion CRM Applications" chapter in the *Oracle Fusion Applications CRM Extensibility Guide* and the "Editing a Page" chapter in the *Oracle Fusion Middleware User's Guide for Oracle WebCenter Portal: Spaces*.

1.3.3 Customizing Pages Using Application Composer

If you want to extend or customize the Sales, Marketing, Customer Center, Trading Community Architecture (TCA), and Order Capture applications that are part of the Oracle Fusion CRM product family of Oracle Fusion Applications, you can use Application Composer to customize your pages, as described in the "Creating a Work Area: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*.

Note: Only certain pages are available for customization. For a complete list, see the product-specific documentation from Oracle Enterprise Repository for Oracle Fusion Applications.

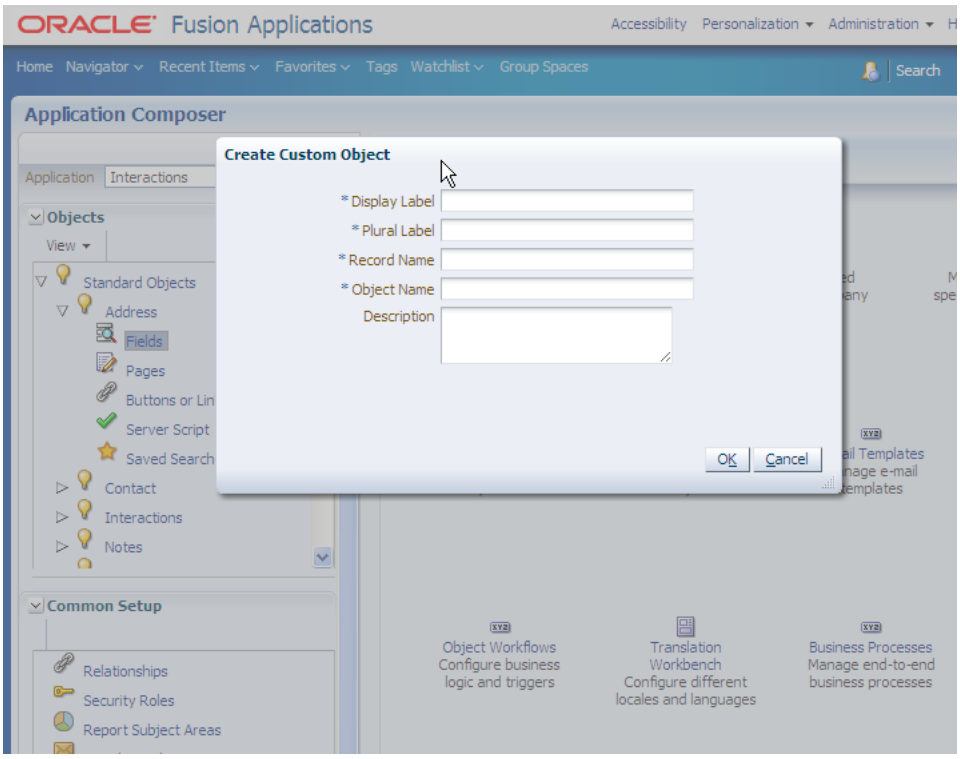
You access Application Composer by clicking the **Application Composer** link from the **Navigator** menu of Oracle Fusion Applications, as shown in Figure 1–8.

Figure 1–8 Navigator Menu

1.3.4 Creating and Customizing Objects

Application Composer allows business analysts to make more complex runtime customizations to Oracle Fusion CRM applications. In addition to customizing pages, business analysts can customize objects and all the artifacts that support them (such as fields, pages, buttons and links, security, server scripts, and saved searches), and can also extend Oracle Fusion applications by creating completely new objects and artifacts, as shown in Figure 1–9. For more information, see the "Defining Objects: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*.

Figure 1–9 Application Composer



When new objects are created, you often also create associated Work Area pages for those objects. You can add those pages to the navigator menu so that they can be accessed in the same way as standard objects. For more information, see the "Managing Menu Customizations: Highlights" section in the *Oracle Fusion Applications Common Implementation Guide*.

1.3.5 Creating and Customizing Business Process Flows for Custom Objects

When you create a new object that is not a subclass of another object, you can also create a new object workflow to manage any business processes associated with it. For example, say you used Application Composer to create a marketing object and you want to create an associated approval flow. From within Application Composer, you can access Oracle Business Process Composer and create the process that defines that flow. For applications outside of Oracle Fusion CRM, you access Business Process Composer directly from the **Navigator** menu. For more information about using the Business Process Composer, see the "Customizing and Extending BPMN Processes" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

1.3.6 Defining Security Policies for Custom Objects

When you create a new object in Application Composer, you can define security policies for it. A security policy defines the end user's level of access to the data records of the object. For more information about creating security policies for custom Oracle Fusion CRM objects, see the "Securing Custom Objects: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*.

1.3.7 Adding Custom Attributes to Business Components

If you need to add an attribute to a business component in an application that is not one of the five Oracle Fusion CRM applications, you can often use flexfields. A **flexfield** enables you to define attributes on a business component and then apply business logic to them. For example, an airline manufacturer might require very specific attributes for their orders that are not provided by the out-of-the-box implementation of an order. Because a flexfield exists for the order business component, you can use it to create and configure the desired attribute. Flexfield configurations are stored in an MDS repository, and so are safe during patching and upgrading. You access flexfields from the Setup and Maintenance menu from the Administration menu. For more information about flexfields, see the "Flexfields: Overview" section in the *Oracle Fusion Applications Common Implementation Guide*.

1.3.8 Customizing Reports and Analytics

Oracle Fusion Applications comes with a complete set of reports. You can customize these reports (for example, change the layout) to fit your particular business needs. Additionally, if you customize or create a business object, you can create a new report for that object. For more information, see the "Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

1.3.9 Performing Design Time Customizations

To customize or create business objects outside of the five Oracle Fusion CRM applications, or when required customizations cannot be made in one of the runtime composers, use JDeveloper. When you work in a JDeveloper environment, you create an application workspace that contains your changes and additions. When you create this application workspace, you do so in the Oracle Fusion Applications Developer role. Like Oracle Fusion Applications, JDeveloper uses roles to shape what you see and can do in the integrated development environment (IDE). Work done in a developer role is stored in actual projects with code that gets deployed to an environment. Use the Oracle Fusion Applications Administrator Customization role when customizing an existing standard object (as opposed to creating a new object). Work done in this role is saved to an XML file that gets deployed into an MDS repository, keeping your changes separate from the base code. For more information about how to set up your

JDeveloper customization environment, see Chapter 3, "Using Oracle JDeveloper for Customizations."

> **Note:** You cannot create your own roles to define what you see and what you can do in JDeveloper.

Developers can use JDeveloper to create and customize view pages, business objects, task flows (reusable components that specify the control flow in an application), searches, and **resource bundles**. All customizations and extensions created in JDeveloper must be deployed to an environment. For more information about using JDeveloper to customize business objects and associated artifacts, see Chapter 4, "Customizing and Extending Oracle ADF Application Artifacts."

SOA composite applications are the foundation on which Oracle Fusion applications are built: they are the glue that holds all the different components together and they allow the different applications to work in a unified manner. SOA composite applications contain **service components** such as Business Process Execution Language (**BPEL**) process flows. These **BPEL process** flows provide communication between applications, additional human-based workflows, and business rules that determine the branching in those flows. Developers can customize existing SOA composite applications or create new ones using a mixture of JDeveloper and browser-based tools. Customized and extended SOA composite applications are all stored in MDS repositories. For more information, see Chapter 5, "Customizing and Extending SOA Components."

1.3.10 Customizing and Extending Oracle BPM Project Templates

Some Oracle Fusion applications provide business process modeling (BPM) project templates that you can use to create BPM projects. BPM projects consist of SOA artifacts, such as **business rules** and **human tasks**, and Business Process Modeling and Notation (BPMN) processes. You can customize these project templates to suit your business needs. For more information, see Chapter 6, "Customizing and Extending Oracle BPM Project Templates."

1.3.11 Understanding Other Available Customizations

When you create custom pages, you may want to make them personalizable, so that end users can change the page for themselves. For more information, see Chapter 10, "Configuring End-User Personalization." Also, when you make any type of customization or extension to Oracle Fusion applications, you might have to change the embedded help that appears on the screen. For more information, see Chapter 11, "Customizing Help."

You can customize the look and feel of Oracle Fusion Applications, such as change the colors or add a logo. For more information, see Chapter 12, "Customizing the Oracle Fusion Applications Skin."

Some customizations can be translated. For more information, see Chapter 9, "Translating Custom Text."

1.3.12 What You Can Customize and Extend and with Which Tool

There are many scenarios for which you can customize Oracle Fusion applications. The following tables identify for each scenario the artifacts that you can customize or create in Oracle Fusion Applications, what tool you use, the type of user that can make the change, and whether the changes are stored in an MDS repository:

- View page customizations: Table 1–1
- Branding customizations: Table 1–2
- Object customization: Table 1–3
- Business process customizations: Table 1–4
- Report customizations: Table 1–5
- Analysis and dashboard customizations: Table 1–6
- Oracle Enterprise Scheduler job customizations: Table 1–7
- Security customizations: Table 1–8

Note: Application Composer is available only if you want to make changes in the following Oracle Fusion CRM applications:

- Marketing
- Sales
- Customer Center
- Trading Community Architecture (TCA)
- Order Capture

Note: While you can customize view pages in Page Composer and Application Composer, only certain pages are configured to allow it. If the customization that you want to make is not available in Page Composer, then you must use JDeveloper to make the customization.

Table 1–1 View Page Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Add, move, delete, show, or hide components on a page.	Page Composer	Business Analyst	Yes	"Building Pages" chapter in the *Oracle Fusion Middleware User's Guide for Oracle WebCenter Portal: Spaces*
Change a page layout.	Page Composer	Business Analyst	Yes	"Changing the Page Layout" section in the *Oracle Fusion Middleware User's Guide for Oracle WebCenter Portal: Spaces*
Create a site-level search for all users.	Page Composer	Business Analyst	Yes	"Editing a Page in Page Composer" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Customize a page title.	Page Composer	Business Analyst	Yes	"Editing a Page in Page Composer" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Customize a task list menu.	Page Composer	Business Analyst	Yes	"Editing a Page in Page Composer" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*

Table 1–1 (Cont.) View Page Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Customize popup window content.	Page Composer	Business Analyst	Yes	"Editing a Page in Page Composer" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Add fields, buttons, links, to a standard page (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Application Composer: Using the Application Composer" chapter in the *Oracle Fusion Applications CRM Extensibility Guide*
Customize attributes for a flexfield on a page.	Page Composer	Business Analyst	Yes	"Flexfields: Overview" section in the *Oracle Fusion Applications Common Implementation Guide*
Customize properties for UI components on a standard page.	Page Composer	Business Analyst	Yes	"Setting Component Properties" section in the *Oracle Fusion Middleware User's Guide for Oracle WebCenter Portal: Spaces*
Customize properties for UI components on a standard page (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Editing an Object: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*
Make UI components on a page personalizable.	Page Composer	Business Analyst	Yes	Section 10.3, "Configuring End-User Personalization for Components"
Customize the **UI Shell template**.	JDeveloper	Developer	Yes	Section 4.10, "Editing the UI Shell Template"
Customize the **UI Shell template**.	Page Composer	Business Analyst	Yes	"Editing the UI Shell Template Used by All Pages" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Define resource bundles.	JDeveloper	Developer	Yes	Section 4.12, "Customizing or Adding Resource Bundles"
Make a custom page personalizable (custom pages created in Application Composer are customizable by default).	JDeveloper	Developer	Yes	Section 10.2, "Allowing Pages to Be Personalized by End Users in Page Composer"
Customize onscreen text that is displayed when the end user mouses over a button or link.	Page Composer	Business Analyst	Yes	Section 11.3, "Customizing or Adding Bubble Embedded Help"

Table 1–1 (Cont.) View Page Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Customize onscreen help text.	JDeveloper	Developer	Yes	Section 11.4, "Customizing or Adding Static Instructions, In-Field Notes, and Terminology Definitions"
Change the look and feel of the entire application.	JDeveloper	Developer	No	Chapter 12, "Customizing the Oracle Fusion Applications Skin"
Translate custom text.	JDeveloper	Developer	Yes	Chapter 9, "Translating Custom Text"

Table 1–2 Branding Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Customize the UI Shell template.	JDeveloper	Developer	Yes	Section 4.10, "Editing the UI Shell Template"
Customize the UI Shell template.	Page Composer	Business Analyst	Yes	"Editing the UI Shell Template Used by All Pages" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Change the look and feel of the entire application.	JDeveloper	Developer	No	Chapter 12, "Customizing the Oracle Fusion Applications Skin"
Change the logo.	JDeveloper	Developer	No	Chapter 12, "Customizing the Oracle Fusion Applications Skin"
Customize report layouts.	Oracle BI Publisher	Business Analyst	No	"Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*

Table 1–3 Object Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Customize business objects.	JDeveloper	Developer	Yes	Section 4.2, "Editing Existing Business Components"
Customize objects (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Editing an Object: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*
Add an attribute to a business object using flexfields (not Oracle Fusion CRM).	Setup and Maintenance work area	Business Analyst	No	"Flexfields: Overview" section in the *Oracle Fusion Applications Common Implementation Guide*
Create business objects.	JDeveloper	Developer	Yes	Section 4.5, "Creating Custom Business Components"
Create objects (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Editing an Object: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*

Table 1–3 (Cont.) Object Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Add a business object page to the navigator menu	Setup and Maintenance work area	Business Analyst	No	"Managing Menu Customizations: Highlights" section in the *Oracle Fusion Applications Common Implementation Guide*
Add custom object work area pages to the navigator menu (Oracle Fusion CRM)	Application Composer	Business Analyst	No	"Managing Menu Customizations: Highlights" section in the *Oracle Fusion Applications Common Implementation Guide*"
Add validation to a business object	JDeveloper	Developer	Yes	"Section 4.5, "Creating Custom Business Components"
Add validation to an object (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Groovy Scripting: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*"
Customize saved searches for a custom object (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Saved Searches for CRM Objects: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*"
Create searches for an object.	JDeveloper	Developer	Yes	Section 4.9, "Customizing or Creating a Custom Search Object"
Create saved searches for a custom object (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Saved Searches for CRM Objects: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*
Customize task flows for an object.	JDeveloper	Developer	Yes	Section 4.3, "Editing Task Flows"
Create task flows for an object.	JDeveloper	Developer	Yes	Section 4.6, "Creating Custom Task Flows"
Customize object workflows for an object (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Object Workflows: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*
Create object workflows for an object (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Object Workflows: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*

Table 1–4 Business Process Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Create a BPMN process in a BPM project.	Business Process Composer	Business Analyst	Yes	"Customizing and Extending BPMN Processes" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Create a BPMN approval process in a BPM project (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Customizing and Extending BPMN Processes" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Customize custom BPM projects.	Business Process Composer	Business Analyst	Yes	"Customizing and Extending BPMN Processes" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Customize custom BPM projects (Oracle Fusion CRM).	Application Composer	Business Analyst	Yes	"Customizing and Extending BPMN Processes" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Customize BPM project templates.	Oracle BPM Studio	Developer	Yes	Chapter 6, "Customizing and Extending Oracle BPM Project Templates"
Customize a business rule (either an approval configuration and assignment rule or a nonapproval business rule), **domain value map**, or composite application endpoint property.	Oracle BPM Worklist, Oracle SOA Composer, and Fusion Applications Control	Developer	Yes	Section 5.2, "Customizing SOA Composite Applications"
Merge the customizations from a previous revision of a SOA composite application into a new revision.	Opatch	Administrator	Yes	Section 5.3, "Merging Runtime Customizations from a Previously Deployed Revision into a New Revision"
Customize a BPEL process or a mediator component, or add additional SOA components.	JDeveloper	Developer	Yes	Section 5.4, "Extending or Customizing Custom SOA Composite Applications"

Table 1–5 Report Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Create report layout.	Oracle BI Publisher	Business Analyst	No	"Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Customize report layouts.	Oracle BI Publisher	Business Analyst	No	"Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Customize style templates.	Oracle BI Publisher	Business Analyst	No	"Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Create a report.	Oracle BI Publisher	Business Analyst	No	"Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Translate a report.	Oracle BI Publisher	Business Analyst	No	"Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Create a report subject area (Oracle Fusion CRM)	Application Composer	Business Analyst	No	"Custom Subject Areas: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*

Table 1–6 Analysis and Dashboard Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Customize analytics.	Reports and Analytics pane	Business Analyst	No	"Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*
Customize and extend the Oracle BI repository (RPD file).	JDeveloper, Oracle BI Administration Tool	Developer	No	"Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*

Table 1–7 Oracle Enterprise Scheduler Job Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Create jobs.	JDeveloper	Developer	No	Chapter 7, "Customizing and Extending Oracle Enterprise Scheduler Jobs"

Table 1–7 (Cont.) Oracle Enterprise Scheduler Job Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Customize jobs.	Fusion Applications Control	Administrator	No	Chapter 7, "Customizing and Extending Oracle Enterprise Scheduler Jobs"
Submit jobs.	Fusion Applications Control	Administrator	No	Chapter 7, "Customizing and Extending Oracle Enterprise Scheduler Jobs"
Submit jobs.	JDeveloper	Developer	No	Chapter 7, "Customizing and Extending Oracle Enterprise Scheduler Jobs"

Table 1–8 Security Customization Scenarios in Oracle Fusion Applications

Customization/ Extension	Tool	Type of User	MDS?	Where to Find Information
Add data security to a custom object.	Manage Data Security task accessible from the Setup and Maintenance work area	Developer	No	Section 8.4, "Defining Data Security Policies on Custom Business Objects"
Opt into data security policies for custom objects.	JDeveloper	Developer	No	Section 8.5, "Enforcing Data Security in the Data Model Project"
Grant access to application artifacts.	JDeveloper	Developer	No	Section 8.6, "Defining Function Security Policies for the User Interface Project"
Grant access to custom objects (Oracle Fusion CRM).	Application Composer	Business Analyst	No	"Securing Custom Objects: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*
Enable elevated privileges customization.	Application Composer	Business Analyst	No	"Securing Custom Objects: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*

1.3.13 Installing Customization Tools

All the business analyst tools are available from the navigator menu of Oracle Fusion Applications. However, for most of the design time tools, you must install and configure a version of JDeveloper that is certified for your Oracle Fusion Applications release. This version of JDeveloper, along with the necessary extensions for customizing and extending Oracle Fusion Applications, is in the release's Oracle Fusion Applications Media Pack, which is available from Oracle Software Delivery Cloud at `http://edelivery.oracle.com`. After installing JDeveloper, they must set up their environment for customization and extending.

For procedures for installing JDeveloper and setting it up for extending (that is, for creating new objects), see the "Setting Up Your Development Environment" and "Setting Up Your JDeveloper Application Workspace and Projects" chapters in the *Oracle Fusion Applications Developer's Guide*.

For procedures for setting up JDeveloper for customizations, see Chapter 3, "Using Oracle JDeveloper for Customizations."

2

Understanding the Customization Development Lifecycle

This chapter discusses the typical workflow for customizing and extending Oracle Fusion applications. It describes how to use sandboxes to perform customizations in an environment that is separate from the full test environment, publish the changes to a full test environment, and move the changes to other environments.

This chapter includes the following sections:

2.1 Understanding Typical Customization Workflows

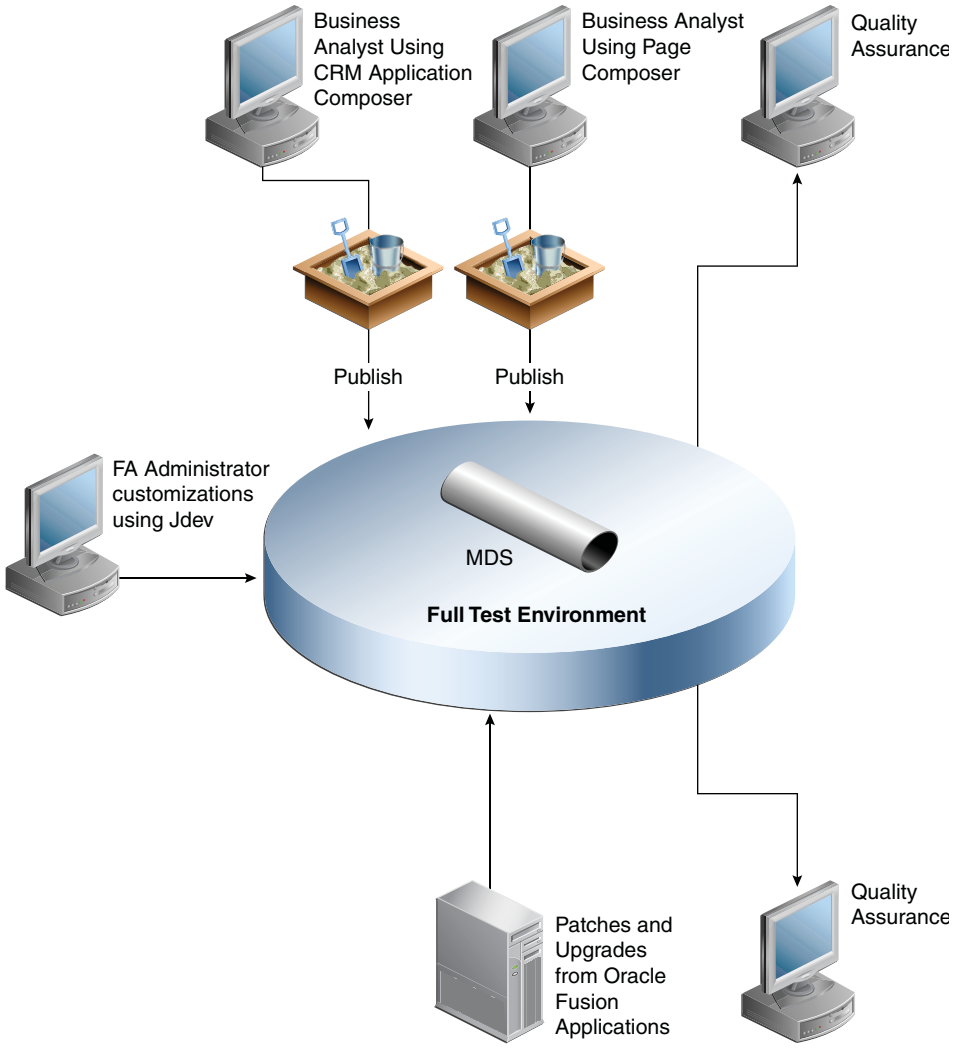
All customizations and extensions to Oracle Fusion Applications should be done in a full test environment, as shown in Figure 2–1. Typically, this environment contains one or more Oracle Fusion applications that will then be moved to a production environment after all customizations and extensions are complete and tested.

As described in Section 2.1.1, "Runtime Customization Workflow," business analysts using Page Composer and Oracle Fusion CRM Application Composer (Application Composer) make their application customizations in a **sandbox**. Sandboxes store the customizations in isolated, protected Oracle Metadata Services (MDS) labels that are available only when you work in that particular sandbox. The changes can be done in a test-only sandbox (that is, the code in the sandbox is for testing only, and is never deployed), or they can be done in a sandbox that is then published to the full test environment.

Developers using design time tools, such as Oracle JDeveloper, have the option to publish their customizations to a sandbox, as described in Section 2.1.2, "Design Time Customization Workflow."

After testing, you can then move the customizations to the mainline code as described in Section 2.3, "Exporting and Moving Customizations."

Figure 2–1 Customization Workflow in a Full Test Environment



Tip: When you **extend** Oracle Fusion applications, you might want users to be able to configure the extensions using Oracle Fusion Functional Setup Manager. For more information about creating task flows for setup activities for extensions, see the *Oracle Fusion Functional Setup Manager Developer's Guide*.

2.1.1 Runtime Customization Workflow

When you use Application Composer and Page Composer to make runtime customizations to Oracle Fusion applications, you use sandboxes to save your changes in an isolated environment. For example, before you begin making customizations, you create a sandbox named `MySandbox` and make your customizations in that sandbox. If others want to see the customizations, then they would use `MySandbox`.

You also use a sandbox when you define security policies for custom objects that you have created using Application Composer. A **security sandbox** stores the security information in new database tables that are available only when you choose to work in that sandbox.

After you complete your customizations, others can review and validate the sandbox. Then you can publish the sandbox to the full test environment where your

customizations become part of that repository. For more information about sandboxes, see Section 2.2, "Using the Sandbox Manager."

2.1.2 Design Time Customization Workflow

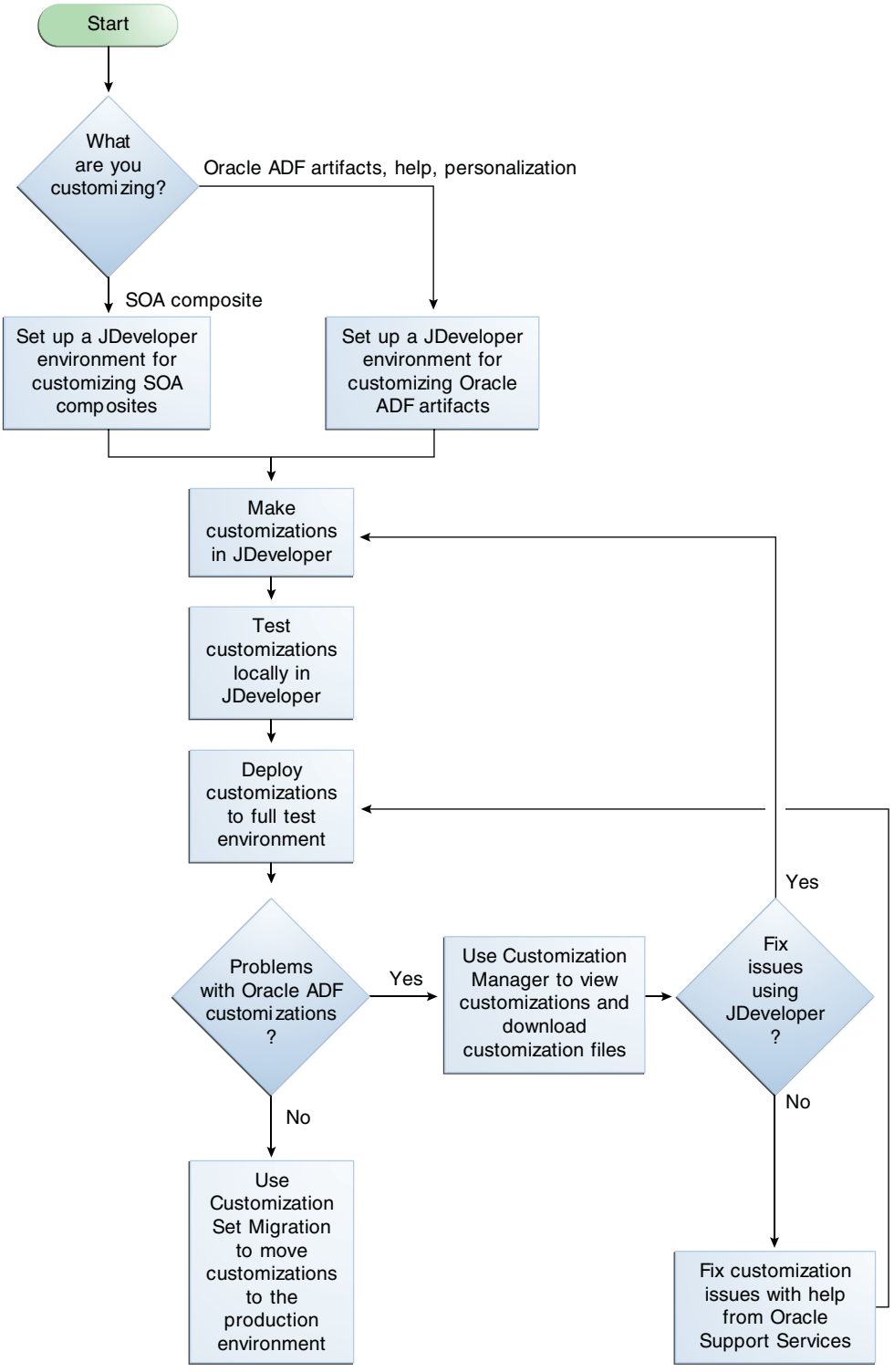
After you create these customizations using JDeveloper, you can test them locally in JDeveloper and then deploy your customizations to a sandbox. Note that security customizations done at design time are not saved to a sandbox. The migration of security customizations is discussed in Section 8.3.10, "What You Cannot Do with Security Policies at Design Time."

Additionally, you can use source control software to manage design time customizations. For more information about what source control software JDeveloper supports, see the "Versioning Applications with Source Control" topic of the JDeveloper online help.

Because your customizations (other than security changes) are stored in customization XML files in an MDS repository, they can also be viewed and managed using the Manage Customizations dialog.

Figure 2–2 shows the flow for a typical design time customization process.

Figure 2–2 Typical Design Time Customization Workflow



2.2 Using the Sandbox Manager

The sandbox manager is a tool for managing the different types of customization changes that can be applied to an application These changes that are contained within a sandbox do not affect the mainline code. You can test and validate the changes by publishing the sandbox to the full test environment. After the application has been tested, it can then be moved to the production environment.

There are three types of sandboxes:

- Metadata

  The metadata sandbox supports making changes to the application's metadata stored in the MDS repository.

- Security

  The security-enabled sandbox supports making **data security** changes.

- Flexfield

  The flexfield sandbox is not created using the sandbox manager. Use the flexfield UI to make changes to the flexfields and then deploy them to the sandbox. The flexfield deployment process manages the creation of the sandbox.

To **customize** an Oracle Fusion application in runtime, you must first create a sandbox and then use Page Composer or Application Composer to make the customizations. For information about using sandboxes for runtime customizations, see the "Application Composer: Using Sandboxes" chapter in the *Oracle Fusion Applications CRM Extensibility Guide* the "Using the Sandbox Manager" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

Oracle Business Process Composer and Oracle SOA Composer are also runtime customization tools, but they do not use the sandbox manager. They have their own mechanisms for handling customization changes:

- For information about using Oracle Business Process Composer, see the "Customizing and Extending BPMN Processes" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.
- For information about using Oracle SOA Composer, see Chapter 5, "Customizing and Extending SOA Components."

For non-Cloud implementations, a **metadata sandbox** that you create using the sandbox manager is available in JDeveloper when you are creating and deploying customizations intended for a deployed Oracle Fusion application in Oracle WebLogic Server. The available sandboxes will appear in a selection list in JDeveloper during deployment. For more information, see Section 4.14, "Deploying Oracle ADF Customizations and Extensions." Note that the security sandboxes created using the sandbox manager are not available in JDeveloper.

The metadata and security sandbox sessions can be saved, downloaded, and imported as files into other Oracle Fusion applications.

If more than one person is using a sandbox, then you must take care to prevent conflicts. For more information, see the "Multiple Sandbox User Conflicts: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide* and the "Using the Sandbox Manager" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

2.3 Exporting and Moving Customizations

There are several tools available for exporting and moving customizations. These tools enable you to perform the following tasks:

- Move customizations and extensions to another Oracle Fusion Applications environment, such as the production environment.
- Diagnose issues seen in the test environment.
- Send files to Oracle Support Services for further diagnosing.
- Import a customization into another environment. For example, a customization developer using JDeveloper might need to see customizations done by someone else.

For information about the tools that are available for exporting and moving customizations, see the "Exporting and Moving Customizations" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

Part II

Design Time Customizations and Extensions

Part II contains the following chapters:

3

Using Oracle JDeveloper for Customizations

This chapter describes how to configure Oracle JDeveloper for implementing customizations in Oracle Fusion applications. It also describes how to **customize** Service-Oriented Architecture (SOA) composite applications with JDeveloper, including setting up the JDeveloper application workspace and **SOA composite application** project for Oracle Metadata Services (MDS) Repository customization, customizing the SOA composite application, and customizing the SOA **resource bundle**.

This chapter includes the following sections:

3.1 About Using JDeveloper for Customization

You use JDeveloper when you need to customize or create objects or security outside of CRM applications, or when you need to make more sophisticated changes, like changes to SOA composite applications, Oracle Enterprise Scheduler jobs, Oracle Business Process Management project templates, or embedded help. While you use JDeveloper to both customize existing standard objects and to create new custom objects, the procedures you use for each are different.

New custom objects created in JDeveloper are not saved into MDS Repository, and are therefore created in a standard application workspace using the Oracle Fusion Applications Developer **role**. However, when you customize standard objects, those customizations *are* saved into MDS Repository, and therefore must be implemented using the Oracle Fusion Applications Administrator Customization role. Implementing the customizations using this customization role ensures that your changes are saved to the upgrade-safe MDS Repository, and not written directly to the standard object. In the future, when you patch or upgrade Oracle Fusion Applications, your customizations held in these metadata files are not affected, so you do not have to redo them. For more information about customizations and MDS Repository, see Chapter 1, "Customizing and Extending Oracle Fusion Applications."

When customizing Oracle Application Development Framework (Oracle ADF) artifacts, you create a special customization application workspace, using the developer role. This application workspace includes a connection to a deployed Oracle Fusion Applications environment (typically a test environment), which allows you to import the artifacts you want to customize into your application workspace. This customization application workspace is automatically configured to work within Oracle Fusion Applications, so that when you test and deploy your customizations,

they behave as though they were native Oracle Fusion Applications. When customizing SOA composite applications, you create a SOA Composite application workspace in the developer role.

After the application workspace is created, you switch roles to the customization role and import the Oracle ADF artifact or the SOA archive you want to customize. You then make your customizations to the imported artifact. After completion, you package and deploy the artifacts in the application workspace to the Oracle Fusion Applications environment.

Often, you must perform both customizations (customizing an existing standard object) and extensions (creating a new object). For example, say you want to create a new entity object and expose that new object in an existing application module. First, because you are creating a new custom entity object, you would create a standard application workspace and then create your entity object. After completion, you would package the application workspace as an ADF Library, and place it into the exploded enterprise archive (EAR) directory for your test environment. Next, you would create a customization application workspace, and import both the new entity object library and the library that contains the application module to which you must add the entity object. After both are imported, you log in using the customization role and make the customizations to the application module. After customizations are complete, you would deploy the customizations to the test environment.

3.1.1 About Customizing Oracle ADF Artifacts

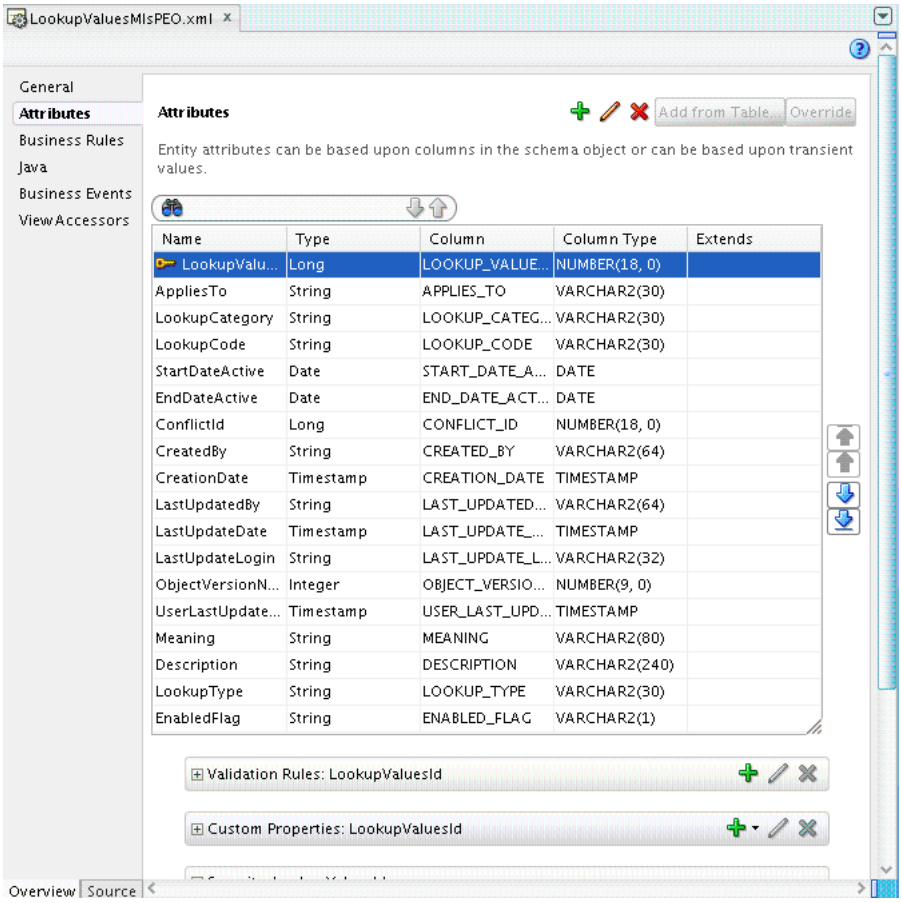
Oracle Fusion applications are built using Oracle Application Development Framework (Oracle ADF) artifacts on Oracle Fusion Middleware, including the following:

- Application modules: An application module is the transactional component that UI clients use to work with application data. It defines an updatable data model along with top-level procedures and functions (called service methods) related to a logical unit of work related to an end-user task.
- Entity objects: An entity object represents a row in a database table and simplifies modifying its data by handling all data manipulation language (DML) operations for you. It can encapsulate business logic to ensure that your **business rule**s are consistently enforced. You associate an entity object with other entity objects to reflect relationships in the underlying database schema to create a layer of business domain objects to reuse in multiple applications.
- View objects: A view object represents a SQL query and simplifies working with its results. You use the SQL language to join, filter, sort, and aggregate data into whatever form is required by the end-user task being represented in the user interface. This includes the ability to link a view object with other view objects to create master-detail hierarchies of any complexity. When end users modify data in the user interface, your view objects collaborate with entity objects to consistently validate and save the changes.
- Task flows: A task flow defines the flow of control throughout an application. It can also be included in a page as a region, where users can navigate through a series of page fragments, without leaving the original page.
- JSPX pages and page fragments: The view layer of Oracle Fusion Applications consists of a small number of pages per application. These pages then contain task flows, which in turn contain several page fragments.

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

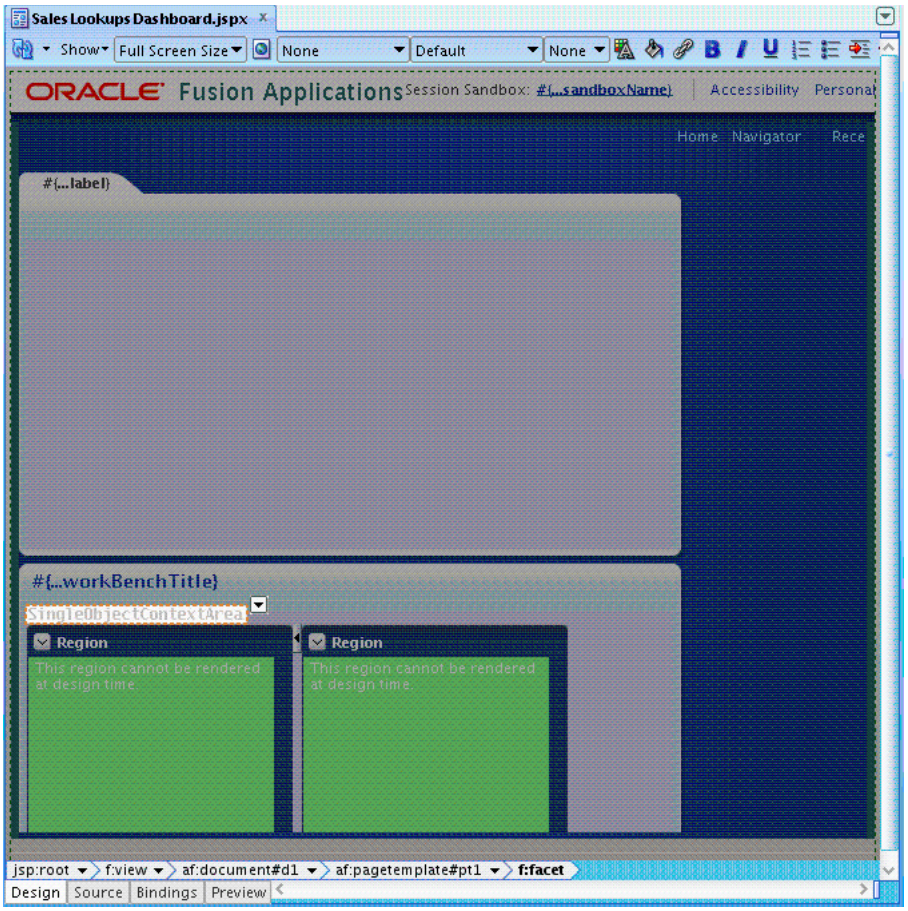
When you customize Oracle ADF artifacts, you usually work in an overview editor that allows you to make your customizations declaratively. For example, Figure 3–1 shows the editor for an entity object. Among other things, you can set validation or change how the UI displays the data.

Figure 3–1 Overview Editor for Entity Object



For JSP pages, you work in a WYSIWYG environment using the Design tab in the editor window, as shown in Figure 3–2.

Figure 3–2 Design Editor for JSP Pages



3.1.2 About Using JDeveloper to Customize SOA Composite Applications

Oracle Fusion applications are built using SOA composite **artifacts** on Oracle Fusion Middleware, which include the following:

- **Service components**: A **service component** implements the business logic or processing rules of a SOA composite application. Available service components include the following:
  - **Business Process Execution Language (BPEL)** processes: A **BPEL process** enables you to integrate a series of business activities and services into an end-to-end business process flow.
  - Business rules: A **business rule** enables you to create business decisions in your business process flow based on rules.
  - Human tasks: A **human task** enables you to create human workflows that describe the tasks for users or groups to act upon as part of an end-to-end business process flow. You use Oracle Business Process Management Worklist (Oracle BPM Worklist) to act upon the tasks during runtime.
  - Oracle Mediator: An Oracle Mediator enables you to define services that perform message and event routing, filtering, and transformations within the SOA composite application.

- **Binding components**: A **binding component** establishes the connection between a SOA composite application and the external world. There are two types of binding components:
  - Services provide the outside world with an entry point to the SOA composite application. The Web Services Description Language (WSDL) file of the service advertises its capabilities to external applications. These capabilities are used for contacting the SOA composite application components. The binding connectivity of the service describes the protocols that can communicate with the service (for example, Simple Object Access Protocol (SOAP)/Hypertext Transfer Protocol (HTTP) or Java EE connector architecture (JCA) adapter).
  - References enable messages to be sent from the SOA composite application to external services in the outside world.
- **Wires**: A **wire** connects services, service components, and references into a complete SOA composite application.

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

3.1.3 Before You Begin Using JDeveloper to Customize

Before you use JDeveloper to customize, be familiar with the Oracle Fusion application architecture that enables customization, as described in Chapter 1, "Customizing and Extending Oracle Fusion Applications." Also understand the typical workflows for working with customizations, as described in Chapter 2, "Understanding the Customization Development Lifecycle."

Do the following before using JDeveloper to customize:

- Deploy the application you are customizing to a test environment. You must have access to the test environment and to the exploded EAR directory for that application.
- Install JDeveloper and set up your development environment. Before you can implement customizations using JDeveloper, you must create a customization application workspace that imports the necessary parts of the application you want to customize. For more information, see Section 1.3.13, "Installing Customization Tools."

Note: Before you can use JDeveloper to customize your application, JDeveloper must have access to the customization layers for the application. To enable JDeveloper to see the customization classes that define the customization layers, use the `-Dide.extension.extra.search.path` VM option, as described in the "Adding Customization Extension Bundles to the jdev.conf File" section of the *Oracle Fusion Applications Developer's Guide*.

For information about locating the Java archive (JAR) files containing the product-specific customization classes, see the product-specific documentation from Oracle Enterprise Repository for Oracle Fusion Applications. You can also use the steps in the "Adding Customization Extension Bundles to the jdev.conf File" section of the *Oracle Fusion Applications Developer's Guide* to locate the JAR files.

3.2 Customizing Oracle ADF Artifacts with JDeveloper

To customize Oracle ADF artifacts, you first create a customization application workspace, using the **Oracle Fusion Applications Developer** role in JDeveloper. After the application workspace is created, you exit JDeveloper and then reenter, using the **Oracle Fusion Applications Administrator Customization** role and import and customize your artifacts.

3.2.1 Creating the Customization Application Workspace

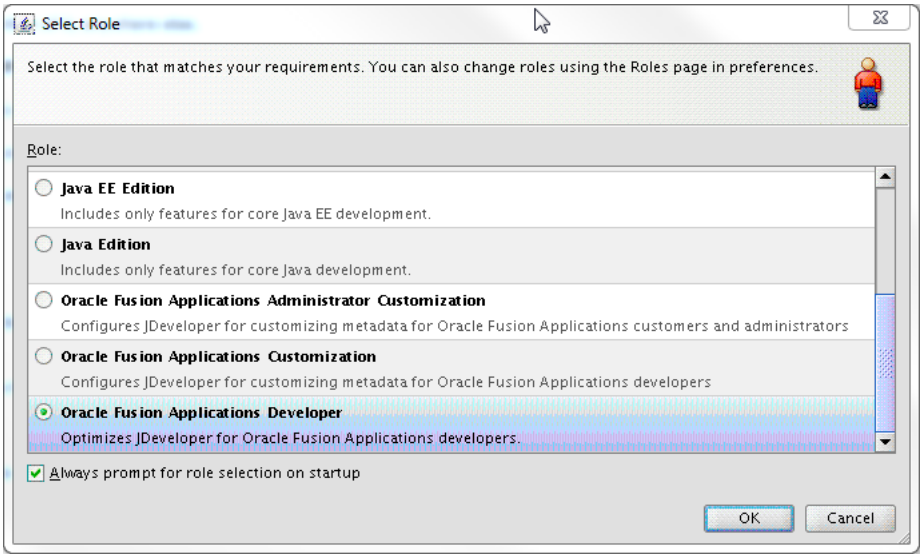
You must set up a customization application workspace in JDeveloper to create the application that holds your customizations.

> **Note:** Before you start the FA Customization Application wizard to create a customization application workspace, make sure that the entire topology for the application you want to customize is up and running. The `conn.xml` file in Oracle Fusion applications contains EL expressions pointing to common server endpoints, such as the BI server, that cannot be resolved unless the entire topology for the application is running.

To create the customization application workspace:

1. Start JDeveloper using the **Oracle Fusion Applications Developer** role, as shown in Figure 3–3.

Figure 3–3 Oracle Fusion Applications Developer Role



2. In JDeveloper, from the main menu, choose **File** > **New** to open the New Gallery. In the New Gallery, select **Applications > Fusion Applications Customization Application**.
3. In the Step 1 page of the FA Customization Application dialog, enter the following and click **Next**:
   - **Application Name** and **Directory**: These are the name and location of your customization application, and can be anything you like.

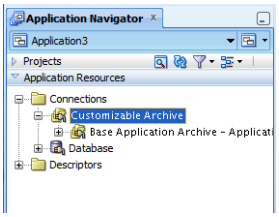
- **Fusion Database**: Enter the connection to your Oracle Fusion database.
- **Application Package Prefix**: This can be anything, but must not start with `oracle`.
- **Deployed Application Ear:** Browse to the exploded EAR for the application you want to customize.
- **Policy Store Security Information**: Browse to the exported `jazn-data.xml` file. First, you must export all predefined **function security** policies of the application that you are customizing into a `jazn-data.xml` file. For details about how to export the application policy store, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*. For information about security customization, see Chapter 8, "Customizing Security for Oracle ADF Application Artifacts."

4. Complete the wizard by changing any default settings as needed.

After you complete the wizard, an application with a project is created for you. This application is configured to be the same as a deployed Oracle Fusion application. For example, it is connected to the same database, same metadata repository, and has similar `web.xml` and `weblogic.xml` settings. This configuration allows it to work correctly when deployed into your Oracle Fusion Applications environment, and also ensures that when you test your customizations locally in JDeveloper Integrated WebLogic Server, the customizations behave as they will in the full test environment.

JDeveloper also creates a connection to the exploded EAR directory named **Customizable Archive**, which is accessible from the Application Resources panel of the Application Navigator. Figure 3–4 shows a connection to the exploded EAR directory for an application.

Figure 3–4 Application Resources Connection to Exploded EAR Contents



3.2.2 Determining Which Oracle ADF Artifacts You Need to Customize

Most often, the customizations you want to make are surfaced on an existing page. For example, say you want to add a field to a page. So, you first must identify the page to customize, which may actually be a page fragment within a task flow. You then must identify which business objects you'll need to customize to add the field.

The easiest way to identify which artifacts you need to customize is to follow this path:

1. In a runtime environment, access the page you want to customize and open it in the Source view of Page Composer. The page's structure is displayed, and from here, you can identify the page name, or if the customization is actually on a page fragment within a task flow, you can identify the task flow name. For more information about using Page Composer, see the "Page Composer: Customizing Oracle Fusion CRM Applications" chapter in the *Oracle Fusion Applications CRM Extensibility Guide* and the "Editing a Page" chapter in the *Oracle Fusion Middleware User's Guide for Oracle WebCenter Portal: Spaces*.

2. If you need to customize a page fragment (`.jsff`) file within a task flow, from Page Composer, click **Manage Customizations** to open the page in the Manage Customizations dialog. From here, you can identify the `.jsff` file name.
3. In JDeveloper, after you have created a connection to the exploded EAR directory, you can use the Filter Customizable Archive dialog to search for the `.jsff` file or the task flow file.
4. Right-click the file and choose **Customize** to import the file and open it in JDeveloper.
5. Right-click the file, and choose **Go to Page Definition**.

   The page definition file shows you the view objects being used by the components on the page to return the data.
6. Open the view object in JDeveloper.

   The view object can be customized, or if needed, you can identify the associated entity object and customize that. Note that you can also identify the application module from here.

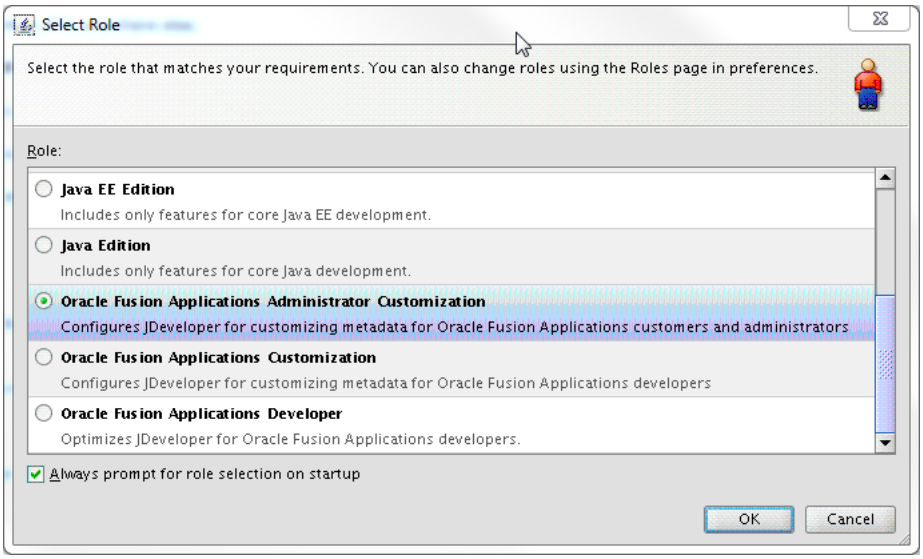
When you have identified the artifacts you want to customize, take note of the top-level page that contains the artifacts. You will need to know the name of the page to run when testing your customizations. For more information, see Section 3.2.5, "Running Customizations Locally."

3.2.3 Customizing the Artifacts

You must switch to the Customization Developer role before you can begin customizing.

1. Restart JDeveloper and select the **Oracle Fusion Applications Administrator Customization** role, as shown in Figure 3–5.

Figure 3–5 Oracle Fusion Applications Administrator Customization Role



2. In the Application Resources panel, expand **Connections**, and then **Customizable Archive**.
3. To locate the artifact you want to customize, right-click **Base Application Archive** and choose **Filter**.

For help in determining which artifacts you need to customize, see Section 3.2.2, "Determining Which Oracle ADF Artifacts You Need to Customize."

4. In the Filter Customizable Archive dialog, enter the file name of the artifact you want to customize, and click the **Go** icon.

   When the file is located, it is displayed in the Application Resources panel.

   Note that sometimes when customizing an ADF Business Components object, you will find two results entries for a given object. For example, when searching for `LookupValuesVO.xml`, you might find both of the following results:

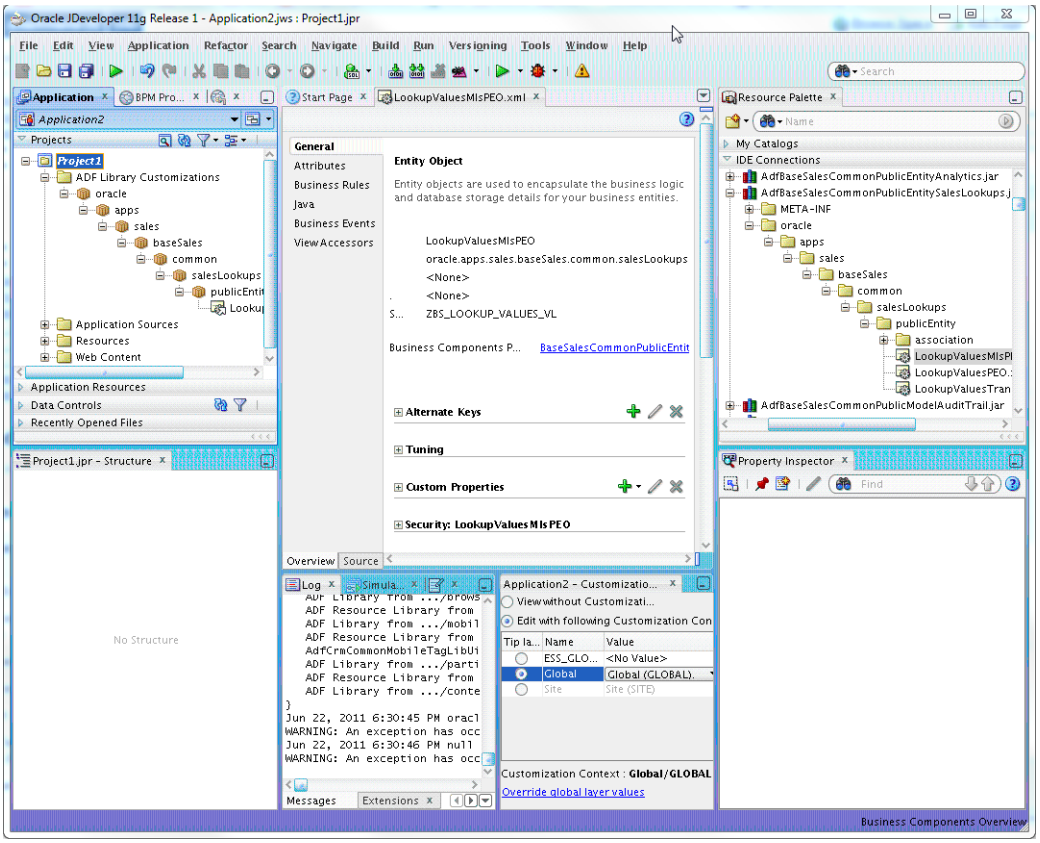
   ```
   jdev.rc:%40scratch%40jdeveloper%40mywork%40FASalesApp%40FASalesAp
   p.jws/Base+Application+Archive+-+FASalesApp/APP-INF/lib/AdfBaseSa
   lesCommonPublicModelSalesLookups.jar/oracle/apps/sales/baseSales/
   common/publicModel/salesLookups/view/LookupValuesVO.xml

   jdev.rc:%40scratch%40jdeveloper%40mywork%40FASalesApp%40FASalesAp
   p.jws/Base+Application+Archive+-+FASalesApp/APP-INF/lib/AdfBaseSa
   lesCommonModelSalesLookups.jar/oracle/apps/sales/baseSales/common
   /salesLookups/salesLookupService/view/LookupValuesVO.xml
   ```

   When you encounter this situation, use the non-service object for customization. That is, customize the object that does not have a service named package in its path. In this example, you would customize the first object because the second object has `salesLookupService` in its path.

5. Right-click the artifact, choose **Customize**, and choose to add the associated library to the project.

   The artifacts from the imported library now display in the Application Navigator, under the **ADF Library Customizations** node, and the artifact you selected to customize opens in the editor window, as shown in Figure 3–6.

Figure 3–6 JSPX Page Open in Editor and Ready to Customize



Note: If imported data controls are not displayed in the Data Controls panel, do the following:

1. From the JDeveloper main menu, choose **Tools > Preferences** to open the Preferences dialog.
2. Expand the **Business Components** node.
3. Select **General**.
4. Select **Display Imported ADF Libraries in Data Control Palette**.

---

6. In the Customization Context window (by default, displayed at the bottom of JDeveloper), select the layer to which you want the customizations written.

   All customizations for Oracle ADF business components must be done in the **global layer**. View layer customizations can be made in any other layer except User. For more information about customization layers, see Section 1.2, "Understanding Customization Layers."

You are now ready to begin customizing your artifact. For more information about customizing specific artifacts, see the following chapters:

- Chapter 4, "Customizing and Extending Oracle ADF Application Artifacts"
- Chapter 7, "Customizing and Extending Oracle Enterprise Scheduler Jobs"
- Chapter 8, "Customizing Security for Oracle ADF Application Artifacts"
- Chapter 10, "Configuring End-User Personalization"

- Chapter 11, "Customizing Help"

3.2.4 Avoiding Conflicts Among Multiple Customization Developers

When working in teams of multiple developers implementing multiple customizations in an application, observe the following guidelines to avoid conflicts of customized metadata:

- Create small, focused customization application workspaces for the application, logically divided among functional areas.
- Ensure that any given artifact is customized in only one customization application workspace.
- Share the customization application workspaces among customization developers, ensuring that only one developer at a time implements customizations in the workspace.
- Make sure that all custom JAR files have different names that begin with the prefix `Xx`, for example `XxMyJar.jar`.

By following these guidelines, you can avoid situations where developers inadvertently overwrite each others customizations, and make sure that customizations don't interfere with product upgrades.

3.2.5 Running Customizations Locally

You can use JDeveloper to run applications in Integrated WebLogic Server. To accomplish this, you need to identify a runnable target that contains the customized object.

When identifying which artifact to customize, you typically start by opening the page that exposes that artifact, and then drill down to identify the specific object to customize. The top-level page you start at (typically a `.jspx` file) is the page you will run to test customizations, so it is important to take note of the name of that file when you begin customization. Page fragments (`.jsff` files) are not runnable objects.

You will also need to make sure that your local test environment has the necessary security configuration to run the application. For more information, see Chapter 8, "Customizing Security for Oracle ADF Application Artifacts." In particular, you will need to consult a security administrator to export all predefined function security policies of the application that you are customizing into a `jazn-data.xml` file. For details about how the security administrator exports the application policy store, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

For more information about running locally, see the "Running a Fusion Web Application in Integrated WebLogic Server" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. For more information about identifying an artifact to customize see Section 3.2.2, "Determining Which Oracle ADF Artifacts You Need to Customize."

3.2.6 Importing Customizations into Your Application Workspace

There may be occasions when you need to import other customizations into your application workspace. For example, someone else may have made customizations to an application module to which you need to make changes as well. Before you make your customization, you must import that application module into your customization application workspace.

If you need to import customizations made to a single page or page fragment, you can use the Manage Customizations dialog to download the file, as described in the "Using the Manage Customizations Dialog to Download Customizations" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*. Save the customization files to a zip or JAR file.

If you need to import multiple customizations available in the metadata repository for an application, you use the `exportMetadata` Oracle WebLogic Scripting Tool (WLST) command. For more information, see the "Application Metadata Management Commands" section of *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*. This command saves the customization files in a JAR file that you can import into your application workspace. For procedures, see the "Viewing ADF Library Runtime Customizations from Exported JARs" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

> **Tip:** You can also use Oracle Enterprise Manager Fusion Applications Control to import and export customization files. For more information, see the "Transferring Metadata Using Fusion Middleware Control" section of the *Oracle Fusion Middleware Administrator's Guide*. The referenced procedure describes using Fusion Middleware Control, but also applies to Fusion Applications Control.

If you want to use extensions (for example, if you want to add a custom entity object to an existing application module), the extensions must be deployed into the environment to which you have a connection. For more information, see Section 4.14, "Deploying Oracle ADF Customizations and Extensions."

3.2.7 Resynchronizing Your Customization Application Workspace Configuration Files

During the process of customization, it is possible that the base application that you are customizing is updated with a patch. If this happens, you might need to resynchronize the configuration files in your local customization application workspace from the exploded EAR of the application you are customizing.

When you create a customization application workspace in your local development environment, workspace configuration files (such as, `adf-config.xml`, `connections.xml`, and `web.xml`) are copied to the local development environment. In some cases, the file is modified to allow you to implement and test customizations locally. When a patch is applied to the base application, these configuration files might change, and would therefore need to be synchronized to your local development environment so that you can continue to implement and test customizations.

JDeveloper allows you to check for and process updates to the workspace configuration files after a patch has occurred on the base application. When you run the check, there are three potential results for each file:

- The file in the local development environment does not need to be updated.
- The file in the local development environment must be updated, and can be updated safely because the local version has not been modified.
- The file in the local development environment must be updated, but cannot be updated safely because the local version has been modified.

After the check, JDeveloper lets you decide how to handle the update. If you choose to proceed with the updates, backups of the local files are created. You can use the backup files to manually merge changes into the updated files if necessary.

To synchronize your customization application workspace configuration files:

1. Start JDeveloper in the **Oracle Fusion Applications Administrator Customization** role, and open your customization application workspace.
2. From the main menu, choose **Application** > **Synchronize Patch Changes**.

   The check is run, and the Synchronize Patch Changes dialog displays the results.
3. If no files in the development environment need updating, the Synchronize Patch Changes dialog gives you the option to review the list of possible updates. Click **Yes** to view possible updates, or **No** to close the dialog.
4. If one or more files need to be updated, the Synchronize Patch Changes dialog displays the files that might be out of date. Files that have been modified locally are indicated with a green icon. Click **Yes** to update the files, or **No** to skip the updates and close the dialog.

Note: If you choose to proceed with the updates, backups of the local files are created. You can use the backup files to manually merge changes into the updated files if necessary.

3.3 Customizing SOA Composite Applications with JDeveloper

Before you begin customizing, you must identify the SOA archive (SAR) file to customize, retrieve the **configuration plan** from the default SOA composite application in MDS Repository, and set up the application workspace and SOA composite application project for MDS Repository customization in JDeveloper using the Oracle Fusion Middleware Developer role. After the application workspace is created, you must exit and reenter JDeveloper using the Oracle Fusion Applications Administrator Customization role.

3.3.1 Before You Begin Using JDeveloper to Customize

Perform the following tasks before you begin customizing SOA composite applications with JDeveloper:

1. Identify the **SAR file** to customize, and locate it in the *APPLICATIONS_BASE*/fusionapps/applications/*product_family*/deploy directory. This directory includes the following files:
   - Composite SAR (sca_*.jar)
   - Business process management (BPM) template (bta_*.jar)
   - List of resource bundle classes (jar_*.jar)
2. Ensure that the SAR file is marked as customizable by Oracle Fusion Applications. Otherwise, customizations do not survive patching or are not properly patched and merged. For information about which SOA composite applications are customizable, see the product-specific documentation from Oracle Enterprise Repository for Oracle Fusion Applications.

   If you encounter the following message when importing the SAR file for customization, it means that Oracle Fusion Applications did not mark the SOA composite application for customizations in JDeveloper and your changes cannot survive patching.

   ```
   The composite from the archive was not created for
   customization. If you import the composite for
   customization, you can customize it but you will have
   ```

```
problems to merge your customizations to any new
versions of that composite.
Do you want to continue?
Otherwise, uncheck "Import for Customization" box, and
click "Finish" option.
```

3. Get the configuration plan from the default SOA composite application in MDS Repository using the following WLST commands:

   a. Identify the default revision of the SOA composite application with `sca_getDefaultCompositeRevision`. For example:

   ```
   wls:/mydomain/ServerConfig> sca_getDefaultCompositeRevision("myhost",
   "7001", "weblogic", "weblogic",
   "FinGlCurrencyUserPreferredCurrencyComposite")
   ```

   b. Export the full SOA composite application corresponding to the default revision with `sca_exportComposite`. For example:

   ```
   wls:/offline/mydomain/ServerConfig> sca_
   exportComposite('http://myhost:8001', 'none', '/tmp/sca_
   FinGlCurrencyUserPreferredCurrencyComposite.0.jar',
   'FinGlCurrencyUserPreferredCurrencyComposite',
    '1.0')
   ```

   c. Extract the configuration plan used originally with the export action with `sca_extractPlan`. For example:

   ```
   wls:/mydomain/ServerConfig> sca_extractPlan("/tmp/sca_
   FinGlCurrencyUserPreferredCurrencyComposite_
   rev1.0.jar", "/tmp/FinGlCurrencyUserPreferredCurrencyComposite_
   configPlan.xml")
   ```

For information about using these commands, see the "Oracle SOA Suite Custom WLST Commands" section of *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*.

3.3.2 Setting Up the JDeveloper Application Workspace and SOA Composite Application Project for MDS Repository Customization

You must set up the JDeveloper application workspace and SOA composite application project for MDS repository customization.

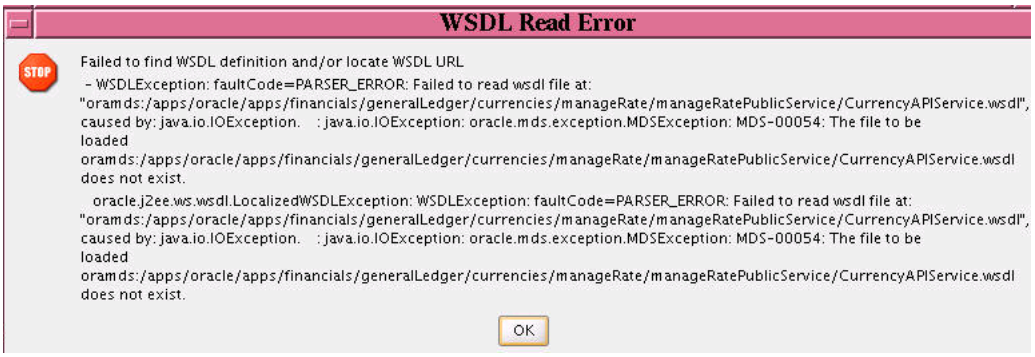
To set up the JDeveloper application workspace and SOA composite application project for MDS Repository customization:

1. Start JDeveloper using the **Oracle Fusion Applications Developer** role.
2. From the **File** main menu, choose **New**, then **Applications**, then **SOA Application**, and then **OK** to create a SOA application with an `XX` prefix in the application name.

   The `XX` prefix identifies a SOA artifact or object created by the customer and distinguishes it from Oracle Fusion Applications artifacts. You can delete the SOA project named **Project1** that was created by default.

   The Oracle Fusion Applications SOA composite application references shared SOA artifacts through the SOA-shared repository stored in MDS Repository instead of replicating the shared SOA artifact throughout the Oracle Fusion Applications code source. If the references to the SOA shared repository are not resolved, you receive the error message shown in Figure 3–7.

Figure 3–7 WSDL Read Error Message



3. To resolve references to the SOA-shared repository (`oramds:/apps`), define an MDS Repository entry in the `adf-config.xml` file. The entry must point to the MDS repository of the SOA Infrastructure partition in the MDS Repository schema corresponding to the SOA cluster where you plan to deploy the customized SOA composite application. Add a `<namespace>` attribute with `path="/apps"` to `<metadata-namespaces>`:

   ```
   <namespace metadata-store-usage="mstore-usage_2" path="/apps"/>
   ```

4. Add a `<metadata-store-usage>` attribute to `<metadata-store-usages>` for a database-based MDS Repository that points to the MDS repository of the SOA Infrastructure partition in the MDS repository for SOA schema.

5. Replace the database schema name, database server, database port, and database name with actual values. To identify the user name, password, and database connection information, see the configuration for the MDS data source for SOA in Oracle WebLogic Server Administration Console.

   ```
   <metadata-store-usage id="mstore-usage_2">
      <metadata-store class-name="oracle.mds.persistence.stores.db.
       DBMetadataStore">
         <property value="soa_mds_schema_name" name="jdbc-userid"/>
         <property value="soa_mds_schema_password" name="jdbc-password"/>
         <property value="jdbc:oracle:thin:@database_server:
           database_port:database_name" name="jdbc-url"/>
         <property value="soa-infra" name="partition-name"/>
      </metadata-store>
   </metadata-store-usage>
   ```

   The following code shows an `<adf-mds-config>` example in the `adf-config.xml` file. The `mstore-usage_2` entry resolves references to the SOA shared repository:

   ```
   <adf-mds-config xmlns="http://xmlns.oracle.com/adf/mds/config">
     <mds-config xmlns="http://xmlns.oracle.com/mds/config">
       <persistence-config>
         <metadata-namespaces>
           <namespace metadata-store-usage="mstore-usage_1" path="/soa/shared"/>
           <namespace metadata-store-usage="mstore-usage_2" path="/apps"/>
         </metadata-namespaces>
         <metadata-store-usages>
           <metadata-store-usage id="mstore-usage_1">
             <metadata-store
    class-name="oracle.mds.persistence.stores.file.FileMetadataStore">
               <property value="${oracle.home}/integration"
                         name="metadata-path"/>
   ```

```
            <property value="seed" name="partition-name"/>
          </metadata-store>
        </metadata-store-usage>
        <metadata-store-usage id="mstore-usage_2">
          <metadata-store
 class-name="oracle.mds.persistence.stores.db.DBMetadataStore">
            <property value="FIN_FUSION_MDS_SOA" name="jdbc-userid"/>
            <property value="FIN_FUSION_MDS_SOA" name="jdbc-password"/>
            <property
 value="jdbc:oracle:thin:@database_server.us.example.com:1521:database_name"
                      name="jdbc-url"/>
            <property value="soa-infra" name="partition-name"/>
          </metadata-store>
        </metadata-store-usage>
      </metadata-store-usages>
    </persistence-config>
  </mds-config>
</adf-mds-config>
```

6. From the **File** main menu, choose **Import**, then **SOA Archive Into SOA Project** to import the SAR file, and then click **OK**.
7. In the **Project Name** field, enter the name of the new SOA project with an `XX` prefix and select a name to identify the base SOA composite application that you are extending. For example, specify `XXFinGlCurrencyUserPreferredCurrencyComposite` if you are customizing `FinGlCurrencyUserPreferredCurrencyComposite`.
8. Click **Next**.
9. In the **Composite Archive** field, perform the following steps:
   a. Click **Browse** to select the SAR file to customize that you identified in Section 3.3.1, "Before You Begin Using JDeveloper to Customize."
   b. Accept the default setting for the SOA composite application name.
   c. Select the **Import for Customization** checkbox.
   d. Click **Finish**.

   Accept the default SOA composite application name to ensure that patching and Oracle SOA Suite can identify whether runtime customizations, JDeveloper customizations, or both types have been applied to the SOA composite application. If the SOA composite application is renamed, patching and SOA have no knowledge of the renamed SOA composite application.

   You may see an error icon on a **partner link** in Design view of the `composite.xml` file that reports the following error:

   ```
   Couldn't resolve classpath:/META-INF/wsdl/ServiceException.wsdl
   ```

   This error is addressed in subsequent steps.
10. Right-click the SOA composite application project and go to **Project Properties**, then **Libraries and Classpath**.
11. Click **Add Library**, and select the **BC4J Service Client** library.
12. Click **OK** to close the Add Library dialog.
13. Click **OK** to close the Project Properties dialog.

By adding this library to your SOA project, you avoid the design time error you may have received in Step 9d.

14. Click the **Validate** icon in the Design view of the `composite.xml` file. The error shown in Step 9d that you may have received for the partner link should now be resolved.

15. Make the customization classes and values available in your project.

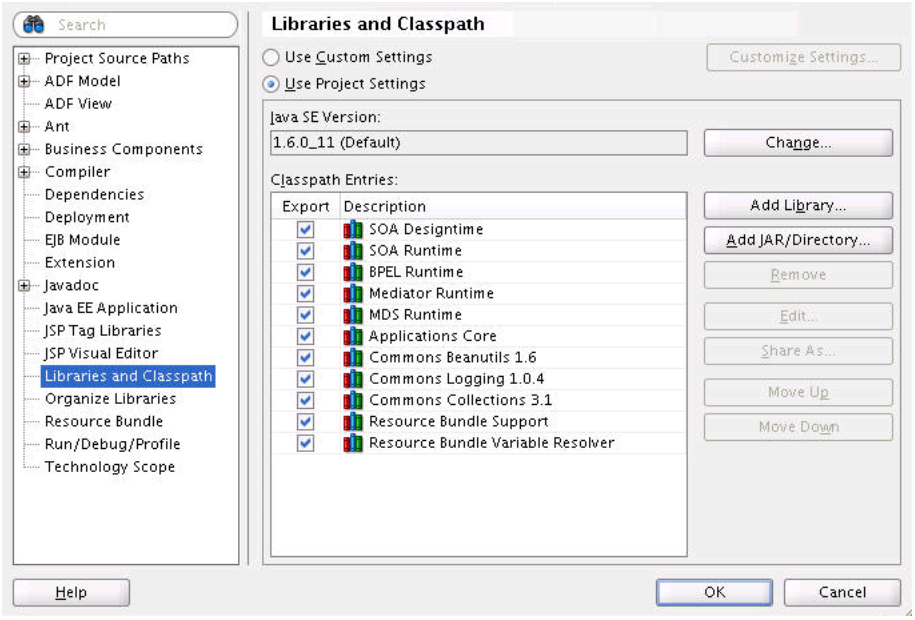
    There are two types of customization classes:

    - Applications Core customizing classes are available from the Applications Core shared library. See Section 1.2, "Understanding Customization Layers" for the list of Applications Core customization classes permitted in JDeveloper.
    - Product team customization classes are available in the appropriate EAR file. These customization classes are bundled in a JAR file in the EAR's `APP-INF/lib` directory. These JAR files follow a naming convention of `Ext...jar`. Therefore, you must get these JAR files from the deployed area, and perform the following steps:
      - Put the customization class JAR file under `$JDEV_HOME/jdev/extensions`.
      - Add the JAR file in the new project's library and class path setting.

16. Right-click the SOA composite application project and go to **Project Properties**, then **Libraries and Classpath**.

17. Add the **Applications Core** library to the SOA composite application project, as shown in Figure 3–8.

Figure 3–8 Applications Core Library



18. Go to **Application Resources**, then **Descriptors**, then **ADF META-INF**, and then **adf-config.xml**.

19. Add the appropriate customization class in the MDS Repository configuration, such as `oracle.apps.fnd.applcore.customization.SiteCC`.

20. Right-click the SOA composite application project and go to **Project Properties**, then **Libraries and Classpath**.

    The following libraries have now been added:

    - Application Core
    - BC4J Service Client

3.3.3 Customizing the SOA Composite Application

You must customize the SOA composite application.

To customize the SOA composite application:

1. Start JDeveloper using the **Oracle Fusion Applications Administrator Customization** role.
2. Select the value for the layer in the Customization Context dialog that you want to customize. Figure 3–9 provides details.

Figure 3–9 Customization Context Dialog

3. See Chapter 5, "Customizing and Extending SOA Components" for instructions about customizing the SOA composite application during design time in JDeveloper and runtime with Oracle SOA Composer, Oracle BPM Worklist, and Oracle Enterprise Manager Fusion Applications Control.
4. When introducing new components, partner links, and SOA artifacts to the SOA composite application, add the `XX` prefix to the name to prevent problems with existing and future components that may be introduced in Oracle Fusion Applications patches.
5. Use the configuration plan that you extracted in Step 3 of Section 3.3.1, "Before You Begin Using JDeveloper to Customize." If any new partner links were added to your SOA composite application, add entries to the configuration plan, if needed. For information about configuration plans, see the "Customizing Your Application for the Target Environment Prior to Deployment" section of the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.
6. Deploy the SOA composite application using the same revision you found in Section 3.3.1, "Before You Begin Using JDeveloper to Customize."

3.3.4 Customizing SOA Resource Bundles

Table 3–1 describes how to customize SOA resource bundles.

Table 3–1 Customizing SOA Resource Bundles

To Modify or Add Translatable Strings for...	Description
An existing human task, activity guide, or BPEL process	This feature is not supported in the first version of Oracle Fusion Applications (for example, modifying the human task title). Runtime modifications do not support this functionality and the `.task`, `.ag`, and `.agdl` files are not customizable in JDeveloper.
Human task mapped attributes	This feature is not supported in the first version of Oracle Fusion Applications. Translations for human task mapped attribute labels are defined in the following resource bundle: `oracle.bpel.services.workflow.resource.WorkflowLabels` For this version, the `WorkflowLabels` resource bundle is deployed to the SOA clusters in the Customer Relationship Management and Human Capital Management domains. Any customizations to the resource bundle are overwritten with future patches.
Server name in Federated Worklist on the Oracle Fusion Applications home page	The server names that appear in the Federated Worklist on the Oracle Fusion Applications home page are defined in the following file: `oracle/apps/common/acr/resource/ResourcesAttrBundle.xliff` See Section 4.12, "Customizing or Adding Resource Bundles" for instructions about overriding strings in XML localization interchange file format (XLIFF) resource bundles.

Note: Oracle Fusion Applications automatically seed human task-protected mapped attributes and labels, but do not seed public mapped attributes. If you require human task mapped attributes, it is recommended that you use the public mapped attributes. However, if protected mapped attributes are required, then add the `XX` prefix to your label names to prevent problems with Oracle Fusion Applications seeded labels.

4

Customizing and Extending Oracle ADF Application Artifacts

This chapter describes how to use Oracle JDeveloper to **customize** and **extend** application artifacts defined by Oracle Application Development Framework (Oracle ADF) in Oracle Fusion applications.

This chapter includes the following sections:

4.1 About Customizing Oracle ADF Application Artifacts

With the customization features provided by Oracle Metadata Services (MDS), developers can customize Oracle Fusion Applications using JDeveloper, making modifications to suit the needs of a particular group, such as a specific country or site.

Using JDeveloper, you can implement customizations on existing artifacts that are stored in a metadata repository and retrieved at runtime to reveal the customized application. You can also extend you application with new custom artifacts that are packaged into a JAR file, and integrated using customizations on the existing application.

Note that many kinds of customizations can be performed in the runtime environment using Oracle Fusion CRM Application Composer, which allows you to customize existing objects and extend an application with new objects for the following CRM applications:

- Sales
- Marketing
- Customer Center
- Trading Community Architecture
- Order Capture

For more information about using Application Composer to customize these applications, see the *Oracle Fusion Applications CRM Extensibility Guide*.

However some kinds of customization (including all customizations to applications other than those listed above) require a lower level approach, for which you will need to use JDeveloper.

4.1.1 Before You Begin Customizing Oracle ADF Application Artifacts

Before you customize application artifacts (such as entity objects, view objects, application modules, and pages) using JDeveloper, you should be familiar with the Oracle Fusion application architecture that enables customization, as described in Chapter 1, "Customizing and Extending Oracle Fusion Applications." You should also understand the typical workflows for working with customizations, as described in Chapter 2, "Understanding the Customization Development Lifecycle."

Before you make any changes to the data model such as adding entity objects or attributes, first check to see if there is an existing **flexfield** that meets your needs. For more information, see the "Flexfields: Overview" section in the *Oracle Fusion Applications Common Implementation Guide*.

> **WARNING: Do not use JDeveloper to customize flexfields. If you require flexfield changes that you cannot accomplish using the Manage Flexfields tasks or the Manage Value Sets tasks as described in the "Define Flexfields" section in the *Oracle Fusion Applications Common Implementation Guide*, contact My Oracle Support at `https://support.oracle.com` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.**

In addition, you will need to perform the following tasks before you can begin customizing your application:

- Set up a test environment.

  All application artifact customizations should be deployed to a test environment. For more information, see Chapter 2, "Understanding the Customization Development Lifecycle."

- Determine which artifacts you want to customize.

  Before you can implement customizations using JDeveloper, you must first determine which **business object**s you want to customize, so that you can create a customization application workspace that imports the necessary parts of the

application. For more information, see Section 3.2, "Customizing Oracle ADF Artifacts with JDeveloper."

- Create an application workspace.

  Before you can implement customizations using JDeveloper, you must create a customization application workspace that imports the necessary parts of the application you want to customize. For more information, see Section 3.2.1, "Creating the Customization Application Workspace."

- Start JDeveloper in the appropriate role.

  If you are implementing customizations on existing application artifacts, you must select the **Oracle Fusion Applications Administrator Customization** role when you start JDeveloper.

  If you are creating new custom application artifacts (such as entity objects, view objects, and pages), you must select the **Oracle Fusion Applications Developer** role when you start JDeveloper.

- Select a layer value.

  When customizing application artifacts in JDeveloper, you must first select the layer and layer value to work in. You use the Customization Context window to make this selection. When customizing business components, such as entity objects and view objects, you must use the **global layer**. For more information about customization layers, see Section 1.2, "Understanding Customization Layers."

4.1.2 Customizing at the Role Level

The layers and layer values that are available depend on which application you are customizing, and the `Role` layer is not available in all Oracle Fusion applications. CRM applications have a `Role` layer that you can select in the Customization Context window, while other applications do not.

To implement role-level customizations in an application that does not have a `Role` layer, you can use an expression in the rendered property of a component to conditionally render the component based on specified security level associated with a role. For example, you might want to display a button or a column in a table only to users that have the role of manager.

To conditionally render a button based on a role, start JDeveloper in the **Oracle Fusion Applications Administrator Customization** role, and then select the layer in which to customize. (Remember that if you want to customize an ADF Business Components object, such as an entity object or view object, then you must use the `global` layer.) Open the page that contains the button, and then in the Properties window, click the **Property Menu** icon beside the **Rendered** field and select **Expression Builder**. In the Expression Builder dialog, enter an EL expression like the one in Example 4–1.

Example 4–1 EL Expression to Verify Security

```
#{securityContext.userGrantedPermission['FND_APP_MANAGE_STANDARD_LOOKUP_PRIV']}
```

Note that the EL expression in Example 4–1 assumes that you have a security permission called `FND_APP_MANAGE_STANDARD_LOOKUP_PRIV` already defined. This expression evaluates to `False` for users that have not been granted the `FND_APP_MANAGE_STANDARD_LOOKUP_PRIV` permission, and the button is not rendered in the page. For more information, see the "Enabling ADF Security in a Fusion Web

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

4.2 Editing Existing Business Components

When customizing an application in JDeveloper, be aware that the layer in which you choose to implement customizations has an impact on what kinds of customizations you can perform. If you want to customize an ADF Business Components object, such as an entity object or view object, then you must use the `global` layer.

Before you begin:

Before you start customizing business objects, you will need to determine which business objects you want to customize and create a customization application workspace. For more information, see Section 4.1.1, "Before You Begin Customizing Oracle ADF Application Artifacts."

Then when customizing ADF artifacts, you need to start JDeveloper in the **Oracle Fusion Applications Administrator Customization** role, and then select the global layer.

Task: Edit Attributes

You can customize the properties of an attribute from an entity object or view object using JDeveloper. When you open an entity object or view object in the overview editor, you click the **Attributes** navigation tab to see the attributes of the object. When you select an attribute, its properties are displayed in the Property Inspector.

It is not necessary to modify the page after customizing the properties of an existing attribute. Customizations to existing attributes are automatically reflected on the pages that show them.

However, if you modify an attribute so that it requires a different UI component, then you must also update the page. For example, if you add a list of values (LOV) to an attribute, then you must edit the page to hide the existing UI component that displays the attribute, and add a new UI component that can display the LOV.

Note that some attribute properties defined in the entity object can be overridden in the view object. For example, you can define the label text for a field in an entity object and subsequently give it a different label in the consuming view object. Then pages that use the view object display the label from the view object.

For more information about attributes in entity objects, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Add Attributes

You can add custom attributes to an entity object or view object using JDeveloper. To do this, you must start JDeveloper in the **Oracle Fusion Applications Administrator Customization** role, and then select the global layer. When you open an entity object or view object in the overview editor, you click the **Attributes** navigation tab to see the attributes of the object. To add a custom attribute, click the **Add** icon.

If you want your custom attribute to be stored in the database, then you must first create the column that will be use to store it.

If you want your custom attributes to be displayed in the application, then you must also customize the pages to display them. For more information, see Section 4.4, "Editing Pages."

For more information about attributes in entity objects, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Edit Entity Objects

In JDeveloper, you edit entity objects using the overview editor. In the Application Navigator, right-click an entity object, and choose **Open**. Then click the navigation tabs to view and edit the various features of the entity object.

For more information about entity objects, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information about using entity objects in Oracle Fusion applications, see the "Getting Started with Business Services" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Edit View Objects

In JDeveloper, you edit view objects using the overview editor. In the Application Navigator, right-click a view object, and choose **Open**. Then click the navigation tabs to view and edit the various features of the view object.

For more information about view objects, see the "Defining SQL Queries Using View Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information about using view objects in Oracle Fusion applications, see the "Getting Started with Business Services" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Edit Validation

In JDeveloper, you edit declarative validation rules for entity objects and view objects using the overview editor. In the Application Navigator, right-click an entity object or view object, and choose **Open**. Then click the **Business Rules** navigation tab to view and edit the validation rules.

When implementing customizations on validation rules, you can add rules, modify the error message, and make rules more restrictive. But avoid removing rules or making rules less restrictive, because this can cause unpredictable results. Also, you can edit only declarative validation rules; programmatic validation rules cannot be customized.

For more information, see the "Defining Validation and Business Rules Declaratively" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Customize Business Logic Using Groovy Triggers

In JDeveloper, you can implement Groovy script to respond to predefined trigger points (such as, Before Delete in Database or After Create) for an entity object. These trigger points are available on the Business Rules page of the overview editor for entity objects.

In the Application Navigator, right-click an entity object, and choose **Open**. Then click the **Business Rules** navigation tab to view the existing validation rules. Click the **Add** icon and choose **Trigger** to display the Add Trigger dialog, which allows you to select a trigger point and enter a Groovy expression that will be executed in response to it. For more information, see the "Customizing Applications with MDS" chapter in the

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

Task: Edit Application Modules

In JDeveloper, you edit application modules using the overview editor. In the Application Navigator, right-click an application module, and choose **Open**.

In JDeveloper, you can make the following kinds of customizations on an application module:

- Add new custom properties. This is done on the General page of the overview editor.
- Add new view object and application module instances. This is done on the Data Model page of the overview editor.
- Add newly created subtype view objects. This is done on the Data Model page of the overview editor.
- Add new application module configurations. This is done on the Configurations page of the overview editor.

It is important to note that you cannot modify the web service interface for a service-enabled application module. You can, for example, add an attribute in a view object that is included in a service-enabled application module, but that attribute cannot be added to the web service interface.

For more information about working with application modules, see the "Implementing Business Services with Application Modules" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Add Customizations to Existing Reports

After you have implemented customizations on your application, you can use Oracle Business Intelligence Publisher to include these customizations in your reports. For more information, see the "Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

4.3 Editing Task Flows

You can use JDeveloper to implement customizations on the task flows that are used in your application. A task flow is a set of ADF Controller activities, control flow rules, and managed beans that interact to allow a user to complete a task. Although conceptually similar, a task flow is not the same as a **human task**, a task in the worklist, or a process flow.

A bounded task flow can be rendered in a JSF page or page fragment (`.jsff`) by using an ADF region. This is typically done to allow reuse of the task flow, as necessary, throughout the application. If you modify a bounded task flow, the changes apply to any ADF region that uses the task flow. For more information, see the "Using Task Flows as Regions" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Before you begin:

Before you start editing task flows, you will need to determine which task flows you want to customize, and create a customization application workspace. For more information, see Section 4.1.1, "Before You Begin Customizing Oracle ADF Application Artifacts."

When editing a task flow in JDeveloper, you must start JDeveloper in the **Oracle Fusion Applications Administrator Customization** role.

Task: Edit Task Flows

In JDeveloper, you use the task flow diagram editor to implement customizations on existing task flows. In the Application Navigator, right-click the task flow you want to customize, and choose **Open**. The page is displayed in the diagram editor, where you can make changes to the existing activities and control flow cases, or create new custom ones. For more information, see the "Getting Started with ADF Task Flows" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

4.4 Editing Pages

You can use JDeveloper to implement customizations on the pages that are used in your application. When editing a page in JDeveloper, you must start JDeveloper in the **Oracle Fusion Applications Administrator Customization** role.

Before you begin:

Before you start editing pages, you will need to determine which pages you want to customize, and create a customization application workspace. For more information, see Section 4.1.1, "Before You Begin Customizing Oracle ADF Application Artifacts."

Task: Edit Pages

In JDeveloper, you use the visual editor to implement customizations on existing pages. In the Application Navigator, right-click the page you want to customize, and choose **Open**. The page is displayed in the visual editor (accessed by clicking the **Design** tab). Then you can edit the page as you typically would using this editor. For more information about editing pages in JDeveloper, see the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

4.5 Creating Custom Business Components

You can use JDeveloper to extend your application by creating custom business components. When creating custom business components in JDeveloper, you must start JDeveloper in the **Oracle Fusion Applications Developer** role. This role is used for creating new custom objects that you want to add to the application. You can use the same application workspace that you created for customization. Then after you have created the custom business components, you switch to the **Oracle Fusion Applications Administrator Customization** role, to make changes to existing artifacts to integrate the new custom artifacts into the application.

Before you begin:

Before you start creating business objects, you will need to determine which business objects you want to create, and create a customization application workspace. For more information, see Section 4.1.1, "Before You Begin Customizing Oracle ADF Application Artifacts."

Task: Create Custom Entity Objects

An entity object represents a row in a database table, and encapsulates the business logic and database storage details of your business entities.

In JDeveloper, you can create entity objects using the Create Entity Object wizard, which you can access from the New Gallery. In the Application Navigator, right-click the project you want to add the entity object to, and choose **New**. Then in the New Gallery, expand **Business Tier**, click **ADF Business Components**, choose **Entity Object**, and click **OK**. Follow the prompts in the wizard to create an entity object.

For more information about creating entity objects, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information about using entity objects and view objects in Oracle Fusion applications, see the "Getting Started with Business Services" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Create Custom View Objects

A view object represents a SQL query and also collaborates with entity objects to consistently validate and save the changes when end users modify data in the UI.

In JDeveloper, you can create view objects using the Create View Object wizard, which you can access from the New Gallery. In the Application Navigator, right-click the project you want to add the view object to, and choose **New**. Then in the New Gallery, expand **Business Tier**, click **ADF Business Components**, choose **View Object**, and click **OK**. Follow the prompts in the wizard to create a view object.

For more information about creating view objects, see the "Defining SQL Queries Using View Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information about using entity objects and view objects in Oracle Fusion applications, see the "Getting Started with Business Services" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Create Custom Application Modules

An application module encapsulates an active data model and the business functions for a logical unit of work related to an end-user task.

In JDeveloper, you can create application modules using the Create Application Module wizard, which you can access from the New Gallery. In the Application Navigator, right-click the project you want to add the application module to, and choose **New**. Then in the New Gallery, expand **Business Tier**, click **ADF Business Components**, choose **Application Module**, and click **OK**. Follow the prompts in the wizard to create an application module.

For more information about creating application modules, see the "Implementing Business Services with Application Modules" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information about using application modules in Oracle Fusion applications, see the "Getting Started with Business Services" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Create a Web Service Interface for a Custom Application Module

In JDeveloper, you can edit a custom application module to create a web service interface that exposes the top-level view objects and defines the available service operations it supports. To do this, open the application module in the overview editor, click the **Service Interface** navigation tab, and click the **Enable support for Service Interface** icon. Then use the Create Service Interface wizard to configure the desired options.

It is important to note that the new web service cannot be deployed to the Oracle Fusion application. You can deploy it only to a new application.

For more information about creating a web service interface for an application module, see the "Integrating Service-Enabled Application Modules" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information about using application modules in Oracle Fusion applications, see the "Getting Started with Business Services" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Add Validation

In JDeveloper, you can create declarative validation rules for entity objects and view objects to help ensure the integrity of the data. To do this, open the entity object or view object in the overview editor, and click the **Business Rules** navigation tab. Then select the attribute you want to provide validation for, click the **Create new validator** icon, and use the Add Validation Rule dialog to configure the rule. For more information, see the "Defining Validation and Business Rules Declaratively" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Enforce Data Security for a Custom Business Object

You can use JDeveloper to enforce row and attribute security for custom ADF Business Components objects. This is done using **data security** policies to secure data from business objects based on the grants made to roles.

When you need to expose data records in an extended application, you can use JDeveloper to create entity objects based on secured database resources, and then opt into data security policies by enabling row-level **privilege** checking for specific operations on the entity objects. For more information, see Section 8.5, "Enforcing Data Security in the Data Model Project."

Task: Add a Business Object to an Existing Report

After you have extended your application with custom business objects, you can use Oracle Business Intelligence Publisher to include these extensions in your reports. For more information, see the "Customizing Reports and Analytics" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

4.6 Creating Custom Task Flows

You can use JDeveloper to create custom task flows that you can include in your application. A task flow is a set of ADF Controller activities, control flow rules, and managed beans that interact to allow a user to complete a task. Although conceptually similar, a task flow is not the same as a human task, a task in the worklist, or a process flow.

Before you begin:

Before you start creating custom task flows, you will need to determine which task flows you want to create, and create a customization application workspace. For more information, see Section 4.1.1, "Before You Begin Customizing Oracle ADF Application Artifacts."

When extending your application with custom task flows in JDeveloper, you must start JDeveloper in the **Oracle Fusion Applications Developer** role.

Task: Create a Custom Task Flow

You can create a custom task flow in JDeveloper using the New Gallery, and then define its activities using the task flow diagram editor. In the Application Navigator, right-click the project you want to add the task flow to, and choose **New**. Then in the New Gallery, expand **Web Tier**, and click **JSF/Facelets**. Then select **ADF Task Flow**, and click **OK**. In the Create Task Flow dialog, you'll specify the details about the type of task flow you want to create. When you click **OK**, the task flow is created and displayed in the diagram editor.

For information about creating and designing task flows, see the "Getting Started with ADF Task Flows" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

4.7 Creating Custom Pages

You can use JDeveloper to create custom pages that you can include in your application. When creating custom pages in JDeveloper, you must start JDeveloper in the **Oracle Fusion Applications Developer** role.

When creating the page (or dropping a view activity onto a task flow), you can create the page either as a JSF JSP or as a JSF JSP fragment. JSF fragments provide a simple way to create reusable page content in a project, and are what you use when you want to use task flows as regions on a page. When you modify a JSF page fragment, the JSF pages that consume the page fragment are automatically updated.

After extending your application with custom pages, you will need to make sure that security for the new pages is implemented appropriately and that the new pages are deployed so that they are accessible from the application. For more information about updating security, see Chapter 8, "Customizing Security for Oracle ADF Application Artifacts." For more information about deployment, see Section 4.14, "Deploying Oracle ADF Customizations and Extensions."

For more information about creating pages in JDeveloper, see the following resources:

- The *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*
- "Getting Started with Your Web Interface" in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- "Implementing UIs in JDeveloper with Application Tables, Trees and Tree Tables" in the *Oracle Fusion Applications Developer's Guide*
- "Implementing Applications Panels, Master-Detail, Hover, and Dialog Details" in the *Oracle Fusion Applications Developer's Guide*
- "Creating Customizable Applications" in the *Oracle Fusion Applications Developer's Guide*

Before you begin:

Before you start creating custom pages, you will need to determine which pages you want to create, and create a customization application workspace. For more information, see Section 4.1.1, "Before You Begin Customizing Oracle ADF Application Artifacts."

When creating custom pages in JDeveloper, you must start JDeveloper in the **Oracle Fusion Applications Developer** role.

Task: Create a Custom Page

In JDeveloper, you can create pages either by double-clicking a view activity in a task flow or by using the New Gallery. In the Application Navigator, right-click the project you want to add the page to, and choose **New**. Then in the New Gallery, expand **Web Tier**, and click **JSF/Facelets**. Then select either **Page** or **ADF Page Fragment**, and click **OK**.

Task: Add a Custom Page to a Task Flow

If you created the page by double-clicking a view activity in a task flow, then it is already added to the task flow. If you created it using the New Gallery, then you can add it to a task flow by dragging the page from the Application Navigator and dropping it in the task flow diagram editor. Then you can connect the page using a control flow. For more information, see the "Getting Started with ADF Task Flows" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Enable Runtime Customization for a Custom Page

To enable a custom page to be customized at runtime, you must make sure that the page and the project that contains it are set to allow runtime customizations. For information on how to do this, see the "How to Authorize the Runtime Customization of Pages and Task Flows" section in the *Oracle Fusion Applications Developer's Guide*.

4.8 Customizing and Extending the Oracle Fusion Applications Schemas

Using the database tools of your choice, you can customize and extend the Oracle Fusion Applications schemas to suit the needs of your organization. However, you should first consider using Application Composer or flexfields to satisfy your additional data storage requirements. For more information about using Application Composer, see the *Oracle Fusion Applications CRM Extensibility Guide*. For more information about using flexfields, see the "Flexfields: Overview" section in the *Oracle Fusion Applications Common Implementation Guide*.

4.8.1 About Customizing and Extending the Oracle Fusion Applications Schemas

If you need to extend the preconfigured Oracle Fusion Applications schemas to address additional data storage needs, create a custom schema. In your custom schema, you can create tables, columns, and all the necessary additional schema objects. This approach allows you to contain and maintain all of your custom data storage objects separately from the preconfigured Oracle Fusion Applications schemas.

If necessary, you can extend the preconfigured schemas within certain constraints. With the exception of customizing a preconfigured table to include new custom objects, such as columns, you must not make any customizations to preconfigured schema objects. Instead, you can extend the schema by adding new custom objects that you can configure as needed.

When making amendments to the schema, such as adding tables or columns, add a prefix to the name of the table or column that is a unique identifier (for example, `XX_`) to prevent collisions with existing objects.

Any code that accesses the new custom schema objects should use fully qualified table names (for example, *`SCHEMA_NAME.TABLE_NAME`*). If your code does not use fully qualified table names, you will need to create synonyms for the custom tables. The synonym must be created in the FUSION schema, and associated privileges must be granted in the FUSION_RUNTIME schema. At runtime, Oracle Fusion applications connect to the FUSION_RUNTIME schema, so privileges must be granted there.

However, because the schema context is set to FUSION, the synonym must be created there. This convention applies in all cases, whether you create custom schema objects in a custom schema or a preconfigured schema.

Note that if you are writing a custom application that is a peer application to an existing Oracle Fusion Applications module, you must define a custom schema that contains all the database objects for your custom application. Additionally, to integrate your custom application using a public Oracle Fusion Applications PL/SQL API, you must:

- Grant EXECUTE privilege to the custom schema on the Oracle Fusion Applications PL/SQL package.

  Public APIs in the Oracle Fusion Applications PL/SQL package are owned by the FUSION schema and are defined with the AUTHID DEFINER clause. This way when the custom schema invokes a PL/SQL API, the application will run with the set of privileges of the FUSION user, so there is no need to grant additional object privileges to the custom schema in order for the program to execute successfully.
- Refer to the package and its procedures and functions using a fully qualified name, for example: `FUSION.<package_name>.<procedure_name>`

If your custom application will interact with objects in the FUSION schema (for example interface tables), then you must also:

- Grant the necessary privileges on Oracle Fusion Applications objects to the custom schema (for example, INSERT privileges on interface tables) as necessary.
- Refer to objects in the FUSION schema with fully qualified names, for example: `FUSION.<table name>`

For information about creating database objects, see the Designing Databases topics in the JDeveloper online help.

4.8.2 What You Can Do with Schema Modifications

Using the SQL Worksheet in JDeveloper or the database tools of your choice, you can issue commands to the database to customize and extend it. When making changes to the database, you can do the following:

- Add a custom schema
- Add or modify tables
- Add columns to preconfigured or custom tables
- Add indexes to custom columns
- Add sequences
- Add PL/SQL packages, procedures, functions and abstract data types

4.8.3 What You Cannot Do with Schema Modifications

When making changes to the database, you cannot do any of the following:

- Modify preconfigured columns or sequences.
- Modify preconfigured PL/SQL packages, procedures, functions and abstract data types (unless explicitly directed to do so by product documentation).
- Delete preconfigured schema objects.

- Add indexes to preconfigured columns (unless explicitly directed to do so by product documentation).

4.8.4 Before You Begin Extending the Oracle Fusion Applications Schemas

Before you modify the Oracle Fusion Applications schema, you should first see if you can address your additional data storage requirements using flexfields, as described in the "Flexfields: Overview" section in the *Oracle Fusion Applications Common Implementation Guide*.

4.8.5 Extending the Schemas Using a Custom Schema

Using the SQL Worksheet in JDeveloper, you can issue commands to the database to customize and extend it. In a custom schema, you can add tables, columns, indexes, and other schema objects to support the customizations and extensions you want to implement in the application (such as, adding an attribute to an entity object).

To access the SQL Worksheet, right-click the database connection (under the **Connections** node in the Application Resources panel of the Application Navigator), and choose **Database Navigator** from the context menu. This will open the selected database connection in the Database Navigator and display the SQL Worksheet.

Before you begin:

Before you attempt to extend the schema, you should be familiar with the guidelines described in Section 4.8.1, "About Customizing and Extending the Oracle Fusion Applications Schemas."

Task: Create a Custom Schema

When creating a custom schema, add a prefix to the name of the schema that is a unique identifier (for example, `XX_`) to prevent collisions with existing schemas. You must grant the privileges to the custom schema that are necessary for it to function properly and for any supporting code to compile (for example, objects referenced in PL/SQL code).

Task: Create Custom Database Tables, Columns, Indexes, and Sequences

Within a custom schema, you can create custom database tables, columns, indexes, and sequences to address your additional data storage needs. When adding custom objects, add a prefix to the name of the object that is a unique identifier (for example, `XX_`) to prevent collisions with existing objects. New custom indexes and sequences should adhere to this convention as well.

After creating a custom table, you will need to grant the necessary object privileges to the FUSION_RUNTIME schema, which Oracle Fusion Applications uses at runtime. You can grant privileges directly to the schema, or through a custom database role, but do not use the preconfigured FUSION_APPS_READ_AND_WRITE database role.

Any code that accesses the new custom schema objects should use fully qualified table names (for example, *`SCHEMA_NAME.TABLE_NAME`*). If your code does not use fully qualified table names, then you will need to create synonyms for the custom tables, as described in Section 4.8.1, "About Customizing and Extending the Oracle Fusion Applications Schemas."

Task: Create Custom PL/SQL Packages, Procedures, Functions, and Abstract Data Types

When adding PL/SQL objects and abstract data types to a custom schema, add a prefix to the name of the object or data type that is a unique identifier (for example, `XX_`) to prevent collisions with existing objects.

Your PL/SQL code should contain the `AUTHID INVOKER` clause so that the code is executed within the context of the privilege set of the FUSION_RUNTIME user. Additionally, the FUSION_RUNTIME user must be granted the EXECUTE privilege on the PL/SQL object or type, either directly or through a database role.

If you need to create synonyms to support your PL/SQL code, then create your synonyms in the FUSION schema, as described in Section 4.8.1, "About Customizing and Extending the Oracle Fusion Applications Schemas."

4.8.6 Extending a Preconfigured Schema

Using the SQL Worksheet in JDeveloper, you can issue commands to the database to customize and extend it. When making changes to the schema, you can add tables or columns to support the customizations and extensions you want to implement in the application (such as, adding an attribute to an entity object). However, do not remove tables or columns, because this can have adverse affects in other parts of the application.

With the exception of customizing a preconfigured table to include new custom objects, such as columns, you must not make any customizations to preconfigured schema objects.

To access the SQL Worksheet, right-click the database connection (under the **Connections** node in the Application Resources panel of the Application Navigator), and choose **Database Navigator** from the context menu. This will open the selected database connection in the Database Navigator and display the SQL Worksheet.

Before you begin:

Before you implement extensions to a preconfigured schema, consider creating your extensions in a custom schema. This approach provides greater flexibility and modularity.

Also, you should be familiar with the guidelines described in Section 4.8.1, "About Customizing and Extending the Oracle Fusion Applications Schemas."

Task: Edit Database Tables

With the exception of customizing a preconfigured table to include new custom objects, such as columns, you must not make any customizations to preconfigured schema objects.

When adding columns to a preconfigured table, add a prefix to the name of the column that is a unique identifier (for example, `XX_`) to prevent collisions with existing columns.

Task: Create Custom Database Tables, Columns, Sequences, and Indexes

You can create custom database tables and columns to address your additional data storage needs. When adding custom tables and columns, add a prefix to the name of the table and columns that is a unique identifier (for example, `XX_`) to prevent collisions with existing tables and columns.

After creating a custom table, you will need to grant the necessary object privileges to the FUSION_RUNTIME schema, which Oracle Fusion Applications uses at runtime.

You can grant privileges directly to the schema, or through a custom database role, but do not use the preconfigured FUSION_APPS_READ_AND_WRITE database role.

Any code that accesses the new custom schema objects should use fully qualified table names (for example, *SCHEMA_NAME.TABLE_NAME*). If your code does not use fully qualified table names, then you will need to create synonyms for the custom tables, as described in Section 4.8.1, "About Customizing and Extending the Oracle Fusion Applications Schemas."

You can create new custom indexes on custom columns, but do not attempt to create an index on a preconfigured column, unless explicitly directed to do so by product documentation.

Task: Create Custom PL/SQL Packages, Procedures, Functions, and Abstract Data Types

When adding PL/SQL objects and abstract data types, add a prefix to the name of the object or data type that is a unique identifier (for example, `XX_`) to prevent collisions with existing objects.

Your PL/SQL code should contain the `AUTHID INVOKER` clause so that the code is executed within the context of the privilege set of the FUSION_RUNTIME user. Additionally, the FUSION_RUNTIME user must be granted the EXECUTE privilege on the PL/SQL object or type, either directly or through a database role.

If you need to create synonyms to support your PL/SQL code, then create your synonyms in the FUSION schema, as described in Section 4.8.1, "About Customizing and Extending the Oracle Fusion Applications Schemas."

4.9 Customizing or Creating a Custom Search Object

In JDeveloper, you can customize and create saved searches and search forms for your application. To customize a search form or saved search in JDeveloper, you will need to set up an application workspace as described in Section 3.2.1, "Creating the Customization Application Workspace." Then, locate and open the object you want to customize. To create a new search form, you open or create the page that will display the form and select a data collection from the Data Controls panel. For more information, see the "Creating ADF Databound Search Forms" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

4.10 Editing the UI Shell Template

To edit the **UI Shell template** in JDeveloper, you will need to set up an application workspace as described in Section 3.2.1, "Creating the Customization Application Workspace." Then, in the **Oracle Fusion Applications Administrator Customization** role, expand the contents of the **Applications Core (ViewController)** library and drill down to the file `oracle/apps/fnd/applcore/templates/UIShell.jspx`. This is the UI Shell template, which you can customize as necessary.

Alternatively, you can access the UI Shell template from any page in the library. Open the page in JDeveloper, right-click on the view ID of the `pageTemplate` tag (`/oracle/apps/fnd/applcore/templates/UIShell.jspx`), and then choose **Go to Declaration** to open the UI Shell template.

You can also use Page Composer to edit the UI Shell template, as described in the "Editing the UI Shell Template Used by All Pages" section in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*.

In addition, you can customize the Oracle Fusion Applications **skin** using ADF Skin Editor as described in Chapter 12, "Customizing the Oracle Fusion Applications Skin."

4.11 Customizing Menus

Using JDeveloper you can customize the menus in your Oracle Fusion applications. Customizing the tasklist menu follows the same pattern as editing most artifacts (such as, pages or business components) from the EAR connection. However, customizing the home page, preferences and navigator menus is slightly different. For these menus, you will need to export the menu's XML file from the MDS repository and copy them into your customization application workspace before you can implement customizations.

> **Note:** You can also customize the navigator menu at runtime from the Setup and Maintenance work area, as described in the "Managing Menu Customizations: Highlights" section in the *Oracle Fusion Applications Common Implementation Guide*.

To export the menu files for an application, you use the `exportMetadata` Oracle WebLogic Scripting Tool (WLST) command. For more information, see the "Application Metadata Management Commands" section of *Oracle Fusion Middleware WebLogic Scripting Tool Command Reference*. This command saves the files in a JAR file that you can import into your application workspace. For procedures, see the "Viewing ADF Library Runtime Customizations from Exported JARs" section of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

> **Tip:** You can also use Oracle Enterprise Manager Fusion Applications Control to import and export metadata files. For more information, see the "Transferring Metadata Using Fusion Middleware Control" section of the *Oracle Fusion Middleware Administrator's Guide*. The referenced procedure describes using Fusion Middleware Control, but also applies to Fusion Applications Control.

When exporting the menu XML files from the MDS repository, you can find them in the `oracle/apps/menu` directory in the repository. The following are their file names:

- Home page menu: `homepage_menu.xml`
- Preferences menu: `pref_menu.xml`
- Navigator menu: `root_menu.xml`

Then you copy the files to the same directory (under project source path) in your local customization application workspace (for example, *`CUSTOMIZATION_APP_PATH`*`/`*`PROJECT_NAME`*`/src/oracle/apps/menu`). After you have copied them into your local customization application workspace, you can customize the menus as necessary.

After you have implemented customizations on a menu, you will need to update the MAR profile to make sure they are included during deployment. In the MAR profile, under **User Metadata** > **Directories**, select the customizations you implemented that correspond to the menu files. For more information about deploying customizations, see Section 4.14, "Deploying Oracle ADF Customizations and Extensions."

For more information about menus in Oracle Fusion Applications, see the "Working with the Global Menu Model" section in the *Oracle Fusion Applications Developer's Guide*.

4.12 Customizing or Adding Resource Bundles

One method of customizing text is defining a new key in the **resource bundle**. There is a single override resource bundle for Oracle Fusion Applications. You can use this resource bundle to override values for existing keys, but you cannot add new keys.

Because you cannot define a new key in the shipped resource bundle, you need to create a new override bundle. You can accomplish this in JDeveloper by creating an XLIFF file from the New Gallery. After the file is generated, you can then enter new keys and their associated text in the XLIFF file.

To make the newly created resource bundle available for customization, you need to register the resource bundle with the customization project. You can do this from the Resource Bundle page of the Project Properties dialog.

You can also extend your application by creating a new resource bundle for a project if, for example, you want to customize the text for a label and you don't want to change the value in the global override bundle. To do this, create an XLIFF file from the New Gallery, package it into an ADF Library JAR file, and import the JAR file into the customization project.

Note: All custom JAR file names must begin with the prefix Xx, for example `XxMyJar.jar`.

To test your customizations locally in JDeveloper Integrated WebLogic Server, you must also include the ADF Library JAR file in the `APP-INF/lib` directory.

For information about translating custom resource bundle strings, see Section 9.2, "Translating Resource Bundles from an MDS Repository."

For more information about working with resource bundles, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

4.13 Extending Oracle Fusion Applications with a Custom Peer Application

Using JDeveloper, you can create a custom peer application to extend the functionality of an existing Oracle Fusion Applications module.

When you create the application, the package that you will use depends on how the functionality is integrated into the existing application.

- If the new custom functionality is completely separate and loosely integrated into the existing application (for example, adding a custom button or link to an existing page to go to a new custom page that exposes the custom peer application), then you will need to create a new package and deploy it separately from the existing application. On the Name page (step 1 of 5) of the Create Fusion Web Application wizard, you can specify any application package prefix that you want, but do not use `oracle.apps.cust`.
- If the new custom functionality is a tightly integrated extension of existing functionality (for example, adding new tasks to an existing task flow exposed in

an existing page), then make sure that you specify `oracle.apps.cust` as the application package prefix on the Name page (step 1 of 5) of the Create Fusion Web Application wizard. In this case, the customization application workspace MAR will pick up your metadata files from the included ADF libraries when you package and deploy the customizations.

After you create the application workspace, create an ADF Library deployment profile for each project in the peer application. Then when you package the peer application, ADF Library JAR files will be generated.

After you have created and packaged your custom peer application, you will need to place the ADF library JAR files into the customization application workspace that you are extending. The ADF library JAR for model artifacts (such as entity objects and view objects) should be placed into the *`ExplodedEarDirectory`*`/APP-INF/lib` directory. The ADF Library JAR for user interface artifacts (such as pages) should be placed in the *`ExploadedWarDirectory`*`/WEB-INF/lib` directory. For more information about packaging and deploying, see Section 4.14, "Deploying Oracle ADF Customizations and Extensions."

When you extend an Oracle Fusion Applications module with a custom peer application, you must define a custom schema that contains all the database objects for the custom application. For more information, see Section 4.8, "Customizing and Extending the Oracle Fusion Applications Schemas."

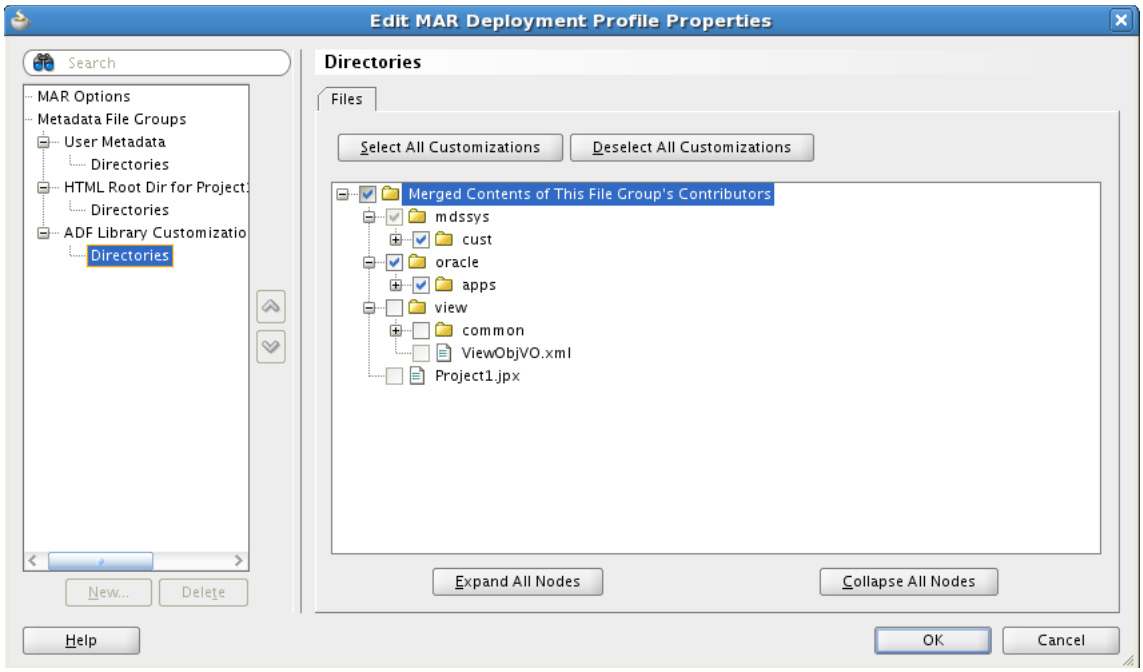
4.14 Deploying Oracle ADF Customizations and Extensions

After you have customized existing artifacts, you can use JDeveloper to deploy the customizations to a **sandbox** or to the Oracle Fusion application. For more information about how to use sandboxes to isolate changes from the mainline code so you can test and validate the changes, see the "Sandboxes: Explained" section in the *Oracle Fusion Applications CRM Extensibility Guide*.

When you create a customization application workspace as described in Section 3.2.1, "Creating the Customization Application Workspace," the wizard generates a MAR profile. By default, the name of the MAR profile is *`application_name`*`_customizations`. It will automatically include the customizations that you implement. You can use this profile to package your customizations for deployment.

When you package customizations from the customization application workspace, the MAR file should include only library customizations. If you have extensions, make sure to include those directories as well (for example, `oracle/apps`), as shown in Figure 4–1. Do not include the **User Metadata** or **HTML Root Dir for Project** in the MAR profile, unless explicitly directed to do so by product documentation.

Figure 4–1 MAR Deployment Profile Properties



If you extend your application with new custom artifacts, then you can use JDeveloper to package them into an ADF Library JAR and place them into the proper location within the application directory structure.

Task: Deploy the Customizations

You can use JDeveloper to deploy the customizations directly or you can use JDeveloper to create a MAR file, and then load the MAR file using WLST commands or the WebLogic Server Administration Console.

> **Tip:** You can also use Oracle Enterprise Manager Fusion Applications Control to import and export customization files. For more information, see the "Transferring Metadata Using Fusion Middleware Control" section of the *Oracle Fusion Middleware Administrator's Guide*. The referenced procedure describes using Fusion Middleware Control, but also applies to Fusion Applications Control.

If you are using JDeveloper to deploy directly, you have a choice to deploy to available sandboxes or into the already deployed Oracle Fusion application.

When you deploy customizations on ADF Business Component objects (such as entity objects and view objects), the server must be restarted for the customizations to be picked up.

For instructions on deploying customizations, see the "How to Deploy New Customizations Applied to ADF LIbrary" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Package New Artifacts into ADF Library

If you have extended your application with new custom artifacts (or you are supplied with new artifacts), then you must package these artifacts into an ADF library JAR and place the JAR files in the proper location within the application.

Note: All custom JAR files must begin with the prefix Xx, for example `XxMyJar.jar`.

The ADF library JAR for the new model artifacts (such as entity objects and view objects) should be placed into the *`ExplodedEarDirectory`*`/APP-INF/lib` directory (for example, `/fusionapps/applications/fin/deploy/EarFinPayables.ear/APP-INF/lib/XxMyJar.jar`). The ADF Library JAR for the new user interface artifacts (such as pages) should be placed in the *`ExploadedWarDirectory`*`/WEB-INF/lib` directory.

For instructions on creating ADF Library, see the "Packaging a Reusable ADF Component into an ADF Library" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

5

Customizing and Extending SOA Components

This chapter describes how to **customize** (edit) a service-oriented architecture (SOA) **component** during runtime in a deployed **SOA composite application** with a runtime tool such as Oracle Business Process Management Worklist (Oracle BPM Worklist), Oracle SOA Composer, or Oracle Enterprise Manager Fusion Applications Control or customize and **extend** (create) a SOA component during design time in Oracle JDeveloper. It also provides recommendations for merging runtime customizations from a previously deployed revision into a new revision and instructions for synchronizing a customized **flexfield** in Oracle Metadata Services (MDS) Repository.

This chapter includes the following sections:

For information about troubleshooting SOA issues, see the "Troubleshooting Oracle SOA Suite" chapter in the *Oracle Fusion Applications Administrator's Troubleshooting Guide*.

Note:

- This chapter does not describe customizing and extending Oracle Business Process Management Suite (Oracle BPM Suite). Oracle BPM Suite is installed on top of Oracle SOA Suite, and provides the ability to run Business Process Modeling and Notation (BPMN) processes. To accomplish this task, there are extensions to JDeveloper for working with BPMN (Oracle BPM Studio) and a web-based application for working with BPMN processes (Oracle Business Process Composer). For information about BPMN process flows, see the "Customizing and Extending BPMN Processes" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts*. For information about Oracle BPM project templates, see Chapter 6, "Customizing and Extending Oracle BPM Project Templates."
- Oracle SOA Suite extensions cannot be used with JDeveloper Integrated WebLogic Server. If an application has references to Oracle SOA Suite shared libraries, then customizations on the application cannot be tested with Integrated WebLogic Server.

5.1 About Customizing and Extending SOA Components

SOA provides an enterprise architecture that supports building connected enterprise applications to provide solutions to business problems. SOA enables you to develop enterprise applications as modular business web services that can be integrated and reused, resulting in a flexible, adaptable IT infrastructure. SOA separates business functions into distinct units, or services.

Oracle SOA Suite provides a complete set of service infrastructure components for designing, deploying, and managing SOA composite applications. A SOA composite application is a service, **service component**, and reference assembly designed and deployed in a single application. Wiring between the services, service components, and references enables message communication.

Oracle SOA Suite consists of SOA components that comprise the business logic and processing rules in a SOA composite application. You can include components such as the following in a SOA composite application:

- Business rules:

  The following **business rule** categories are available:

  – Approval configuration (expirations, escalations, and notifications) and assignment rules:

    Define complex task routing slips for approval management by taking into account business documents and associated rules to identify the approval hierarchy for a work item. Additionally, approval management lets you define multistage approvals with associated list builders based on supervisor or position hierarchies. You can also define expiration, escalation, and notification configurations. For example, an expense approval task may use rules to define its approvers.

    Approval configuration and assignment rules are within the context of a human workflow.

  – Nonapproval business rules:

Define a business decision based on rules that enables dynamic decisions to be made at runtime that automate policies, computations, and reasoning while separating rule logic from underlying application code. For example, you can define a business rule to select a supplier with the lowest shipping price to fulfill a shipping order.

Nonapproval business rules are in the context of Oracle SOA Suite, but outside of human workflow.

– Rules in non-Oracle SOA Suite applications

Use of standalone rules in non-Oracle SOA Suite applications is supported. You can completely control how the rule dictionaries are structured and how these applications are patched. You may structure the rules as recommended for Oracle SOA Suite rules, as described in this chapter.

A **rule dictionary** is a business rules container for facts, functions, globals, bucketsets, links, decision functions, and rulesets. A dictionary is an XML file that stores the application's rulesets and the data model. Dictionaries can link to other dictionaries. A **bucketset** enables you to define a list of values or a range of values of a specified type. After you create a bucketset, you can associate the bucketset with a fact property of a matching type. Business rules use the bucketsets that you define to specify constraints on the values associated with fact properties in rules or in a **decision table**. A **ruleset** is a business rules container for rules and decision tables. A ruleset provides a namespace, similar to a Java package, for rules and decision tables.

- **Domain value maps**:

  Operate on actual data values that move through the infrastructure at runtime. A **domain value map** enables you to map from one vocabulary used in a given domain to another vocabulary used in a different domain. For example, one domain can represent a city with a long name (Boston), while another domain can represent a city with a short name (BO). In such cases, you can directly map the values by using domain value maps.

- Human tasks:

  Extend a workflow that describes the tasks for users or groups to perform as part of an end-to-end business process flow. For example, a vacation request workflow is assigned to a manager. The manager must act on the request task three days before the vacation starts. If the manager formally approves or rejects the request, the employee is notified with the decision. If the manager does not act on the task, the request is treated as rejected. Notification actions similar to the formal rejection are taken.

- Business Process Execution Language (**BPEL**) processes:

  Integrate a series of business activities and services into an end-to-end process flow. For example, a **BPEL process** flow calls a credit rating service. When you run this process, you enter a social security number into a user interface. The credit rating service takes the number and returns a credit rating.

- Oracle Mediator:

  Defines services that perform message and event routing, filtering, and transformations. For example, Oracle Mediator can accept data contained in a text file from an application or service, transform it into a format appropriate for updating a database that serves as a customer repository, and then route and deliver the data to that database.

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

Oracle SOA Suite supports the following types of customizations and extensions of these components:

- Customizing several components during runtime
- Customizing and extending several components during design time

Oracle SOA Suite supports customizing several components during runtime.

The tool to use depends on the component you are customizing or extending and whether you are performing these tasks during runtime or design time. Table 5–1 provides details.

Note: If you are customizing approval configuration and assignment rules or nonapproval business rules for a *deployed* project (either for Oracle SOA Suite or Oracle BPM Suite), always use Oracle BPM Worklist or Oracle SOA Composer. If you are customizing approval configuration and assignment rules or nonapproval business rules as part of a *new* Oracle BPM Suite project being extended in Oracle Business Process Composer, then use Oracle Business Process Composer. For information about using Oracle Business Process Composer, see Chapter 6, "Customizing and Extending Oracle BPM Project Templates."

Table 5–1 Customization and Extension Tools for Oracle SOA Suite

To Perform These Tasks...	Use This Tool...	Use This Tool At...	Tool User
Customize business rules:			
■ Approval configuration and assignment rules	■ Oracle BPM Worklist (recommended) or Oracle SOA Composer	Runtime in a deployed SOA composite application	Technical analyst
■ Nonapproval business rules	■ Oracle SOA Composer **Note**: If you use Oracle SOA Composer to customize approval configuration and assignment rules during runtime, changes in subsequent revision patches are *not* applied successfully.	Runtime in a deployed SOA composite application	Business analyst
Customize domain value maps	Oracle SOA Composer	Runtime in a deployed SOA composite application	Business analyst
Customize SOA composite application endpoint properties such as the following: ■ Attached Oracle Web Services Manager (Oracle WSM) security policies ■ Service and reference **binding component** properties	Fusion Applications Control	Runtime in a deployed SOA composite application	System administrator

Table 5–1 (Cont.) Customization and Extension Tools for Oracle SOA Suite

To Perform These Tasks...	Use This Tool...	Use This Tool At...	Tool User
■ Customize or extend business rules ■ Customize or extend BPEL processes ■ Customize or extend human tasks ■ Customize or extend Oracle Mediator ■ Customize SOA composite application components such as a binding component and **wire** ■ Customize or extend transformations ■ Extend Web Services Description Language (WSDL) or Extensible Markup Language (XML) schema definition (XSD) files ■ Extend business rules ■ Extend Java EE connector architecture (JCA) adapters	JDeveloper (when logged in with the Customization Developer **role**)	Design time (when complete, you must deploy the SOA composite application)	System integrator

Note:

- You *cannot* customize human tasks, BPEL processes, and Oracle Mediators during runtime in a deployed SOA composite application.
- When using Oracle SOA Composer, you can save your customizations in a **sandbox** environment without applying them to a running instance. You can later return to the sandbox to make additional customizations. These customizations are applied to the running instance only when you click **Commit**.
- When you click **Save** or **Commit** in Oracle SOA Composer, a dialog is displayed in which you can optionally enter comments. When complete, you click **OK**, which performs the save or commit action, along with saving the comments.

5.1.1 Before You Begin Customizing SOA Composite Applications

Before you customize SOA components, become familiar with the Oracle Fusion application architecture that enables customizations, as described in Chapter 1, "Customizing and Extending Oracle Fusion Applications." Also understand the typical workflows for working with runtime customizations, as described in Chapter 2, "Understanding the Customization Development Lifecycle."

In addition, you need to perform the following tasks before you can begin customizing your application:

- Install JDeveloper and set up your development environment. Before you can implement customizations using JDeveloper, you must create an application workspace that imports the necessary parts of the application you want to customize. For more information, see Section 1.3.13, "Installing Customization Tools" and the "Setting Up Your Development Environment" chapter in the *Oracle Fusion Applications Developer's Guide*.
- Create a customization application workspace. For more information, see Chapter 3, "Using Oracle JDeveloper for Customizations."
- Start JDeveloper in the appropriate role.

  For more information, see Chapter 3, "Using Oracle JDeveloper for Customizations."

5.2 Customizing SOA Composite Applications

As described in Table 5–1, you can customize SOA components during runtime in a deployed SOA composite application with a runtime tool. This section provides an overview of these tasks and provides references to additional documentation for more specific instructions.

> **Note:** You cannot customize SOA components in Oracle Fusion CRM Application Composer (Application Composer). However, you can extend a **business event** in Application Composer and use the **Event notification** action to notify a SOA composite application by email of the occurrence of the event.

Task: Start the Runtime Customization Tool

Use a web browser to start the tools shown in Table 5–2 for customizing approval configuration and assignment rules, nonapproval business rules, domain value maps, and SOA composite application endpoint properties at runtime.

Table 5–2 Starting the Customization Tool

For Customizing...	Start...	By Entering...
Business rules		
▪ Approval configuration and assignment rules	▪ Oracle BPM Worklist (recommended)	`http://host:port/integration/worklistapp`
	▪ Oracle SOA Composer	`http://host:port/soa/composer`
	Note: If you use Oracle SOA Composer to customize approval configuration and assignment rules during runtime, changes in subsequent revision patches are *not* applied successfully.	
▪ Nonapproval business rules	Oracle SOA Composer	`http://host:port/soa/composer`
Domain value maps	Oracle SOA Composer	`http://host:port/soa/composer`

Table 5–2 (Cont.) Starting the Customization Tool

For Customizing...	Start...	By Entering...
SOA composite application endpoint properties such as Oracle WSM security policies and binding component properties	Fusion Applications Control	`http://host:port/em`

Task: Select the Data to Customize

After accessing the runtime customization tool to use, select the data to customize:

- Oracle SOA Composer:
  1. From the **Open** list in Oracle SOA Composer, select the data to customize, as shown in Figure 5–1.

Figure 5–1 Open Menu of Oracle SOA Composer

Table 5–3 describes the options available for selection.

Table 5–3 Selecting the Data to Customize

For Customizing...	Select...
Nonapproval business rules	**Open Rules**
Domain value maps	**Open DVM**
Approval configuration and assignment rules	**Open Task** **Note**: If you use Oracle SOA Composer to customize approval configuration and assignment rules during runtime, changes in subsequent revision patches are *not* applied successfully.

- Oracle BPM Worklist:
  1. In the **Administration** section, click the **Task Configuration** tab.
  2. Select a specific approval configuration and assignment rule task to customize.

     The **Event Driven** and **Data Driven** tabs are now accessible.
  3. Select a task to view or customize from the list of task types.
- Fusion Applications Control:
  1. In the navigation pane in Fusion Applications Control, expand the **SOA** folder.
  2. Expand **soa-infra**.

3. Expand the partition in which the SOA composite applications are deployed (for example, **default**).
4. Select the SOA composite application to customize.

Task: Customize Business Rules

Two categories of rules are available:

- Approval configuration and assignment rules:

  You can customize approval configuration and assignment rules included in a deployed SOA composite application using Oracle BPM Worklist (recommended), as shown in Figure 5–2, or in Oracle SOA Composer, as shown in Figure 5–3.

  For more information, see the following:

  – The "Using Approval Management" chapter in the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management* (for Oracle BPM Worklist)

  – The "Working with Tasks at Run Time" section in the *Oracle Fusion Middleware User's Guide for Oracle Business Rules* (for Oracle SOA Composer)

Figure 5–2 Approval Configuration and Assignment Rule Customizations in Oracle BPM Worklist

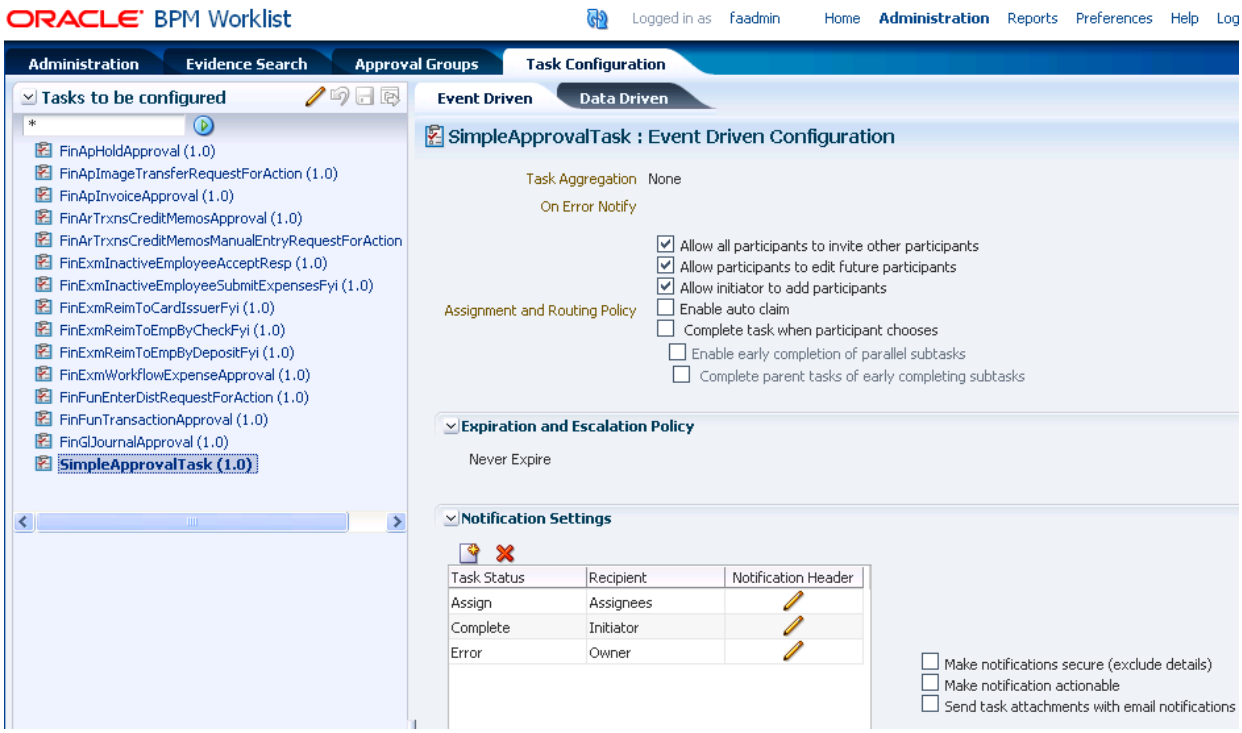
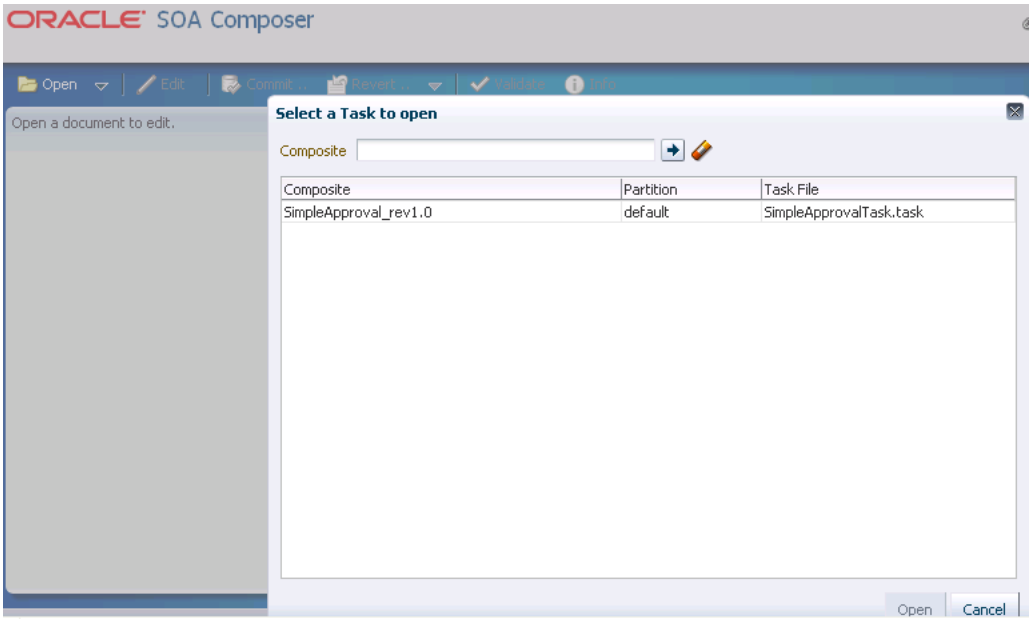


Figure 5–3 Approval Configuration and Assignment Rule Customizations in Oracle SOA Composer



How to customize the text in notifications in Oracle BPM Worklist is decided by what you want to customize in the task detail page (the page rendered when you click the task in Oracle BPM Worklist):

- Some strings are part of Oracle SOA Suite, other strings are part of the Oracle Fusion Applications-owned Oracle Application Development Framework (Oracle ADF) **resource bundle**, and other strings are part of the Oracle Fusion Applications-owned SOA resource bundle.
- The task title, task outcome, approval reason, stage name, and participant type strings are stored in the Oracle Fusion Applications-owned SOA resource bundles. You *cannot* customize these because there is no support for that functionality in Oracle SOA Suite.
- The **business object**-specific text and sections are implemented in Oracle ADF and resource bundles are owned by Oracle Fusion Applications. These strings can be customized only in JDeveloper.
- The Oracle SOA Suite-owned strings correspond to those in the **Comments**, **Attachment**, and **History** sections in Oracle BPM Worklist. The actions along the top of the page (excluding the custom actions defined in the `.task` file) are also part of Oracle SOA Suite. These strings in the Oracle SOA Suite-owned resource bundles can be customized by following the instructions in the "Resource Bundles in Workflow Services" section in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

How text appears in email notifications for human tasks is also decided by what you want to customize:

- The subject (derived from the task title) and custom outcomes are defined in the Oracle Fusion Applications-owned SOA resource bundle. You cannot customize these because there is no support for that functionality in Oracle SOA Suite.
- You can customize the notification message (the first line of instructions in the email) during runtime in Oracle BPM Worklist.

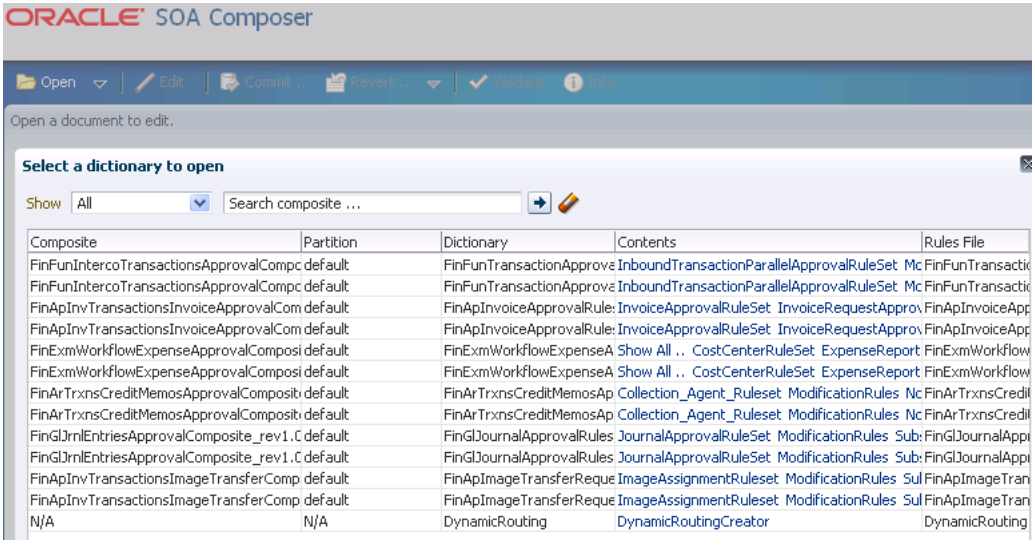
  - The remaining email content is the same as customizing the text in notifications in Oracle BPM Worklist.

- Nonapproval business rules:

  You can view, customize, and commit changes to a rule dictionary included in a deployed SOA composite application using Oracle SOA Composer, as shown in Figure 5–4. Supported customizations consist of the following:

  - Customizing dictionary bucketsets
  - Customizing rules in a ruleset
  - Customizing advanced rule settings
  - Customizing conditions and actions
  - Customizing advanced mode rules
  - Customizing a decision table
  - Validating rule dictionaries

Figure 5–4 Nonapproval Business Rule Customizations in Oracle SOA Composer

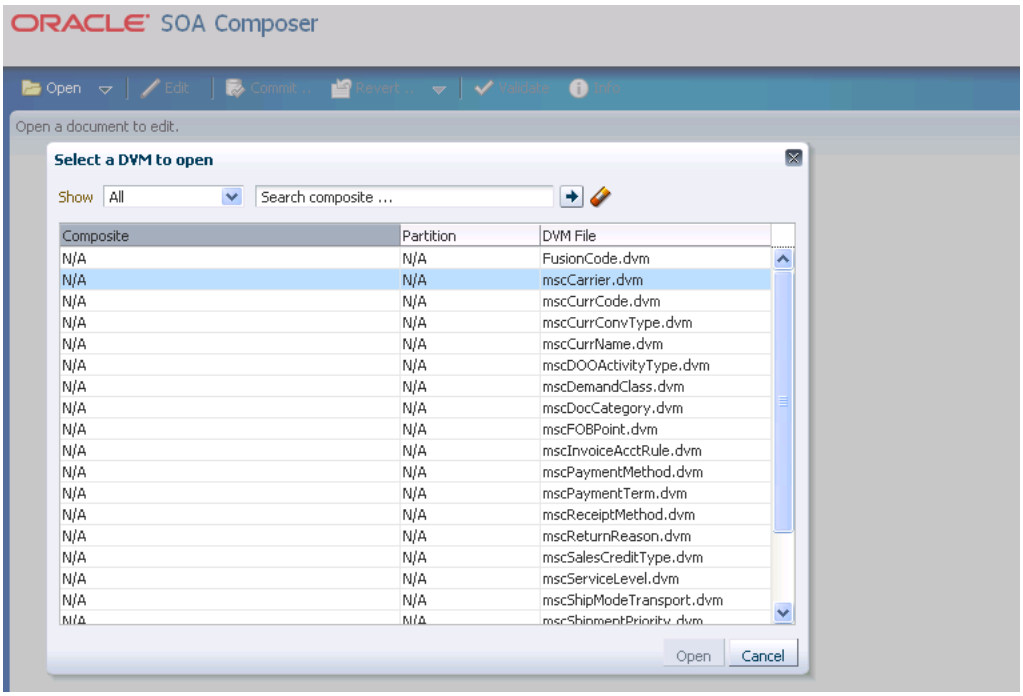


For more information about customizing business rules in Oracle SOA Composer, see the "Using Oracle SOA Composer with Oracle Business Rules" chapter in the *Oracle Fusion Middleware User's Guide for Oracle Business Rules*.

Task: Customize Domain Value Maps

You can customize domain value map rows included in a deployed SOA composite application using Oracle SOA Composer, as shown in Figure 5–5. For more information, see the "Using Oracle SOA Composer with Domain Value Maps" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Figure 5–5 Domain Value Map Customizations in Oracle SOA Composer



Task: Customize SOA Composite Application Endpoint Properties

You can customize endpoint address properties for an external reference such as Oracle WSM security policies and binding components included in a deployed SOA composite application using Fusion Applications Control.

Figure 5–6 provides details about customizing Oracle WSM security policies. For more information, see the "Managing SOA Composite Application Policies" section in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Figure 5–6 Security Policy Customizations in Fusion Applications Control

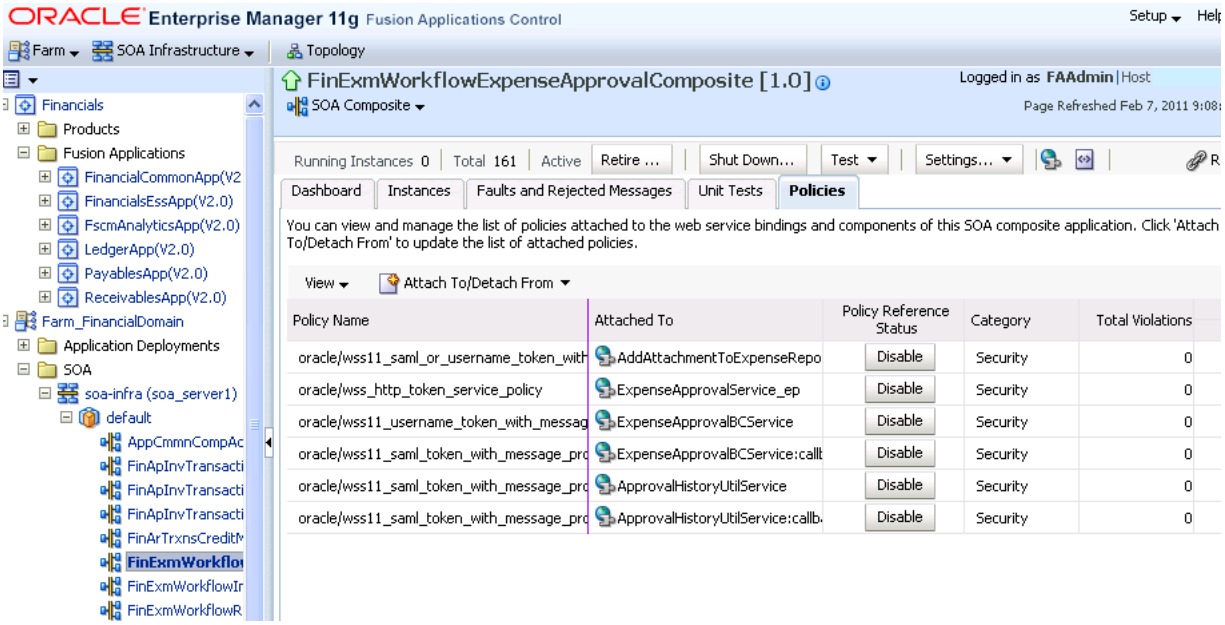
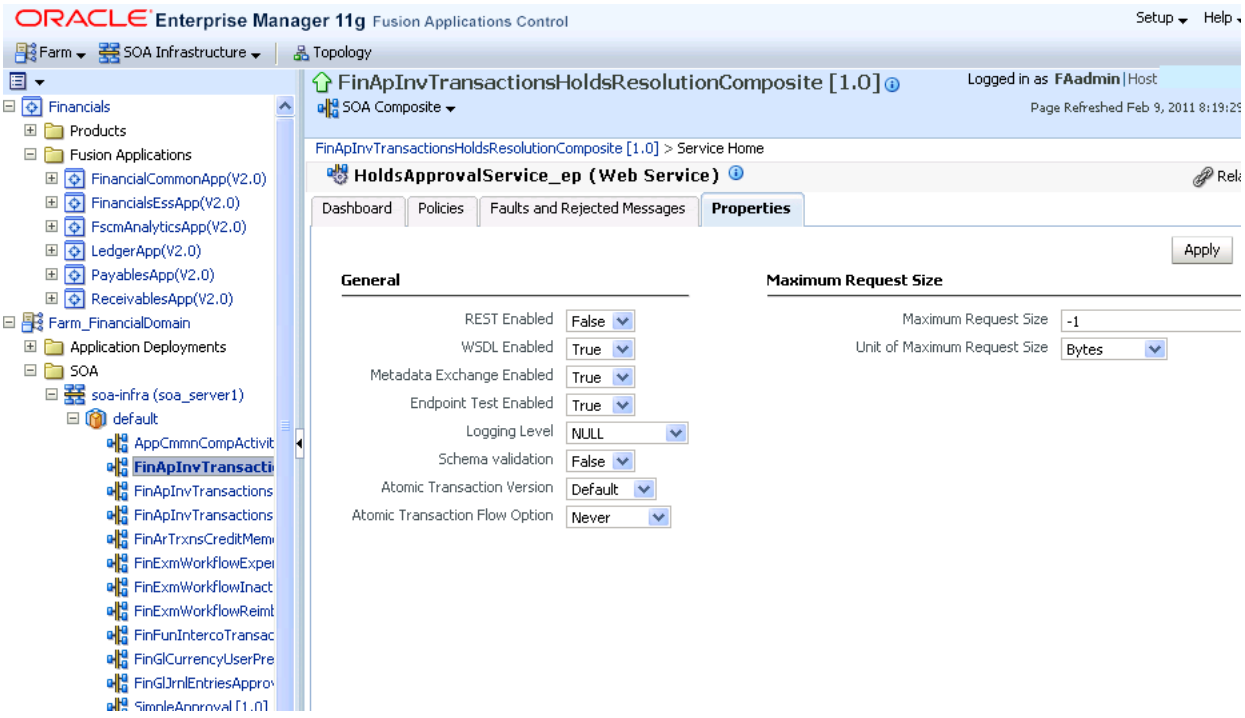


Figure 5–7 provides details about customizing binding component properties for services and references. For more information, see the "Configuring Service and Reference Binding Component Properties" chapter in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Figure 5–7 Binding Component Property Customizations in Fusion Applications Control



Task: Synchronizing Customized Flexfields in the MDS Repository for SOA

SOA composite applications in Oracle Fusion Applications reference copies of the original XSD schema files included in the MDS repository for SOA. When you

customize and deploy Oracle Fusion Applications flexfields (or upgrade the base table, after which the flexfields are automatically reapplied), which result in a new XSD file being generated in the MDS repository for Oracle Fusion Applications, the updated XSD files must be synchronized in the MDS repository for SOA for use in the fact models in business rules.

To perform this synchronization, a special SOA composite application named UpdateSOAMDS is included with Oracle Fusion Applications. By default, UpdateSOAMDS is automatically deployed. When a synchronization is required, you manually invoke an instance of this SOA composite application to synchronize the updated XSD files in the MDS repository for SOA. You can view the results of this synchronization in the audit trail in Fusion Applications Control.

1. Invoke UpdateSOAMDS.
   a. Log in to Fusion Applications Control.
   b. In the navigation pane, expand **soa-infra** and the domain.
   c. Select **UpdateSOAMDS**.
   d. At the top of the Dashboard page for **UpdateSOAMDS**, click **Test**.
   e. In the **Operation** list, note that the **updateDuring** operation is selected, as shown in Figure 5–8.

Figure 5–8 Operations to Perform

The **updateDuring** operation specifies how far back in time to go to get flexfield updates for synchronizing in the MDS repository for SOA.

   f. In the **Value** field of the **Input Arguments** section, enter a value, as shown in Figure 5–9.

Figure 5–9 Value Field

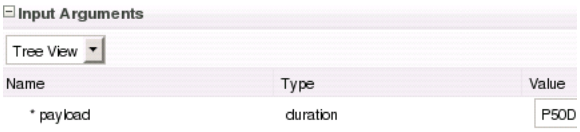


Table 5–4 provides examples of how to specify a value. The **updateDuring** operation uses the `xsd:duration` type as input to obtain the data.

Table 5–4 Operation Value Examples

If You Enter...	Description
`P50D`	The operation goes back 50 days to get flexfield updates that occurred.

Table 5–4 (Cont.) Operation Value Examples

If You Enter...	Description
`P1M2DT3H`	The operation goes back one month, two days, and three hours to get flexfield updates that occurred.

g. Click **Test Web Service**.

All rule dictionaries in the MDS repository for SOA that use the affected XSD schemas are altered. The data model of the rule dictionaries is modified and the fact types are reimported. After reimporting the XSD schemas, the rule dictionaries are saved in the MDS repository for SOA.

The Java Architecture for XML Binding (JAXB) 2.0 classes for the fact type model of the rule dictionaries that have been altered are regenerated and compiled into the appropriate SOA composite application `SCA-INF/gen-classes` directories.

Other SOA instances in the cluster are notified of the flexfield customizations.

The class loader for the SOA composite applications in which the rule dictionaries were altered is invalidated and a new class loader is extended with the next request for the SOA composite application.

The SOA instances not involved in updating the rule dictionaries in the MDS repository for SOA regenerate the JAXB 2.0 classes for the SOA composite applications that comprise rule dictionaries in which the fact type model was altered.

2. View the results in the audit trail:

a. In the navigator, click **soa-infra**.

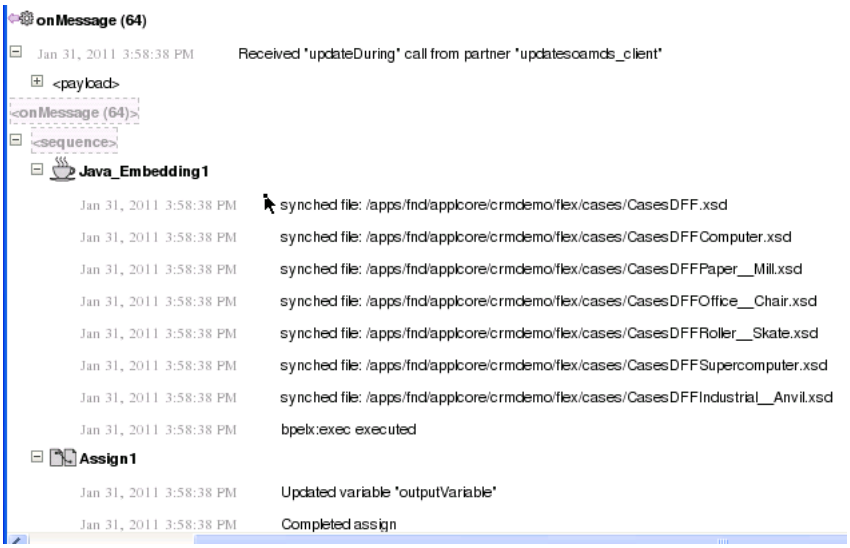
b. In the **Recent Instances** section of the Dashboard page of the SOA Infrastructure, click the instance ID.

c. In the **Trace** section of the Flow Trace page, click the **UpdateSOAMDS** BPEL service component.

d. Click **View XML Document** to expand the activities in the audit trail, as needed.

e. View the list of XSD schema files synchronized in the MDS repository for SOA in the audit trail, as shown in Figure 5–10.

Figure 5–10 Audit Trail Results



Note: Before Release 11*g* R1 (11.1.1.4), Oracle BPM Worklist included a feature known as flex fields. Starting with Release 11*g* R1 (11.1.1.4), flex fields are now known as mapped attributes. Do *not* confuse Oracle BPM Worklist flex fields with Oracle Fusion Applications flexfields; they are completely different features.

5.3 Merging Runtime Customizations from a Previously Deployed Revision into a New Revision

After using a SOA composite application customized at runtime for a while, a new patch revision of the SOA composite application may become available. Repeating the process of customizing the new revision of the SOA composite application at runtime can be cumbersome and time-consuming. To avoid these challenges, use OPatch. OPatch is an Oracle-supplied, Java-based utility that enables you to merge customizations made during runtime in a previously deployed SOA composite application into a new patch revision of the SOA composite application. OPatch preserves your runtime customizations and prevents you from having to reenter the customizations again for the next patch revision.

OPatch merges a new patch revision into a SOA composite application that was previously customized during design time in JDeveloper and runtime in Oracle SOA Composer, Oracle BPM Worklist, or Fusion Applications Control. For specific procedures on patching SOA composite applications with OPatch, see the "Patching Service-Oriented Architecture (SOA) Composites" section in the *Oracle Fusion Applications Patching Guide*.

Task: Merge Runtime Customizations from a Previously Deployed Revision into a New Revision

Before using OPatch to merge runtime customizations from a previously deployed revision into a new revision, review the recommendations in Table 5–5 to ensure that you merge customizations successfully.

Table 5–5 Recommendations on Merging Patch Revision Customizations and Extensions

Component	Recommendation
Deletion of base components	Delete only components that you added as part of the customization, and not components that are part of the base revision. This is because the deletion of base components does not survive the move to the new revision, but the deletion of the wiring does. If you delete an existing base component, it comes back again when you get the new revision, which still has the component. However, the wire deletion survives the upgrade because the `composite.xml` file is customizable.

Table 5–5 (Cont.) Recommendations on Merging Patch Revision Customizations and Extensions

Component	Recommendation
Business rules	If business rules are customized at runtime, and those runtime customizations must be preserved in subsequent revisions of the base version of the SOA composite application, it is recommended that the rules dictionaries be split into two dictionaries and linked using the dictionary linking functionality. The base rule, linked dictionary contains the data model of the dictionary and the custom rules dictionary contains all the rules customized at runtime. The OPatch process preserves the customized dictionary when it merges the customized application with subsequent versions of the application. Business rules are used in different scenarios and the following information identifies how to handle each situation: ■ Approval configuration and assignment rules These rules are used within human tasks to identify approvers and the routing of human tasks. Approval rules are always generated as base and custom dictionaries at design time. No further configuration is necessary at design time. Runtime customizations: If you must customize approval configuration and assignment rules during runtime, use only Oracle BPM Worklist to perform this task. Using Oracle BPM Worklist enables: -) Approval assignment and configuration rules to automatically be stored in a custom rules dictionary (`Rule.rules`). The custom rules dictionary is initially shipped with only sample, pre-seeded rules. The custom rules dictionary is separate from the base rule, linked dictionary (`RuleBase.rules`). The base rule, linked dictionary contains Oracle Fusion Applications fact definitions. Revision patches are applied to the base rule, linked dictionary. -) Changes in subsequent revision patches to be applied successfully to the base rule, linked dictionary. If you use Oracle SOA Composer to customize approval configuration and assignment rules during runtime, changes in subsequent revision patches are *not* applied successfully. Design time customizations: You cannot customize existing rules that are part of the base version of the SOA composite application at design time in JDeveloper. However, you can extend new rules that you later customize. ■ Nonapproval business rules These rules are used directly in processes like BPEL and BPMN outside of the context of a human task. These dictionaries are not generated as linked dictionaries in JDeveloper and must be manually split as linked dictionaries. Runtime customizations: If the dictionaries are split as linked dictionaries, ensure that only the linked dictionaries are customized from Oracle SOA Composer. Identification of the base rule and linked rule dictionary is up to you to develop. Design time customizations: You cannot customize existing rules that are part of the base version of the SOA composite application at design time in JDeveloper. However, you can extend new rules that you later customize.

Table 5–5 (Cont.) Recommendations on Merging Patch Revision Customizations and Extensions

Component	Recommendation
Default uniform resource locators (URLs) for service binding components	Use default URLs for service binding components. If the revision is used in the URL for service binding components, when the SOA composite application is patched using OPatch, the revision of the SOA composite application is customized. In this case, the reference to URLs for service binding components fails to work. In this scenario, you must manually update all the URL references for service binding components.
Oracle BPEL Process Manager scope activity	If a base SOA composite application team removes the scope activity in the next revision of the SOA composite application, when a vertical SOA composite application team or customer runs OPatch to apply the new revision of the SOA composite application to their customized version, all customizations they performed on that scope activity in the first revision are lost.
Renaming of a SOA composite application whose SOA archive (SAR) file is imported in JDeveloper	When importing a **SAR file** for customization in JDeveloper, the SOA composite application must *not* be renamed. In addition, if you rename a SOA composite application, OPatch cannot detect runtime customizations made in Oracle SOA Composer, Oracle BPM Worklist, and Fusion Applications Control. You must manually re-apply those customizations.
Base revision of a SOA composite application with JDeveloper customizations	Assume you customize the base revision of a SOA composite application with the Customization Developer role in JDeveloper, and then deploy the SOA composite application. When the base revision is updated and a newer revision is made available, the customer uses OPatch to apply the patch revision. OPatch may then fail because there are JDeveloper customizations in the deployed SOA composite application. To resolve this issue, perform the following steps: 1. Open the customized SOA composite application with the **Default Role** in JDeveloper. 2. Import the patched base version 2 SAR file into this SOA composite application project extended in Section 3.3, "Customizing SOA Composite Applications with JDeveloper." 3. Restart JDeveloper with the **Customization Developer** role. 4. Open the preceding customized SOA composite application. Error messages are shown in case of conflicts. 5. Resolve the conflicts in the SOA composite application. 6. Deploy the SOA composite application to the SAR file. The new SAR file should be replaced by the patched base version 2 SAR file. 7. Proceed with the OPatch process. **Note:** Ensure that the backup of the SAR files is taken properly.

5.4 Extending or Customizing Custom SOA Composite Applications

You can customize or extend some SOA components during design time in JDeveloper when logged in with the Customization Developer role. Components that are extended in JDeveloper can be further customized in JDeveloper when again logged in with the Customization Developer role. Customization changes are maintained in separate `.xml` files from the base component files. Note that you cannot make customizations in Source view in JDeveloper; only customizations made in Design view are supported.

Note:

- A new SOA **artifact (SAR file)** extended in the SOA composite application survives patching.
- Ensure that you provide unique names for any new components and SOA artifacts that you extend. For example, add a prefix to each component and SOA artifact name that is a unique identifier.

Table 5–6 describes which existing base SOA artifacts in a SOA composite application can be customized and which new SOA artifacts can be extended when logged in to JDeveloper with the Customization Developer role.

Table 5–6 Customizable and Extendable SOA Artifacts in Customization Developer Role

SOA Artifacts	Existing Artifact in Base SOA Composite Application Is Customizable with Customization Developer Role?	SOA Artifact Is Extendable with Customization Developer Role?
SOA composite application components	Yes	Yes
BPEL process	Yes	Yes
Oracle Mediator	Yes	Yes
Human task	No	Yes
Business rule	No	Yes
Extensible style sheet language transformations (XSLT) map	No	Yes
Cross references (XREFs)	No	No
Domain value maps	No	No
XSD	No	Yes
WSDL	No	Yes
Business events	No	Yes
JCA Adapters	No	Yes

Table 5–7 provides more specific details about which SOA artifacts can be extended when logged in to JDeveloper with the Customization Developer role.

Table 5–7 SOA Artifact Extensibility in JDeveloper with Customization Developer Role

SOA Artifact	Extendable?	Description
SOA composite application	No	Only one SOA composite application per Oracle SOA Suite project is permitted.
BPEL process	Yes	Can drag a BPEL process from the Component Palette into SOA Composite Editor or Oracle BPEL Designer.
Oracle Mediator	Yes	Can drag an Oracle Mediator from the Component Palette into SOA Composite Editor or Oracle BPEL Designer.

Table 5–7 (Cont.) SOA Artifact Extensibility in JDeveloper with Customization Developer

SOA Artifact	Extendable?	Description
Human task	Yes	Can drag a human task from the Component Palette into SOA Composite Editor or Oracle BPEL Designer.
Business rule	Yes	Can drag a business rule from the Component Palette into SOA Composite Editor or Oracle BPEL Designer.
XSLT map	Yes	Can extend a transformation in a transform activity in Oracle BPEL Designer or Mediator Editor.
Domain value maps	No	The New Gallery dialog is disabled with the Customization Developer role.
XREFs	No	The New Gallery dialog is disabled with the Customization Developer role.
XSD	Yes	Right-click an Oracle SOA Suite project and select **SOA**, or as the result of extending other SOA artifacts.
WSDL	Yes	Right-click an Oracle SOA Suite project and select **SOA**, or as the result of extending other SOA artifacts.
Business events	Yes	Subscribe to or publish events for a BPEL process or Oracle Mediator component in SOA Composite Editor, Oracle BPEL Designer, or Mediator Editor.
JCA adapters	Yes	Drag adapters from the Component Palette into SOA Composite Editor or Oracle BPEL Designer.

Task: Customize a Base SOA Composite Application in JDeveloper

You can customize a base SOA composite application of Oracle Fusion Applications in JDeveloper. These steps provide an overview of SOA composite application customization and assume that you know the following:

- How to set up the customization layer through the `adf-config.xml` editor
- The customization classes defined by Oracle Fusion Applications

For more information, see the "Customizing SOA Composite Applications" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

> **Note:** See Section 3.3.2, "Setting Up the JDeveloper Application Workspace and SOA Composite Application Project for MDS Repository Customization" for instructions on setting up the JDeveloper workspace (JWS) and SOA composite application project when customizing Oracle Fusion Applications SOA composite applications.

1. Install Oracle Fusion Applications with a SOA composite application that you want to customize in JDeveloper.
2. In Fusion Applications Control, go to the home page of the SOA composite application to export.
3. From the **SOA Composite** menu at the top of the page, select **Export**.
4. Obtain the base SAR file for initially customizing from other locations, including:
   - Checking out the base SOA composite application project from the source control system where the base SOA composite application project was checked

in by the base development team. This way, no SAR file deployment, export command, or import command is involved.

- Importing the base SOA composite application SAR file that was deployed from the base SOA composite application project.
- Importing the base SOA composite application SAR file that was exported (without runtime changes) from the Export Composite page of the Fusion Applications Control installation from which the SOA server is managed.

5. Extend layer values for customization to the `CustomizationLayerValues.xml` file (can perform this task in JDeveloper or from the directory structure).
6. Start JDeveloper in the **Default Role**.
7. Extend a new SOA composite application.
8. From the **File** main menu, choose **Import**, then **SOA Archive Into SOA Project** to import the exported SAR file into the new SOA composite application in JDeveloper.
9. In the Import Composite Archive wizard, select the **Import For Customization** checkbox.
10. From the **Tools** main menu, choose **Preferences**, then **Roles**, and then **Customization Developer**.
11. Restart JDeveloper, and customize the layers of the SOA composite application.
12. Right-click the project and choose **Deploy** to extend a customized SAR file of the SOA composite application in Oracle Fusion Applications.

Note: After performing the initial customizations described in these procedures, you can no longer export the SOA composite application from the runtime. This is because the SOA composite application is a merged SOA composite application, and no longer the original base SOA composite application.

For more information about exporting SAR files, see the "Exporting a Deployed SOA Composite Application" section in the *Oracle Fusion Middleware Administrator's Guide for Oracle SOA Suite and Oracle Business Process Management Suite*.

Task: Extend or Customize Custom Business Rules

You can extend business rules in a SOA composite application during design time in JDeveloper when logged in with the Customization Developer role. After extending these business rules, you can further customize them in JDeveloper when again logged in with the Customization Developer role. You cannot customize existing business rules that are part of the base version of the SOA composite application.

For information about customizing business rules during runtime, see Section 5.2, "Customizing SOA Composite Applications."

Task: Extend or Customize Custom BPEL Processes

You can extend or customize BPEL processes in a SOA composite application during design time in JDeveloper when logged in with the Customization Developer role. For example, you can perform the following tasks:

- Extend or delete a new scope or other activity
- Customize an activity

- Extend a **partner link**
- Extend a transformation

For more information about extending or customizing BPEL processes, see the "Using the BPEL Process Service Component" part in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Task: Extend or Customize Custom Human Tasks

You can extend human tasks in a SOA composite application during design time in JDeveloper when logged in with the Customization Developer role. After extending these human tasks, you can further customize them in JDeveloper when again logged in with the Customization Developer role. You cannot customize existing human tasks that are part of the base version of the SOA composite application.

For more information about extending human tasks, see the "Using the Human Workflow Service Component" part in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Task: Extend or Customize Custom Oracle Mediators

You can extend or customize an Oracle Mediator in a SOA composite application during design time in JDeveloper when logged in with the Customization Developer role. For example, you can perform the following tasks:

- Extend a routing rule
- Customize an XPath condition
- Make any other changes, except those that affect files such as extensible style sheet languages (XSLs) (for transformations), WSDLs, event definition languages (EDLs) (for business events), or XSDs. Note that *new* SOA artifacts can be extended or customized.

For more information, see the "Using the Oracle Mediator Service Component" part in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Task: Customize SOA Composite Application Components

You can customize SOA composite application endpoint properties in a SOA composite application during design time in JDeveloper when logged in with the Customization Developer role. For example, you can perform the following tasks:

- Extend and delete a reference binding component
- Extend and delete a service binding component (entry point)
- Extend, customize, and delete a wire between components

For more information, see the "Developing SOA Composite Applications with Oracle SOA Suite" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Task: Extend or Customize Transformations (in a Transform Activity)

You cannot customize existing transformations that are part of the base SOA composite application in JDeveloper. However, you can extend a new transform activity in a BPEL process or in the Transformation Map dialog of Oracle Mediator during design time in JDeveloper when logged in with the Customization Developer role. After extending the transformation, you can further customize it in JDeveloper when again logged in with the Customization Developer role. For example, you can perform the following tasks:

- Specify the mapper file (`.xsl`) to which the transform activity points from the **Mapper File** field of a transform activity in a BPEL process or the Transformation Map dialog of Oracle Mediator. However, you cannot extend or customize mappings. The mappings are defined in the XSL file (not in the transform activity), which is not customizable.
- Copy an out-of-the-box XSL file into a custom XSL artifact, add the custom logic to the custom XSL, and customize the transform activity to reference the custom XSL. Additionally, you must copy the contents of the XSL file in the base SOA composite application into the custom XSL file.

Task: Extend XSD or WSDL Files

You can extend an XSD schema or WSDL document in JDeveloper when logged in with the Customization Developer role.

1. Right-click the Oracle SOA Suite project in the Application Navigator.
2. Select **SOA**.
3. Select the SOA artifact to extend:
   - **Create XML Schema**

     Invokes the Create XML Schema dialog for extending a new XML schema file in the project. When complete, the new schema file automatically opens.
   - **Create WSDL Document**

     Invokes the Create WSDL dialog to extend a new WSDL file in the project.

Task: Extend Business Events

You cannot directly extend business events in JDeveloper when logged in with the Customization Developer role. The New Gallery dialog that is displayed when you select **New** from the **File** main menu is disabled with the Customization Developer role. However, you can create business events as part of other Oracle SOA Suite customizations such as when allowing Oracle Mediator to subscribe to an event.

For more information, see the "Using Business Events and the Event Delivery Network" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Task: Extend JCA Adapters

You can extend JCA adapters in JDeveloper when logged in with the Customization Developer role.

For more information, see the *Oracle Fusion Middleware User's Guide for Technology Adapters*.

5.5 Deploying SOA Composite Application Customizations and Extensions

You must redeploy a customized or extended SOA composite application after making changes in JDeveloper. The development and deployment phase is as follows:

- During base SOA composite application development, you create a customizable SOA project from the Default role in JDeveloper, set up customization layers, and deploy the SOA composite application to a base SAR file.

- During customization, you import (for customization) the base SOA composite application SAR file to extend a new SOA project, change from the Default role to the Customization Developer role, perform customizations, and deploy the SOA composite application to create a customized SAR file.

For more information, see the "Customizing SOA Composite Applications" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

5.6 Extending a New Oracle SOA Suite Service

You can extend new SOA composite application services to integrate with Oracle Fusion Applications. This section provides an overview of tasks for extending and consuming new services and provides references to documentation that more specifically describes these tasks.

Task: Setting Up a Development Environment

You must set up and configure a development environment in JDeveloper to create new Oracle SOA Suite services. For more information, see the "Getting Started Building Your Oracle Fusion Applications" part in the *Oracle Fusion Applications Developer's Guide*.

Task: Using JDeveloper to Create Applications, Projects, and Services

Whenever you create new projects, you must first create an application using templates provided by JDeveloper. For more information, see the "Setting Up Your JDeveloper Application Workspace and Projects" chapter in the *Oracle Fusion Applications Developer's Guide*.

You can select an Oracle SOA Suite project template when creating a JDeveloper application. For more information about creating Oracle SOA Suite projects, see the "Developing SOA Composite Applications with Oracle SOA Suite" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

You can extend an ADF Business Components service to be consumed by the SOA composite application. The ADF Business Components service is used for connecting Oracle ADF applications using service data object (SDO) data formats with the SOA composite application. For more information, see the "Getting Started with Binding Components" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Task: Understanding Common Service Use Cases and Design Patterns

There are fundamental patterns for Oracle Fusion Applications developers to follow when building applications involving Oracle ADF and Oracle SOA Suite. These patterns fall into three main categories:

- Using business events to initiate business processes
- Orchestrating over business logic implemented with Oracle ADF, Java, procedural language/structured query language (PL/SQL), and SOA composite applications
- Modeling human task flows in Oracle ADF applications

For more information about these and other design categories, see the "Common Service Use Cases and Design Patterns" part in the *Oracle Fusion Applications Developer's Guide*.

Task: Using Oracle SOA Suite with MDS Repository

MDS Repository contains metadata for certain types of deployed applications, such as SOA composite applications. You can store Oracle Fusion Applications artifacts and custom artifacts in MDS Repository. You connect to the repository to consume these artifacts.

For more information about MDS Repository, see the "Managing the Metadata Repository" chapter in the *Oracle Fusion Middleware Administrator's Guide*.

For more information about creating a connection from Oracle SOA Suite to MDS Repository, using the MDS repository for SOA to store custom SOA artifacts, and connecting to and consuming SOA artifacts from the MDS repository for SOA, see the "Creating a SOA-MDS Connection" section in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

Task: Discovering Oracle Fusion Applications Services

Oracle Fusion Applications includes web services that are available for public consumption. These web services are defined in Oracle Enterprise Repository and available for discovery. When extending Oracle Fusion Applications and building SOA composite applications to invoke services built by Oracle Fusion Applications, you can use Oracle Enterprise Repository to perform the following tasks:

- Use Oracle Enterprise Repository to discover the service.
- Follow the link provided by Oracle Enterprise Repository to access the WSDL file.
- When building the client, have JDeveloper download the WSDL file locally so that the client is not accessing the runtime WSDL file.

For more information about Oracle Enterprise Repository, see the *Oracle Fusion Middleware User Guide for Oracle Enterprise Repository*.

Task: Securing Oracle Fusion Applications and Services

You must secure Oracle Fusion Applications and services to be consumed by SOA composite applications.

For more information about Oracle Fusion Applications security, see the *Oracle Fusion Applications Security Guide*.

For more information about Oracle ADF Application Artifacts security, see Chapter 8, "Customizing Security for Oracle ADF Application Artifacts."

For more information about web services security, see the "Securing Web Services Use Cases" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Deploying SOA Composite Applications and Services

You must deploy SOA composite applications and the services to be consumed.

For more information about deploying SOA composite applications, see the "Deploying SOA Composite Applications" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle SOA Suite*.

For more information about deploying external references such as web services, see the "Deploying Web Services Applications" chapter in the *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*.

Task: Understanding Fusion Applications Deployment Topology

An enterprise deployment is an Oracle guidelines blueprint based on proven Oracle high-availability and security technologies and recommendations for Oracle Fusion

Applications. For more information about deployment in an enterprise environment, see the *Oracle Fusion Applications Enterprise Deployment Guide for Customer Relationship Management*.

6

Customizing and Extending Oracle BPM Project Templates

This chapter describes how to use Oracle JDeveloper and Oracle Business Process Management Studio (Oracle BPM Studio) to customize and extend Oracle BPM project templates. BPM projects contain the Business Process Modeling Notation (BPMN) processes used by Oracle Fusion applications. Several Oracle Fusion applications use BPMN processes to define process flows within the application.

This chapter includes the following sections:

6.1 About Customizing Project Templates

BPM project templates are templates used to create new BPM projects. Project templates are created by developers and contain all of the elements necessary to create a new BPM project that can be deployed to runtime. This includes all of the necessary BPMN processes and business catalog **component**.

Oracle Fusion applications provide default project templates containing the required BPMN processes and business catalog component. See the product-specific documentation in Oracle Fusion Applications Help for a list of the default BPM project templates provided by Oracle Fusion applications.

Developers can customize and extend these project templates. Project templates are customized or extended by developers using Oracle BPM Studio, which is an extension to JDeveloper that provides additional editors for creating and customizing BPMN processes and related artifacts.

For more information about Oracle BPM Studio, see the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*.

In the context of Oracle Fusion applications, developers can customize project templates when it is necessary to customize or extend business catalog component that are part of the default project templates. See the product-specific documentation in Oracle Fusion Applications Help for a list of the default BPM project templates provided by Oracle Fusion applications.

After customizing or extending a project template, it can be published to the Oracle BPM repository. Project templates are shared between Oracle BPM Studio and Oracle Business Process Composer using the Oracle BPM repository. Additionally, BPM

projects can be shared between Business Process Composer and JDeveloper users via the Oracle BPM repository.

> **Note:** When customizing a project template, you must first make a copy of the existing template using JDeveloper. This enables you to avoid overwriting project templates previously saved to the Oracle BPM repository.

After a template is published to the repository, it is available to Business Process Composer users. Business Process Composer users can create and edit BPM projects created using these templates and can collaborate on these projects with process developers using JDeveloper. They can also create a **SAR file** or deployment plan, or deploy BPM projects directly to full test environment without having to reedit and deploy a project using JDeveloper.

See the "Workflow: Creating Project Templates" section in the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management* for information about the typical workflow for sharing project templates between Oracle BPM Studio and Business Process Composer.

6.1.1 About the Business Catalog

The business catalog is a set of reusable components that contain all of the necessary technical implementation to create a BPMN process flow that can be deployed as part of a running Oracle Fusion application.

The business catalog contains the following components:

- **business rule**

  Define a business decision that enables dynamic decisions to be made at runtime. Use rules to automate policies, computations, and reasoning while separating rule logic from underlying application code.

- **human task**

  Create a workflow that describes the tasks for users or groups to perform as part of an end-to-end business process flow.

- service

  Define how a BPMN process connects to other business processes and systems, including databases and web services.

Table 6–1 lists the business catalog that can be customized using Oracle Business Process Composer.

Table 6–1 List of Business Catalog Components

Business Catalog Component	Can be created using Business Process Composer?	Can be customized using Business Process Composer?
Business rules	Yes	Yes
Human tasks	Yes You can create human tasks using Business Process Composer, however not all functionality of a human task can be customized.	Yes

Table 6–1 (Cont.) List of Business Catalog Components

Business Catalog Component	Can be created using Business Process Composer?	Can be customized using Business Process Composer?
Services	Yes	Yes

6.1.2 Before You Begin Using JDeveloper to Customize Project Templates

Before you customize the artifacts within a project template, including business catalog components, business processes, and SOA components using JDeveloper, you should be familiar with the Oracle Fusion applications architecture that enables customization, as described in Chapter 1, "Customizing and Extending Oracle Fusion Applications."

You should also understand the typical workflows for working with customizations, as described in Chapter 2, "Understanding the Customization Development Lifecycle."

You will also need to install Oracle JDeveloper and set up your development environment. For more information, see Section 1.3.13, "Installing Customization Tools."

6.2 Customizing or Extending a Project Template

The following outlines the general tasks you must perform to customize or extend a BPM project template.

Task: Open a Project Template

You can open a project template with Oracle BPM Studio.

For information on opening a project template, see the "Working with Project Templates" section in the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management.*

The specific project or project template you must open depends on which Oracle Fusion application you are customizing. See the product-specific documentation in Oracle Fusion Applications Help for a list of the default BPM project templates provided by Oracle Fusion Applications.

Task: Create or Customize BPMN Processes

BPMN processes are accessible using the BPM Project Navigator. For information on using the BPM Project Navigator, see the "Oracle BPM Project Navigator" section in the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*.

For information about opening a BPMN process, see the "How to Open a Business Process" section in the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*.

For more information about working with flow objects in your process, see the "Working with Flow Objects in Your Process" section in the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*.

For general information about BPMN flow objects, see the "BPMN Flow Object Reference" appendix in the *Oracle Fusion Middleware Business Process Composer User's Guide for Oracle Business Process Management.*

Task: Create or Modify Business Catalog Components

Using Oracle BPM Studio, you can create or modify the following business catalog components within a project template:

- Services
- Human tasks
- Business rules

Task: Customize SOA Components

BPM projects are based on technology provided by Oracle SOA Suite. This includes reusable components and services that are included as part of a project template.

In addition to customizing business catalog components, you can customize applications by customizing SOA components, including the following:

- **domain value map**
- **BPEL process**
- Oracle Mediator

See Section 5.4, "Extending or Customizing Custom SOA Composite Applications" for more information.

6.3 Publishing Project Templates to the Oracle BPM Repository

In Oracle BPM, publishing a project template refers to the process of saving it in the Oracle BPM repository. You can publish project templates to the repository to make them available to Business Process Composer users.

The repository can also be used to share BPM projects between Business Process Composer and JDeveloper users as part of the process development lifecycle.

Publishing a project template to the Oracle BPM repository makes them available to Business Process Composer users, enabling collaboration between application developers and business users.

Task: Configure an Oracle BPM MDS Connection

Before publishing a project template to the Oracle BPM MDS repository, you must configure an MDS connection.

For more information about creating a connection to the repository, see the "How to Configure a Connection to the Oracle BPM Metadata Service Repository" section in the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*.

Task: Publish a Project Template

For information about publishing a project template, see the "How to Publish a Project or Project Template to Oracle BPM MDS" section of the *Oracle Fusion Middleware Modeling and Implementation Guide for Oracle Business Process Management*.

After publishing a project template, it is available to Business Process Composer users who can use it to create new BPMN process flows. See the "Customizing and Extending BPMN Processes" chapter in the *Oracle Fusion Applications Extensibility Guide for Business Analysts* for more information.

7

Customizing and Extending Oracle Enterprise Scheduler Jobs

This chapter describes how to use Oracle JDeveloper or Oracle Enterprise Manager Fusion Applications Control to create and **extend** scheduled jobs using Oracle Enterprise Scheduler.

This chapter includes the following sections:

7.1 About Customizing and Extending Oracle Enterprise Scheduler Jobs

Enterprise applications require the ability to respond to many real-time transactions requested by end users or web services. However, they also require the ability to offload larger transactions to run at a future time, or automate the running of application maintenance work based on a defined schedule.

Oracle Enterprise Scheduler provides the ability to run different job types, including: Java, PL/SQL, and spawned processes, distributed across nodes in a server cluster. Oracle Enterprise Scheduler runs these jobs securely, and provides monitoring and management through Fusion Applications Control.

Oracle Enterprise Scheduler provides scheduling services for the following purposes:

- Distributing job request processing across a cluster of servers
- Running Java, PL/SQL, and binary jobs
- Scheduling job requests based on recurrence
- Managing job requests with Fusion Applications Control

Oracle Enterprise Scheduler provides the critical requirements in a service-oriented environment to automate processes that must recur on a scheduled basis and to defer heavy processing to specific time windows. Oracle Enterprise Scheduler lets you:

- Support sophisticated scheduling and workload management
- Automate the running of administrative jobs
- Schedule the creation and distribution of reports

- Schedule a future time for a step in a business flow for business process management

7.1.1 Main Steps for Extending Oracle Enterprise Scheduler Jobs

Extending Oracle Enterprise Scheduler jobs involves the following main steps:

1. Develop the code that implements the job logic.
2. Create a metadata file for the job definition.
3. Grant **permissions** to the job, such that only those with the proper permissions can request job submission.
4. Enable job request submission, using an existing host application, a preconfigured user interface, or a new **customized** application.

7.1.2 Main Steps for Customizing Oracle Enterprise Scheduler Jobs

Customizing Oracle Enterprise Scheduler jobs involves editing job properties using Oracle Enterprise Manager Fusion Applications Control. The job properties that you can modify are described in Table 7–8.

7.1.3 Before You Begin Extending and Customizing Oracle Enterprise Scheduler Jobs

Before you extend and customize Oracle Enterprise Scheduler jobs, you should be familiar with the Oracle Fusion application architecture that enables customization, as described in Chapter 1, "Customizing and Extending Oracle Fusion Applications." You should also understand the typical workflow for working with customizations, as described in Chapter 2, "Understanding the Customization Development Lifecycle."

You will need to do the following before you can begin extending Oracle Enterprise Scheduler jobs:

- For developers:

  Set up JDeveloper. For more information, see Section 1.3.13, "Installing Customization Tools."

- For administrators:
  - Install Oracle Fusion Applications, making sure to provision Oracle Enterprise Scheduler services. For more information, see the *Oracle Fusion Applications Installation Guide*.
  - Start Fusion Applications Control. For more information about starting and using Fusion Applications Control, see the "Getting Started with Administering Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

7.2 Extending Custom Oracle Enterprise Scheduler Jobs Using Existing Oracle Fusion Applications

There are two main use cases for creating Oracle Enterprise Scheduler jobs.

Oracle Enterprise Scheduler Administrator

Administrators can create a new job definition using Oracle Enterprise Manager Fusion Applications Control console, using an existing host application. Scheduled jobs typically required by administrators include database maintenance tasks using

PL/SQL or running spawned jobs or scripts such as SQL*Plus scripts to load data into the database. After you have defined the job, use Oracle Enterprise Manager Fusion Applications Control to submit the job request.

Developer or System Integrator

When using an existing host application, use Fusion Applications Control to create Oracle Business Intelligence Publisher, PL/SQL, and spawned jobs. Use JDeveloper to create Java jobs and develop a new host application that executes a set of jobs. A Java job might invoke an ADF Business Components service or a service-oriented architecture (SOA) composite application, for example.

In cases where there is no need to repackage the host application, PL/SQL, binary, Oracle BI Publisher and Java jobs can be added to any host application. Optionally, you can execute Java jobs from a custom host application.

System integrators may want to use Fusion Applications Control to develop a job, while developers may prefer JDeveloper. Jobs are typically submitted using the scheduled request submission UI. Alternatively, it is possible to develop an Oracle Application Development Framework application with screens for submitting Oracle Enterprise Scheduler jobs.

Task: Implement the Logic for the Oracle Enterprise Scheduler Job

An Oracle Enterprise Scheduler job is a request to execute a specific task written in code or a script, such as Java, PL/SQL, spawned jobs, and so on.

An example of logic to be implemented by a scheduled job is writing particular data to a database under certain conditions, for example, daily shift schedules for a given subset of employees.

Task: Create a Job Definition Metadata File

An Oracle Enterprise Scheduler job definition specifies the type of job to be run (such as Java, PL/SQL type jobs, and so on), the host application that will run the job, and any additional required or optional parameters and properties for the job.

It is possible to create a job definition in Oracle Enterprise Manager Fusion Applications Control or JDeveloper.

The minimum required properties and parameters for each job type are as follows:

- Oracle BI Publisher jobs: Specify the `reportid` parameter. Specify Oracle BI Publisher parameters as job parameters with required validation. These can be entered by end users during request submission using the request submission user interface.

  If the report is a bursting report, identify it as such by selecting the bursting checkbox.

- PL/SQL jobs: In the job definition, specify the PL/SQL procedure that includes the job logic implementation.

  All input arguments to the PL/SQL procedure can be specified as parameters with required validation. These can be entered by end users during request submission using the request submission user interface.

- All other job types: Specify the name of the implementation logic and parameters in the job definition.

For more information about creating a job definition in Oracle Enterprise Manager Fusion Applications Control, see the "Managing Oracle Enterprise Scheduler Service and Jobs" chapter in the *Oracle Fusion Applications Administrator's Guide*.

For more information about creating a job definition in JDeveloper, see the "Working with Extensions to Oracle Enterprise Scheduler" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Grant Relevant Permissions

Grant the appropriate permissions for the application using Oracle Authorization Policy Manager.

An example of the use of relevant permissions is to grant execution permissions to a **role** so that users belonging to that role can submit the job.

For more information about granting relevant permissions to a deployed application, see the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

Task: Enable Job Request Submission

You can enable job request submissions through an Oracle ADF user interface using JDeveloper or Fusion Applications Control.

When using JDeveloper to enable job request submissions through an Oracle ADF user interface, you must define a view object to capture properties filled in by end users.

If a job is defined with properties that must be filled in by end users, the user interface allows end users to fill in these properties prior to submitting the job request. For example, if the job requires start and end times, end users can fill in the desired start and end times in the space provided by the user interface.

The properties that are filled in by end users are associated with a view object, which in turn is associated with the job definition itself. When the job runs, Oracle Enterprise Scheduler accesses the view object to retrieve the values of the properties.

You could, alternatively, submit job requests using Fusion Applications Control. Using Fusion Applications Control to enable job request submissions through an Oracle ADF user interface does not require you to create a view object for capturing end user data.

Note: Suppose a parameter view object is VLinked to another view object (VO1). If you customize the view object using JDeveloper, then the Oracle Enterprise Scheduler job submission UI list of values reflects this customization, if the customization is in the Oracle Metadata Services runtime database.

For more information about submitting job requests using Fusion Applications Control, see the "Managing Oracle Enterprise Scheduler Service and Jobs" chapter in the *Oracle Fusion Applications Administrator's Guide*.

For more information about defining a view object for use with a job submission interface, see the "Working with Extensions to Oracle Enterprise Scheduler" chapter in the *Oracle Fusion Applications Developer's Guide*.

7.2.1 Extending a Custom PL/SQL Oracle Enterprise Scheduler Job

Extending a custom PL/SQL Oracle Enterprise Scheduler job involves creating a PL/SQL package and defining job metadata.

Task: Implement the Logic for the PL/SQL Job

Implementing a PL/SQL scheduled job involves creating a PL/SQL package and defining the job metadata using the PL/SQL job type.

To implement the logic for a PL/SQL job:

1. Create a PL/SQL package, including the required `errbuf` and `retcode` arguments. A sample PL/SQL package is shown in Example 7–1.

Example 7–1 Sample PL/SQL package

```
CREATE OR REPLACE PACKAGE XxSamplePkg AUTHID CURRENT_USER AS

Procedure SampleJob (
     errbuf out NOCOPY varchar2,
     retcode out NOCOPY varchar2,
     name in varchar2 );

END XxSamplePkg;
/

CREATE OR REPLACE PACKAGE BODY XxSamplePkg AS

Procedure SampleJob (
 errbuf out NOCOPY varchar2,
 retcode out NOCOPY varchar2,
 name in varchar2 )
 IS

begin
      -- Write log file content using the FND_FILE API.
      FND_FILE.PUT_LINE(FND_FILE.LOG, 'Running Stored procedure
SampleJob.........');
      FND_FILE.PUT_LINE(FND_FILE.LOG, 'FND USERNAME : ' || FND_GLOBAL.USER_NAME);

      -- Write log file content using the FND_FILE API.
      FND_FILE.PUT_LINE(FND_FILE.OUTPUT,' Name : ' || name );
      FND_FILE.PUT_LINE(FND_FILE.OUTPUT, 'Job Request id : ' || FND_JOB.REQUEST_ID
);

      errbuf := fnd_message.get_string('FND', 'COMPLETED NORMAL');
      retcode := 0;

  end SampleJob;

END XxSamplePkg;
/
```

2. Deploy the package to Oracle Database.
3. Grant the required permissions, and perform any other necessary tasks in the database.

   ```
   grant execute on xxSampleJob to FUSION_APPS_EXECUTE;
   ```

   For more information about granting permissions for the execution of a PL/SQL job, see the "Performing Oracle Database Tasks for PL/SQL Stored Procedures" section in the *Oracle Fusion Middleware Developer's Guide for Oracle Enterprise Scheduler*.
4. Test the package.

Task: Create a Job Definition Metadata File for the PL/SQL Job

Use the Setup and Maintenance work area to define a job definition metadata file for the PL/SQL job. The job definition metadata file may also include user properties for the PL/SQL job as well as UI parameters to be displayed at runtime.

For more information about creating an Oracle Enterprise Scheduler metadata file, see the "Creating Job Definitions" section in the *Oracle Fusion Applications Administrator's Guide*.

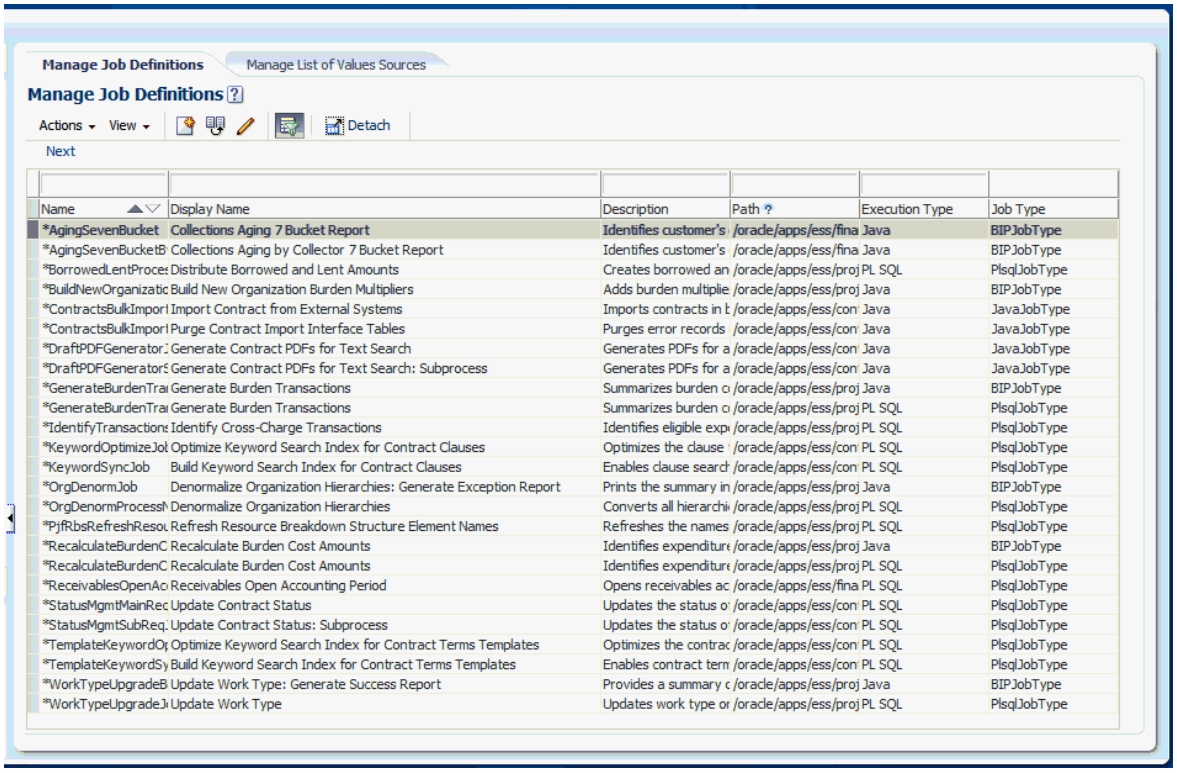
To create a job definition metadata file for a PL/SQL job:

1. From the **Administration** menu in the global area of Oracle Fusion Applications, choose the **Setup and Maintenance** work area and click the **All Tasks** tab. Search for all tasks.
2. From the list of tasks that is displayed, select the relevant UI application you will use to host the job definitions and parameter view objects. This Oracle Fusion application is the portlet producer application for the job.

   Click the **Go to Task** button.

   The Manage Job Definitions tab is displayed, as shown in Figure 7–1.

Figure 7–1 The Manage Job Definitions Tab



3. In the **Manage Job Definitions** tab, click the **New** button.
4. In the **Create Job Definition** tab, click **Show More** to display all parameters and enter the values for the job shown in Table 7–1.

Table 7–1 PL/SQL Job Definition Values

Field	Description
Display Name	Enter a display name for the job.
Name	Enter a name for the job definition.
Path	Specify the trailing package name for the job definition metadata. The default namespace or path for custom job definitions begins with `oracle/apps/ess/custom`. For example, when entering `test` in the **Path** text field, the job definition is stored in the `globalEss` MDS namespace as `oracle/apps/ess/custom/test`.
Job Application Name	From the dropdown list, choose the name of the host application running the Oracle Enterprise Scheduler job.
Job Type	Choose the job type from the dropdown list, namely the **PlsqlJobType**.
Procedure Name	Enter the name of the stored procedure to run as part of the PL/SQL job execution.
Standard request submission flag	Check this box to indicate that the job request is to be submitted in the standard manner.

5. At the bottom of the pane, click the **User Properties** tab. Define the following user properties by clicking the **New** button, as shown in Table 7–2.

Table 7–2 PL/SQL User Properties

Name	Data Type	Default Value	Read Only
`EXT_ PortletContainerWebModule`	String	For the default value, enter the name of the web module that will be used as a portlet when submitting the job request.	N/A
`numberOfArgs`	String	Set the number of job submission arguments, including `errbuf` and `retcode`.	N/A

Note: Typically, the web context is registered as the web module name. Verify with your applications administrator the value of the registered web module name in the Topology Manager work area. Registering the correct web module name enables the correct remote rendering of the Oracle Fusion application job request parameters from the Oracle Enterprise Scheduler central UI.

6. Click the ***<Job Definition Name>*: Parameters** tab and specify UI parameters as required. The UI parameter fields are described in Table 7–3.

Table 7–3 PL/SQL Job UI Parameters

Field	Description
Prompt	Enter the text to be displayed at the prompt that is displayed during runtime.
Data Type	From the dropdown list, choose the relevant data type.
Page Element	From the dropdown list, choose the UI page element you want to use to display the parameter, for example, a text box.

7. Click **Save and Close** to create and save the new Oracle Enterprise Scheduler PL/SQL job definition.

7.2.2 Extending a Custom Oracle BI Publisher Oracle Enterprise Scheduler Job

Implementing an Oracle BI Publisher scheduled job involves creating an Oracle BI Publisher report on Oracle BI Server and defining the Oracle Enterprise Scheduler job metadata.

Task: Implement the Logic for the Oracle BI Publisher Job

For information about implementing an Oracle BI Publisher job, see the "Using BI Publisher with Oracle JDeveloper" chapter in the *Oracle Fusion Middleware Developer's Guide for Oracle Business Intelligence Publisher (Oracle Fusion Applications Edition)*.

Task: Create a Job Definition Metadata File for the Oracle BI Publisher Job

Using the Setup and Maintenance work area, create an Oracle BI Publisher type job definition.

To create a job definition metadata file for an Oracle BI Publisher job:

1. Follow the instructions in Task: Create a Job Definition Metadata File for the PL/SQL Job.
2. From the **Job Type** dropdown list, choose **BIPJobType**.
3. In the **User Properties** tab, define only the **EXT_PortletContainerWebModule** property.

7.2.3 Extending a Custom Java Oracle Enterprise Scheduler Job

Implementing a Java scheduled job involves implementing the Java business logic and defining the relevant Oracle Enterprise Scheduler job metadata. Use JDeveloper to implement a Java job and deploy the job as a shared library. Modify the deployment descriptor of the deployed user interface or host application Enterprise Archive (EAR) file so that it points to the shared library. Redeploy the file.

Deploying the job as a shared library allows you to add additional jobs in the future without having to redeploy the host application. For more information about deploying Oracle ADF applications, see the "Deploying Fusion Web Applications" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Implement the Logic for the Java Job

In order to develop an application that runs a Java class under Oracle Enterprise Scheduler, you must define the Java class that implements the Oracle Enterprise Scheduler executable interface. The executable interface defines the contract that enables using Oracle Enterprise Scheduler to invoke a Java class.

To create a Java class for an existing Oracle Enterprise Scheduler Oracle Fusion application, take the following steps:

- Create an application in JDeveloper.
- Create a project in JDeveloper.
- Develop the application code that uses the Oracle Enterprise Scheduler Java APIs.

To implement the logic for an Oracle Enterprise Scheduler Java job:

1. In JDeveloper, create an application and project. Make sure to include Enterprise JavaBeans (EJB) and Java technologies in the project.
2. Add the Oracle Enterprise Scheduler extensions to the project.

a. In the Application Navigator, right-click the project you just created. Choose **Project Properties**, and then choose **Libraries and Classpath**.

b. In the **Libraries and Classpath** pane, click **Add Library**.

c. In the Add Library window, in the **Libraries** field, choose **Enterprise Scheduler Extensions** and click **OK**.

3. Create a Java class using the Oracle Enterprise Scheduler package.

a. In the project overview tab, click the **Java Files** link.

b. In the Java Files pane, click the **New** button. From the **Create New in Project** menu, choose ***Project Name*** and then choose **Java Class**.

The Create Java Class window is displayed.

c. In the Create Java Class window, enter a name for the Java class and the package name in the fields provided. For example, if working with the Financials Oracle Fusion application, the package name would be **oracle.apps.financials.ess.program**. Accept the remaining default values.

4. In the Java class, develop the code that will do the work of the Java job. Example 7–2 shows sample code that illustrates the use of an Oracle Enterprise Scheduler job request file handle and writes a job request parameter submitted to the request log and output files.

Example 7–2 Sample Java code

```
package oracle.apps.financials.ess.program;

   import java.io.IOException;
   import oracle.as.scheduler.Cancellable;
   import oracle.as.scheduler.Executable;

   import oracle.as.scheduler.ExecutionCancelledException;
   import oracle.as.scheduler.ExecutionErrorException;
   import oracle.as.scheduler.ExecutionPausedException;
   import oracle.as.scheduler.ExecutionWarningException;
   import oracle.as.scheduler.RequestExecutionContext;

   import oracle.as.scheduler.RequestParameters;
   import oracle.as.scheduler.SystemProperty;

   import oracle.as.scheduler.cp.exec.ProcessHelper;
   import oracle.as.scheduler.cp.file.LogFile;
   import oracle.as.scheduler.cp.file.OutputFile;

   public class XxSampleJob implements Executable, Cancellable {

      private OutputFile requestOutput;
      private LogFile requestLog;

      private boolean m_isCancelled = false;

      private long request_id = 0L;
      private String requestParameter1 = null;

      public XxSampleJob() {
          super();
      }
```

```
public void execute(RequestExecutionContext ctx,
                    RequestParameters params) throws ExecutionErrorException,
                                                     ExecutionWarningException,
                                                     ExecutionCancelledException,
                                                     ExecutionPausedException {

    request_id = ctx.getRequestId();

    System.out.println("XxSampleJob Running, Request ID: " +
                       ctx.getRequestId());

    try {

        String userFileDir =
                    (String)params.getValue(SystemProperty.USER_FILE_DIR);

        String sysPropUserName =
                    (String)params.getValue(SystemProperty.USER_NAME);

        // Read the job request parameter.
        requestParameter1 = (String) params.getValue("submit.argument1");

        requestOutput = ProcessHelper.getOutputFile();
        requestOutput.writeln("Sample ESS Java job execution OUTPUT");
        requestOutput.writeln("USER_NAME as SystemProperty: " +
                              sysPropUserName);
        requestOutput.writeln("ESS Job requestID: " + request_id);
        requestOutput.writeln("ESS Job request parameter: "
                                                    + requestParameter1);

        requestLog = ProcessHelper.getLogFile();
        requestLog.writeln("Sample ESS Java job execution LOG");
        requestLog.writeln("ESS requestFileDirectory: " + userFileDir);
        requestLog.writeln("ESS Job requestID: " + request_id);
        requestLog.writeln("ESS Job request parameter: "
                                                    + requestParameter1);

    }  catch (Exception ex) {

        System.out.println("Exception running XxSampleJob: " +
                           ex.getMessage());
        ex.printStackTrace();

    } finally {

        try {

            // Close all open job request log and output files.
            ProcessHelper.closeAllFiles();

        } catch (IOException ioe) {

            System.out.println("Exception closing files: " +
                               ioe.getMessage());
            ioe.printStackTrace();
        }

    }

}
```

```
    @Override
    public void cancel() {
        m_isCancelled = true;
    }

}
```

Task: Deploy the Java Business Logic

To deploy the Java logic of an Oracle Enterprise Scheduler Java job, identify an existing Oracle Fusion application as the target host application.

Next, update the Java business logic for an existing Oracle Fusion application as follows:

- Package the Java application in a Java Archive (JAR) file.
- Update JAR module in the Oracle Fusion application class path.
- Bounce the server instance to load the Java program logic in the Oracle Fusion application class loader.

To deploy the Java business logic:

1. Create a deployment profile for the project.
   a. In JDeveloper, from the Application Navigator, choose the project you created. Build the project to ensure that the Java class successfully compiles.
   b. Right-click the project, choose **Project Properties** and then **Deployment**.
   c. In the **Deployment Profiles** field, click **New** to create a deployment profile for the JAR file.

      The Create Deployment Profile window is displayed.
   d. In the Create Deployment Profile window, enter a name for the deployment profile and click **OK**.
   e. In the Edit JAR Deployment Profile Properties window, verify that the Java job class is included in the JAR module output and click **OK**.
2. Package the Oracle Enterprise Scheduler Java class into a JAR file and deploy it.
   a. From the Application Navigator, right-click the project you created. Choose **Deploy** and then choose the JAR file you just created.

      The Deployment Action window is displayed.
   b. In the Deployment Action window, click **Finish** to create a packaged JAR file.

      The archive module is deployed to the default project deployment path, for example, **$JDEV_HOME**/<**PROJECT_NAME**>/deploy/<**JAR_NAME**>.jar.

      Note: All custom JAR files must begin with the prefix Xx, for example XxMyJar.jar.
3. Update the JAR module in the application class path of the Oracle Enterprise Scheduler host application.
   a. Locate the expanded deployment directory of the EAR file for the existing Oracle Fusion application, for example **$MW_**

`HOME`/fusionapps/applications/fin/deploy/EarFinancialsEss.ear/APP-INF/lib.

b. Copy the deployed custom JAR file to the expanded directory.

4. In the domain to which the Oracle Enterprise Scheduler host application is deployed, restart Oracle Enterprise Scheduler.

   The Oracle Enterprise Scheduler job executes the updated Java class after the application class loader successfully loads the updated class.

   For more information about restarting Oracle Enterprise Scheduler, see the "Starting and Stopping Oracle Enterprise Scheduler Service Components" section in the *Oracle Fusion Applications Administrator's Guide*.

Task: Create a Job Definition Metadata File for the Java Job

Using the Setup and Maintenance work area, create a Java type job definition.

To create a job definition metadata file for a Java job:

1. Follow the instructions in Task: Create a Job Definition Metadata File for the PL/SQL Job.
2. In the Create Job Definition window, from the **Job Type** dropdown list, choose **JavaJobType**.
3. In the Create Job Definition window, in the **Class Name** field, enter the fully qualified class name of the Java business logic.
4. In the Create Job Definition window, In the **User Properties** tab, define only the **EXT_PortletContainerWebModule** property.
5. Click the **<*Job Definition Name*>: Parameters** tab and specify UI parameters as required. The UI parameter fields are described in Table 7–3.
6. Click **Save and Close** to create and save the new Oracle Enterprise Scheduler Java job definition.

7.2.4 Submitting Oracle Enterprise Scheduler Jobs

You can use Oracle Fusion Applications to submit Oracle Enterprise Scheduler jobs.

To submit Oracle Enterprise Scheduler jobs:

1. In the global area of Oracle Fusion Applications, access the Schedule Processes page by clicking the **Navigator** menu and then choosing **Tools** and **Schedule Processes**.
2. Click **Schedule New Process**.

   The Search and Select: Process Name window is displayed.
3. In the **Process Name** field, enter the name of the Oracle Enterprise Scheduler job you want to schedule and click **Search**.

   The job name is displayed in the search results table.
4. From the search results table, choose the job name and click **OK**.

   The Process Details page is displayed.
5. In the Process Details page, in the **Parameters** field, specify any required parameters.

6. Click **Submit** to request that the Oracle Enterprise Scheduler instance execute the job. Click **Close** to return to the Scheduled Processes page.
7. In the Scheduled Processes page, refresh the Search Results table to monitor the status of the submitted job.

7.3 Creating a Custom Oracle Enterprise Scheduler Application to Extend Oracle Enterprise Scheduler Jobs

Use Apache Ant scripts to develop and deploy an Oracle Enterprise Scheduler host application and user interface. Use JDeveloper to create the relevant metadata.

7.3.1 Creating Host and UI Applications Using an Ant Script

Use the supplied Ant script to create the host and user interface applications for the Oracle Enterprise Scheduler jobs.

When deploying the application, be sure to identify the product family and use an existing registered Oracle WebLogic Server domain. This allows you to test your application without having to create and register a domain, or register any associated applications with the product family.

To create host and user interface applications using scripts:

1. Extract the Oracle Enterprise Scheduler `customer_extensibility` script from the JDeveloper installation or JDeveloper extensions to the development work environment, for example, into a folder called `template_home`.

   The `template_home` directory contains an Ant `build.xml` driver file that processes the template Oracle Enterprise Scheduler host and producer web applications by replacing macros with specified input.
2. Change directories to the `template_home` directory to create the `user_home` directory that will contain the resulting macro-substituted files copied from the `template_home` directory.
3. Run the script in any of the following ways:
   - Interactively, where you are prompted for the relevant input. Accept the default, if there is one, by pressing Enter at each prompt.

     In the `template_home` directory, enter `ant` or `ant create-user-home`. A sample running script is shown in Example 7–3.

Example 7–3 Interactive Script Execution

```
$ ant
 Buildfile: build.xml
-init:

 create-user-home:
    [input] Enter which template should be used (source_template) (default=Fusion)
    [input]      ([Fusion], Standalone)
                   Fusion
    [input] Enter Middleware Home Directory path (fmw_home_dir) (default=) []
            /JDEVADF_INSTALLATION/
    [input] Enter host application name (hosting_application_name) (default=MyAppEss) [MyAppEss]
            MyAppEss
    [input] Enter host application JPS stripe id (hosting_application_stripe_id)
            (default=MyAppEss) [MyAppEss]
            MyAppEss
```

```
    [input] Enter UI application name (ui_application_name) (default=MyApp) [MyApp]
            MyApp
    [input] Enter UI application JPS stripe ID (ui_application_stripe_id) (default=MyApp) [MyApp]
            MyAppEss
    [input] Enter the shared library name for the job business logic (jobdef_library_name)
            (default=MyJobsLibrary) [MyJobsLibrary]
            oracle.ess.shared
    [input] Enter an empty directory where the applications will be created (user_home)
            /workspace/ess_user_home
     [echo]
     [echo]
    [mkdir] Created dir: /workspace/ess_user_home
[propertyfile] Creating new property file: /workspace/ess_user_home/template.properties
     [copy] Copying 31 files to /workspace/ess_user_home
     [copy] Copied 36 empty directories to 14 empty directories under /workspace/ess_user_home
     [copy] Copying 19 files to /workspace/ess_user_home
     [move] Moving 1 file to /workspace/ess_user_home/Template_Hosting
     [move] Moving 1 file to /workspace/ess_user_home/Template_UI
     [echo]
     [echo] ========================================
     [echo]
     [echo] A new application workspace has been created at: /workspace/ess_user_home
     [echo] This application workspace can be opened and modified using JDeveloper
     [echo] To deploy the applications, run the following command:
     [echo]     ant -f /workspace/ess_user_home/ant/build-ess.xml deploy
     [echo] To create new jobs from predefined templates, run the following command:
     [echo]     ant -f /workspace/ess_user_home/build.xml create-new-job-def

BUILD SUCCESSFUL
```

- Using predefined property files. Any properties not defined in a file can be entered at the prompt. A sample properties file is shown in Example 7–4. To create a properties file, run the command `$> cat myProperties.properties`, where `myProperties.properties` is the name of the properties file.

Example 7–4 Script Execution Via Property Files

```
user_home=/home/myuser/ess_user_home/

 ui_application_name=MyApp
 ui_application_stripe_id=MyApp
 ui_application_version=V2.0

 hosting_application_name=MyAppEss
 hosting_application_stripe_id=MyAppEss
 hosting_application_version=V2.0

 jobdef_library_name=oracle.ess.sharedlibrary
 jobdef_library_spec_version=11
 jobdef_library_impl_version=11.1.1.5.0
```

Then run the following command:

```
$> ant create-user-home -propertyfile myProperties.properties
```

- Specifying individual properties at the command line. Any properties not defined in a file can be entered at the prompt. A sample is shown in Example 7–5.

Example 7–5 Script Execution Via the Command Line

```
$> ant create-user-home -Dui_application_name=MyApp -Dhosting_application_
name=MyAppEss
```

To view supported options, enter `ant help-create-user-home` at the prompt.

4. On successful execution, you can modify the template application workspace from the `user_home` directory in JDeveloper.

   At the prompt, enter `ant help-deploy` to list the supported deployment options.

7.3.2 Generating an Oracle Enterprise Scheduler Synchronous Java Job Business Logic Template

If you want to run a synchronous Java scheduled job, then you must develop the business logic for the job. Use the `build.xml` file extracted in Section 7.3.1 to create a template for the business logic of the Java job.

To generate an Oracle Enterprise Scheduler Java job business logic template:

1. To create new jobs from predefined templates, run the following command:

   ```
   ant -f ${ess_user_home_dir}/build.xml create-new-job-def
   ```

2. When prompted, enter the Oracle Enterprise Scheduler job name, for example, `HelloSyncJavaJob`, and the package name, for example, `oracle.apps.financials.ess.program`.

   Note: Ensure that the full job package name is unique across product families.

   A sample command execution is shown in Example 7–6.

Example 7–6 Creating a Java Job Business Logic Template

```
Buildfile: /workspace/ess_user_home/build.xml

-init:

create-new-job-def:
    [echo] Available Job Definition Templates:
    [echo]     1) Simple Synchronous Java Job
   [input] Enter number of job definition template to create (job_template_to_create)
 1
    [echo] Calling default target on /my_ess_main/ess/util/customer_extensibility/Fusion/
           Template_JobLibrary/simple_synchronous_job/build.xml

-init:

create-job-definition:
   [input] Enter Java package name for Job Definition (jobdef_package_name)
           (default=oracle.apps.ess.custom) [oracle.apps.ess.custom]
           oracle.apps.financials.ess.program
   [input] Enter Java class name for Job Definition (jobdef_class_name)
           (default=MySynchronousJavaJob) [MySynchronousJavaJob]
           HelloSyncJavaJob
    [copy] Copying 1 file to /workspace/ess_user_home/MyAppEss/EssSharedLibrary/src
```

```
    [copy] Copying 1 file to /workspace/ess_user_home/MyAppEss/EssSharedLibrary/src/oracle/
           apps/financials/ess/program

BUILD SUCCESSFUL
```

3. In JDeveloper, open the Oracle Enterprise Scheduler host application project saved to the `user_home` application workspace location.
4. In the Navigator, expand the **EssSharedLibrary Model** project to modify the template-generated Java job business logic.
5. Modify the file as required and save your changes.

7.3.3 Creating Oracle Enterprise Scheduler Job Metadata Using JDeveloper

To submit job requests using the Oracle Enterprise Scheduler host application, you must create metadata that defines a job request, including the following:

- **Job type**: This specifies an execution type and defines a common set of parameters for a job request.
- **Job definition**: This is the basic unit of work that defines a job request in Oracle Enterprise Scheduler.

7.3.3.1 Creating an Oracle Enterprise Scheduler Job Definition in the Host Application

To use a Java class with Oracle Enterprise Scheduler you must create a job definition. When creating a job definition, specify a name, choose a job type, and specify system properties.

To create a job definition in the host application:

1. In the Application Navigator, right-click the **EssSharedLibrary** project and choose **New** to display the New Gallery window.
2. In the New Gallery in the Categories area, expand **Business Tier** and choose **Enterprise Scheduler Metadata**.
3. From the New Gallery Items area, choose **Job Definition** and click **OK**.

   The Create Job Definition window is displayed.
4. In the Create Job Definition window, specify the following:
   - In the **Name** field, enter a name for the job definition. For example, for the scheduler host application, enter **SampleJob**.
   - In the **Package** field, enter a package name. For example, enter `oracle/apps/ess/custom/test`.
   - From the **Job Type** dropdown list, choose **JavaJobType**.

   Click **OK**. The job definition `SampleJob.xml` is created, as well as the jobs folder in the package `oracle/apps/ess/custom/test`. The Job Definition page is displayed.
5. In the Job Definition page, specify the fully qualified class name of the template-generated Java business logic created in Section 7.3.2.
6. Next to the Class Name field, choose the **Overwrite** checkbox.
7. In the **Class Name** field, enter the name of the Java class you created, for example, `oracle.apps.financials.ess.program.HelloSyncJavaJob`.

8. In the System Properties section, click the **Add** button and create a system property called **EffectiveApplication**. Set the value of the property to the host application name, for example, `MyAppEss`.
9. In the Parameters section, define the following required properties:
   - **jobDefinitionName**: The short name of the job. For example, `SampleJob`.
   - **jobDefinitionApplication**: The short name of the host application running the job. For example, `MyAppEss`.
   - **jobPackageName**: The name of the package running the job. For example, `/oracle/apps/ess/custom/test`.
   - **srsFlag**: A boolean parameter (Y or N) that controls whether the job is displayed in the job request submission user interface. Enter **Y**.
   - **EXT_PortletContainerWebModule**: The name of the web module for the Oracle Enterprise Scheduler Central UI application to use as a portlet when submitting a job request. For example, `MyApp`, or any producer web application (if you prefer to use an existing registered web module that hosts the Oracle ADF view objects).
   - **parametersVO**: The ADF Business Components view object you define so that end users may enter additional properties at runtime through an Oracle ADF user interface. For example, `oracle.apps.financials.ess.SampleVO`. For more information about creating a view object in the Oracle ADF producer application, see Task: Create an ADF Business Components View Object for Oracle Enterprise Scheduler.

7.3.3.2 Creating a Schedule Request Submission UI to Enable End Users to Fill in Properties

If your job includes any properties to be filled in by end users at runtime, you need to create an Oracle ADF user interface and an ADF Business Components view object with validation, and the parameters to be filled in. The Oracle Enterprise Scheduler schedule request submission UI allows end users to fill in these properties prior to submitting the job request.

For more information about Oracle ADF view objects, see the "Creating a Business Domain Layer Using Entity Objects" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Task: Create an Oracle ADF Model Project

Create an Oracle ADF model project to display the properties to be filled in by end users at runtime.

To create an Oracle ADF model project:

1. In JDeveloper, open the Oracle Enterprise Scheduler Oracle ADF application.
2. From the **Application** menu, choose **New Project**.
3. In the New Gallery under Categories, expand **General** and choose **Projects**.
4. In the Items area, choose **ADF Model Project** and click **OK**.
5. On the Name Your Project wizard page, enter the project name, for example **EssModel**. Click **Finish** to close the wizard.
6. From the Application Navigator, right-click the **EssModel** project and choose **Project Properties**, then **Libraries and Classpath**, and then **Add Library**.

7. Add the required data model project libraries as described in the chapter "Setting Up Your JDeveloper Application Workspace and Projects" in the *Oracle Fusion Applications Developer's Guide*.
8. Click **OK** to close the Project Properties dialog.

Task: Create an ADF Business Components View Object for Oracle Enterprise Scheduler

Use a parameters view object for jobs with parameters that require collecting values from end users at runtime. The properties filled in by end users are associated with an ADF Business Components view object, which is associated with the job definition itself. When the job runs, Oracle Enterprise Scheduler accesses the view object to retrieve the values of the properties.

To create an ADF Business Components view object for Oracle Enterprise Scheduler:

1. In JDeveloper in the Application Navigator, right-click the project **EssModel** in which you want to create the view object, and choose **New**.
2. In the New Gallery, expand **Business Tier**, choose **ADF Business Components** and then **View Object**. Click **OK**.

   If this is the first component you are creating in the project, then the Initialize Business Components Project dialog is displayed, allowing you to choose a database connection.
3. In the Initialize Business Components Project dialog, choose the database connection or choose **New** to create a connection.

   Click **OK**. This launches the Create View Object wizard.
4. In the Create View Object wizard on the Name page, enter the following.
   - **Package**: Enter package information for the view object, for example `oracle.apps.financials.ess`.
   - **Name**: Provide a name, for example, `SampleVO`.
   - **Select the data source type you want to use as the basis for this view object**: For the data source, choose **Rows Populated Programmatically, Not Based on a Query**.

   Note: Enter the view object package and name values specified for the job definition property `parametersVO` in Section 7.3.3.1.
5. Click **Next**. In the Attributes page, click **Finish** to create the Oracle Enterprise Scheduler parameter view object `SampleVO`.
6. Define attributes for the view objects sequentially, `ATTRIBUTE1`, `ATTRIBUTE2`, and so on, with an attribute for each required parameter.
7. Create a query for the view object:
   a. On the View Object page, from the left-hand list panel, choose **Query**.
   b. In the Query panel, click the **Edit** icon.
   c. Use the following query and test for validity:

      ```
      select null as ATTRIBUTE1 from dual
      ```

   d. Click **OK**.

> **Note:** A maximum of 100 attributes can be used for the property `parametersVO`. The attributes should be named incrementally, for example `ATTRIBUTE1`, `ATTRIBUTE2`, and so on. Attribute names are not case-sensitive, such that `ATTRIBUTE1` and `Attribute2` can be used sequentially.

8. Ensure that the view object attributes can always be updated:
   a. On the View Object page, from the left-hand list panel, choose **Attributes**.
   b. Edit the `ATTRIBUTE1` table row.
   c. In the Edit Attribute: Attribute1 window, select the option **Always**.
   d. In the Edit Attribute: Attribute1 window, click **Control Hints** to display the Control Hints page. In the Control Hints page, specify required prompts, validation, and formatting for each parameter.
   e. Click **OK**.
9. If not already specified, add the property `parametersVO` to your Oracle Enterprise Scheduler host application job definition and specify the fully qualified path of the view object as the value of `parametersVO`. For example, set `parametersVO` to `oracle.apps.financials.ess.SampleVO` in the job definition `/oracle/apps/ess/custom/test/SampleJob.xml`.

```
<parameter name="parametersVO"
data-type="string">oracle.apps.financials.ess.SampleVO</parameter>
```

7.3.4 Assembling Oracle Enterprise Scheduler Oracle Fusion Applications

Assembling the Oracle Enterprise Scheduler Oracle Fusion applications involves the following main steps:

- Assembling an Oracle Enterprise Scheduler shared library
- Assembling the host application
- Assembling the Oracle ADF producer application

Task: Assemble an Oracle Enterprise Scheduler Shared Library

Assembling a shared library for Oracle Enterprise Scheduler involves the following main steps:

- Creating or updating a shared library JAR manifest
- Updating the shared library JAR deployment profile

The name and version information for a shared Java EE library are specified in the `META-INF/MANIFEST.MF` file.

To assemble a shared library:

1. Specify attributes for the shared library in a manifest file.
   a. Create or edit the manifest file in a text editor.
   b. Enter the following command:

```
cd <ess_user_home>/MyAppEss/EssSharedLibrary/emacs MANIFEST.MF
```

c. Add or edit a string value to specify the name of the shared Java EE library. For example:

```
Extension-Name: oracle.ess.shared
```

`Extension-Name` specifies the name of the shared Java library. Use the value specified in the script prompt for the shared library name. Oracle Enterprise Scheduler host applications that reference the library must specify `Extension-Name` exactly to use the shared files.

As a best practice, enter the optional version information for the shared Java EE library. A sample `MANIFEST.MF` file is shown in Example 7–7.

Example 7–7 Sample MANIFEST.MF File

```
Extension-Name: oracle.ess.shared
Specification-Version: 11.1.0
Implementation-Version: 11.1.0.0.0
```

d. Save the file. The `MANIFEST` file is used by the JAR deployment file.

2. Compile the project. In the Application Navigator, right-click the Oracle Enterprise Scheduler shared library project and choose **Make EssSharedLibrary*.jpr***.
3. Right-click the Oracle Enterprise Scheduler shared library project and choose **Project Properties** to display the Project Properties window.
4. In the Project Properties window, choose **Deployment**.
5. In the Deployment Profiles region, choose **EssSharedLibrary (Shared Library JAR File)**.
6. Click **Edit** to open the Edit JAR Deployment Profile Properties window.
7. In the Edit JAR Deployment Profile Properties window, click **JAR Options**.
8. In the JAR Options window, choose the checkbox **Include Manifest File (META-INF/MANIFEST.MF)**.
9. Click **Add** to specify the manifest file you created. This file should be merged into the manifest file that is generated by JDeveloper.
10. In the Edit JAR Deployment Profile Properties window, expand **File Groups** and choose **Filters**. Under the **Merged Contents of this File Group's Contributors** list, deselect **essmeta**.
11. In the JAR Deployment Profile Properties page, click **OK**.
12. In the Project Properties page, click **OK**.

Task: Assemble the Host Application

Assembling the host application involves the following main steps:

- Creating a MAR deployment file
- Updating the EAR deployment file

To assemble the host application:

1. Open the Oracle Enterprise Scheduler host application in JDeveloper, and from the **Application** menu, choose **Application Properties**.
2. In the Application Properties window, choose **Deployment**.
3. Click **New** to display the Create Deployment Profile page and do the following:

a. In the **Archive Type** field, from the dropdown list, choose **MAR File**.

b. In the **Name** field enter a name, for example **myAppEss_MAR**.

c. Click **OK**.

4. In the Edit MAR Deployment Profile Properties window, choose **MAR Options**.
5. Modify the name of the MAR file, removing `_MAR` from the end of the name, for example, changing `myAppEss_MAR.mar` to `myAppEss.mar`.
6. Choose the Oracle Enterprise Scheduler metadata:

a. In the Edit MAR Deployment Profile Properties window, expand **Metadata File Groups** and choose **User Metadata**.

b. In the Order of Contributors panel on the right-hand side, click **Add** to display the Add Contributor dialog.

c. In the Add Contributor dialog, browse to the location of the project directory, and expand it to add the `essmeta` metadata that contains the namespace for the jobs directory. Choose the path that you need to include in the Add Contributor dialog by double-clicking the **essmeta** directory.

d. In the Add Contributor dialog, click **OK**.

7. Choose the directory for the metadata:

a. In the Edit MAR Deployment Profile Properties window, expand **Metadata File Groups and User Metadata**, and choose **Directories**.

b. Choose the directory that contains the Oracle Enterprise Scheduler application user metadata for the host application.

c. Choose the bottommost directory in the tree. This is the directory from which the namespace is created. The folder you choose in this dialog determines the top-level namespace in `adf-config.xml` file.

d. This namespace should be the same as the package defined in the job definition, for example `oracle/apps/ess/custom/<directory name>`.

Note: In general, to create the namespace `oracle/apps/<product>/<component>/ess`, choose the **ess** directory.

---

e. In the Edit MAR Deployment Profile Properties page, click **OK**.

8. In the Application Properties window, choose **Deployment**.
9. In the Deployment Profiles pane on the right-hand side, choose the EAR profile and click **Edit**.
10. In the Edit EAR Deployment Profile Properties window, choose **Application Assembly**.
11. Under Java EE Modules, choose the checkbox for the MAR module.
12. In the Edit EAR Deployment Profile Properties window, choose **EAR Options**.
13. Deselect **Include Manifest File (META-INF/MANIFEST.MF)**.
14. In the Edit EAR Deployment Profile Properties page, click **OK**.
15. In the Application Properties page, click **OK**.

Task: Assemble the Oracle ADF Producer Application

Assembling the Oracle ADF application involves the following main steps:

- Creating an ADF Library job deployment file
- Preparing a Web Application Archive (WAR) deployment profile

Oracle ADF libraries have the option of automatic compilation that happens with deployment profile dependencies. This option allows the Oracle Enterprise Scheduler Oracle ADF Library used by the user interface project to be automatically included in the `WEB-INF/lib` directory in the WAR file.

To assemble the Oracle ADF producer application:

1. Open the Oracle Enterprise Scheduler Oracle ADF application in JDeveloper.
2. In the Application Navigator, right-click the **EssModel** project and click **New** to display the New Gallery window.
3. In the New Gallery in the Categories area, expand **General** and choose **Deployment Profiles**. Create the deployment profile as follows:
   a. In the Items region, choose **ADF Library Jar File**.
   b. Click **OK** to open the Create Deployment Profile window.
   c. In the Create Deployment Profile - ADF Library Jar File window, enter a name for the profile, using the format `Adf<projName>` in accordance with package structure and naming standards.
   d. Click **OK** to save the new deployment profile and close the Create Deployment Profile window.
4. In the Application Navigator, right-click the **SuperWeb** project and choose **Project Properties**, and then **Deployment**.
5. In the Deployment Profiles region, edit the SuperWeb WAR deployment profile.
6. In the Edit WAR Profile Deployment Properties window, choose **Profile Dependencies**.
7. In the pane on the right-hand side, under Java EE Modules, choose the dependency under the ADF library JAR deployment file (`EssModel.jpr`), for example, **ADFMyApp**.
8. Click **OK** to save the WAR deployment profile.

7.3.5 Deploying Oracle Enterprise Scheduler Oracle Fusion Applications

Deploying Oracle Enterprise Scheduler Oracle Fusion applications involves the following main steps. You must deploy the Oracle Enterprise Scheduler Oracle Fusion application in the order specified.

1. Deploy the shared Oracle Enterprise Scheduler library using JDeveloper or an Ant script.
2. Deploy the Oracle Enterprise Scheduler host application using JDeveloper or an Ant script.
3. Deploy the Oracle Enterprise Scheduler Oracle ADF producer application using JDeveloper or an Ant script.

Application-specific policies packed with script-generated host and Oracle ADF applications automatically migrate to the policy store when the application is

deployed. Prior to deployment, verify that any grant of permissions in the application `jazn-data.xml` file contains no duplicates.

For more information about securely deploying applications, see the "Deploying Secure Applications" chapter in the *Oracle Fusion Middleware Application Security Guide*.

Task: Deploy the Shared Oracle Enterprise Scheduler Library Using JDeveloper

You can deploy the shared Oracle Enterprise Scheduler library using JDeveloper or an Ant script.

To deploy the share library using JDeveloper:

1. In the Application Navigator, right-click the Oracle Enterprise Scheduler shared library project, choose **Deploy** and then choose the shared library JAR.

   The Deploy EssSharedLibrary_JAR window is displayed.

2. Choose **Deploy to a WebLogic Application Server** and click **Next**.
3. In the Select Server window, choose the application server to which you want to deploy the Oracle Enterprise Scheduler shared library.
4. Click the **Add** button to create a connection to the application server if none is defined. Click **Next**.
5. In the WebLogic Options window, make the following selections:
   a. Choose **Deploy to selected instances in the Domain**, and choose the Oracle Enterprise Scheduler server instance in the table row. The Oracle Enterprise Scheduler shared library should be deployed to the same server as the Oracle Enterprise Scheduler host application.
   b. Choose **Deploy as a shared library**.
   c. Click **Finish**.
6. Verify the deployment using the deployment log. Upon successful deployment, you can see the Oracle Enterprise Scheduler jobs shared library deployed as 'oracle.ess.shared(11,11.1.1)' in the Oracle WebLogic Server Administration Console.

Task: Deploy the Shared Oracle Enterprise Scheduler Library Using an Ant Script

To deploy the shared library using an Ant script:

1. Run the following Ant command:

   ```
   ant -f ${ESS_HOME}/ant/build-ess.xml deploy_job_logic
   ```

   The command `deploy_job_logic` builds, packages and deploys only the Oracle Enterprise Scheduler jobs shared library.

   Note: When prompted, enter the Oracle WebLogic Server password.

2. To specify a different value for the ESS shared library name, take the following steps:
   a. In a text editor, modify the shared library JAR `MANIFEST` file. For example:

   ```
   vi ${ess_user_home_dir}/MyAppEss/EssSharedLibrary/MANIFEST.MF
   ```

b. Edit the string value of `Extension-Name` to specify the name of the shared Java EE library.

c. Enter the optional version information for the shared Java EE library.

d. Update the Oracle Enterprise Scheduler `build.properties` file by editing `${ESS_HOME}/ant/config/ess-build.properties`.

e. Change the value of the property `customEss.shared.library.name` to match the value specified in the JAR `MANIFEST` file. A sample `build.properties` file is shown in Example 7–8.

Example 7–8 Sample build.properties File

```
# ESS build properties
 ess.script.base.dir=${user_home}

 fmw.home=${fmw_home}
 jdev.home=${fmw.home}/jdeveloper
 oracle.common=${fmw.home}/oracle_common

 # ========== ESS JDev project details ===============
 customEss.project.dir=${ess.script.base.dir}
 customEss.hostapp.workspace=${hosting_application_name}
 customEss.hostapp.jwsfile=${hosting_application_name}
 customEss.hostapp.earprofile=${hosting_application_name}
 customEss.hostapp.jprproject=EssSharedLibrary
 customEss.hostapp.jarprofile=EssSharedLibrary
 customEss.hostapp.jarfile=${jobdef_library_name}

 customEss.shared.library.name=${jobdef_library_name}

 customEss.hostapp.mds.partition=globalEss
 customEss.hostapp.mds.jdbc=mds-ApplicationMDSDB
 customEss.hostapp.name=${hosting_application_name}

 customEss.producerapp.workspace=${ui_application_name}
 customEss.producerapp.jwsfile=${ui_application_name}
 customEss.producerapp.earprofile=${ui_application_name}
 customEss.producerapp.name=${ui_application_name}

 # ========== WebLogic Server details ===============
 MW_HOME=${fmw.home}
 ORACLE_HOME=${jdev.home}
 MW_ORA_HOME=${jdev.home}
 COMMON_COMPONENTS_HOME=${oracle.common}
 WEBLOGIC_HOME=${fmw.home}/wlserver_10.3
 weblogic.server.host=<server_host>
 weblogic.server.port=<server_port>

 weblogic.server.ssl.port=<server_ssl_port>

 weblogic.admin.user=<admin_username>
 weblogic.t3.url=t3://${weblogic.server.host}:${weblogic.server.port}
 # WebLogic server name where ESS producer web application is targeted for
 # deployment
 adfapp.server.name=AdminServer
 # WebLogic server name where ESS host application is targeted for deployment
 ess.server.name=ess_server1
```

f. Save the file.

Task: Deploy the Oracle Enterprise Scheduler Host Application Using JDeveloper

You can deploy the Oracle Enterprise Scheduler application using JDeveloper or an Ant script.

To deploy the Oracle Enterprise Scheduler host application using JDeveloper:

1. In JDeveloper, open the Oracle Enterprise Scheduler host application.
2. From the **Application** menu, choose **Deploy** and then choose the name of the host application, for example **MyAppEss**.
3. In the Deploy MyAppEss window, choose **Deploy to Application Server** and click **Next**.
4. In the Select Server window, choose the application server to which you want to deploy the Oracle Enterprise Scheduler host application.

   Click the **Add** button to create a connection to the application server if none is defined.
5. Click **Next**. In the WebLogic Options window, make the following selections:

   a. Choose **Deploy to selected instances in the Domain**, and choose the Oracle Enterprise Scheduler server instance in the table row, to which the Oracle Enterprise Scheduler host application is to be deployed.

   b. Choose **Deploy as a standalone Application**.

   c. Click **Finish**.

   JDeveloper displays the Deployment Configuration page. Choose the relevant options for your metadata repository.
6. Click **Deploy**.

   Verify the deployment using the deployment log.

   Upon successful deployment, you can expect to see the Oracle Enterprise Scheduler host application deployed in Fusion Applications Control.

Task: Deploy the Oracle Enterprise Scheduler Host Application Using an Ant Script

To deploy the Oracle Enterprise Scheduler host application using an Ant script:

- Run the following Ant command:

  ```
  ant -f ${ESS_HOME}/ant/build-ess.xml deploy_ess_host
  ```

  The command `deploy_ess_host` builds, packages, and deploys only the Oracle Enterprise Scheduler host application. It is assumed that the Oracle Enterprise Scheduler shared job library is already deployed prior to running this command.

  Note: When prompted, enter the Oracle WebLogic Server password.

Task: Deploy the Oracle ADF Producer Application Using JDeveloper

You can deploy the Oracle ADF producer application using JDeveloper or an Ant script. This step is optional if using an existing deployed producer web application. The value you defined for **EXT_PortletContainerWebModule** in Section 7.3.3.1 indicates the name of the application to be used.

To deploy the Oracle ADF producer application using JDeveloper:

1. In JDeveloper, open the Oracle ADF producer application.
2. From the **Application** menu, choose **Deploy** and then choose the name of the Oracle ADF producer application.
3. In the Deploy MyApp window, choose **Deploy to Application Server** and click **Next**.
4. In the Select Server window, choose the application server to which you want to deploy the Oracle Enterprise Scheduler Oracle ADF application.
5. Click the **Add** button to create a connection to the application server if none is defined.
6. Click **Next**. In the WebLogic Options window, make the following selections:
   a. Choose **Deploy to selected instances in the Domain**, and choose the Oracle Enterprise Scheduler server instance in the table row, to which the Oracle Enterprise Scheduler Oracle ADF application is to be deployed.
   b. Choose **Deploy as a standalone Application**.
   c. Click **Finish**.
   d. The Select Deployment Type dialog window is displayed, prompting you to expose the MyApp portlet application as a WSRP service. Choose **Yes**.
7. Click **Next**. The Deployment Configuration page is displayed. Choose the relevant options for your metadata repository.
8. Enter **globalEss** as the partition name.
9. Click **Deploy**.
10. Verify the deployment using the deployment log.

    Upon successful deployment, you can expect to see the deployed Oracle Enterprise Scheduler Oracle ADF application in Fusion Applications Control.
11. Open the WSRP Producer test page to validate the deployment using the following URL:

    ```
    http://<ADF_HOST>:<ADF_PORT>/<MyApp-context-root>/
    ```

Task: Deploy the Oracle ADF Producer Application Using an Ant Script

To deploy the Oracle ADF producer application using an Ant script:

- Run the following Ant command:

  ```
  ant -f ${ess_user_home_dir}/ant/build-ess.xml deploy_ess_ui
  ```

  The `deploy_ess_ui` command builds, packages, and deploys only the Oracle Enterprise Scheduler Oracle ADF producer application.

  Note: When prompted, enter the Oracle WebLogic Server password.

7.3.6 Registering Oracle Enterprise Scheduler Topology Objects

Registering Oracle Enterprise Scheduler topology objects involves the following main steps:

- Creating Oracle Enterprise Scheduler topology objects

- Registering Oracle Enterprise Scheduler topology objects

> **Note:** Register the topology objects only when using an Ant script-generated Oracle ADF producer web application. Alternatively, you can use an existing registered web or Oracle Enterprise Scheduler Oracle ADF producer application and skip this section.

Task: Create Oracle Enterprise Scheduler Topology Objects

Use the Setup and Maintenance work area to create Oracle Enterprise Scheduler topology objects, including the Oracle Enterprise Scheduler domain, host application, and Oracle Enterprise Scheduler Oracle ADF producer application.

To create Oracle Enterprise Scheduler topology objects:

1. Create the Oracle Enterprise Scheduler domain topology object.
   a. In the global area of Oracle Fusion Applications, from the **Administration** menu, choose the **Setup and Maintenance** work area.
   b. From the Tasks Pane, choose **Topology Objects** and then choose **Manage Domains**.
   c. On the Manage Domains page in the list of domains, click the **Actions** dropdown list and choose **Create**.
   d. In the Create Domain window that is displayed, enter a name for the domain and click **Save and Close**.
2. Create the Oracle Enterprise Scheduler host application topology object:
   a. In the global area of Oracle Fusion Applications, from the **Administration** menu, choose the **Setup and Maintenance** work area.
   b. From the Tasks Pane, choose **Topology Objects** and then choose **Manage Enterprise Applications**.
   c. On the Manage Enterprise Applications page in the list of domains, click the **Actions** dropdown list and choose **Create**.
   d. In the Create Enterprise Application page, enter the details in Table 7–4.

Table 7–4 Enterprise Application Topology Object Details

Field	Description
Name	Enter the name of the enterprise application that you want to register, for example **EarCustomHostEss**.
Code	Enter a unique code to identify the enterprise application. After you have created it, the code cannot be changed.
Domain	Choose the name of the domain to be used by the enterprise application, for example **EssDomain**.
Default URL	Enter a static URL if the enterprise application is always to be deployed at the same location. Optional.
Source File	Enter the name of the EAR file. Optional.
Pillar	From the Available Pillars list, shuttle the relevant pillar or pillars to the Selected Pillars list.

   e. Click **Save and Close** to create the Oracle Enterprise Scheduler host application topology object.

3. Repeat Step 2 to create the Oracle Enterprise Scheduler Oracle ADF producer application topology object.

Task: Register Oracle Enterprise Scheduler Topology Objects

Use the Setup and Maintenance work area to register the Oracle Enterprise Scheduler topology objects you created.

To register Oracle Enterprise Scheduler topology objects:

1. Register the Oracle Enterprise Scheduler domain.
   a. In the global area of Oracle Fusion Applications, from the **Administration** menu, choose the **Setup and Maintenance** work area.
   b. From the Tasks Pane, choose **Topology Registrations** and then choose **Register Domains**.
   c. On the Register Domains page in the list of domains, click the **Actions** dropdown list and choose **Create**.
   d. In the Add Domain window that is displayed, enter the details for the Oracle Enterprise Scheduler domain created in "Task: Create Oracle Enterprise Scheduler Topology Objects" as described in Table 7–5.

Table 7–5 Domain Registration Values

Field	Description
Enterprise Environment	From the dropdown list, choose the enterprise environment to be used, for example `oracle`.
Domain	From the dropdown list, choose the name of the domain.
Name/Administrator Server Name	Enter a name for the registered domain. Enter a name for the domain's administration server.
Internal/External/Administrator Server Host/Port/Protocol	Enter the URL, port number, and protocol (such as HTTP, HTTPS, and so on) for the internal server to be registered, as well as the external server and the administration server.
Enterprise Manager Protocol	From the dropdown list, choose the protocol to be used for accessing Oracle Enterprise Manager, for example HTTP or HTTPS.
Enterprise Manager Port	Enter the port number to be used when accessing Oracle Enterprise Manager in the domain.
Java Management Extensions Port	Enter the port number to be used for Java management extensions.

   e. Click **Save and Close** to save your changes.
2. Register the Oracle Enterprise Scheduler web producer module.
   a. In the global area of Oracle Fusion Applications, from the **Administration** menu, and choose the **Setup and Maintenance** work area.
   b. From the Tasks Pane, choose **Topology Objects** and then choose **Manage Modules**.
   c. On the Manage Modules page from the list of applications, click the **Actions** dropdown list and choose **Register Modules**.
   d. In the Register Modules window that is displayed, enter the details as shown in Table 7–6.

Table 7–6 Domain Registration Values

Field	Description
Name	Enter the name of the module that you want to register.
Code	Enter a unique code to identify the module. After you have created it, the code cannot be changed.
Description	Enter a brief, meaningful description of the module. Optional.
Enterprise Application	Choose and associate the enterprise application to which the module belongs.
Type	Choose the relevant module type from the list.
Context Root	Enter the context root of the module.

e. Click **Save and Close** to save your changes.

3. Register the Oracle Enterprise Scheduler host and producer applications.

a. In the global area of Oracle Fusion Applications, from the **Administration** menu, choose the **Setup and Maintenance** work area.

b. From the Tasks Pane, choose **Topology Registrations** and then choose **Register Enterprise Applications**.

c. On the Register Enterprise Applications page from the list of applications, click the **Actions** dropdown list and choose **Add**.

d. In the Add Enterprise Application window that is displayed, enter the details in Table 7–7.

Table 7–7 Domain Registration Values

Field	Description
Enterprise Environment	From the dropdown list, choose the enterprise environment to be used, for example `oracle`.
Enterprise Application	Choose and associate the enterprise application to which the module belongs.
Name	Enter the name of the enterprise application.
External Server Protocol/Host/Port	Enter the URL, port number, and protocol (such as HTTP, HTTPS, and so on) for the external server to be registered with the enterprise application.

e. Click **Save and Close** to save your changes.

f. In the Register Enterprise Applications page, click the **Actions** dropdown list and choose **Add** to display the Add Enterprise Application window.

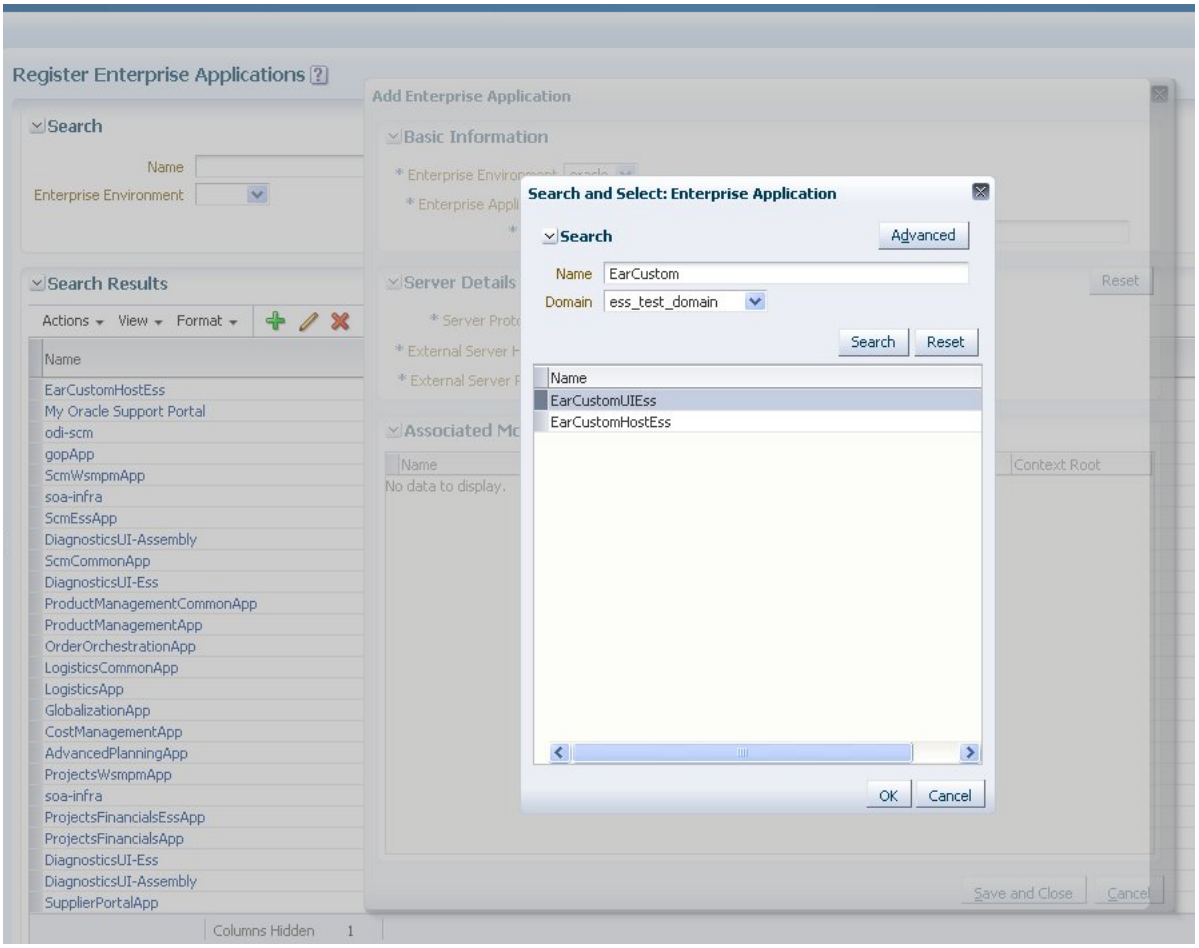
g. Click the Enterprise Application dropdown list to display the Search and Select: Enterprise Application window.

h. In the **Name** field, enter a name for the application you want to search for and click the **Domain** dropdown list to choose the domain in which you want to search.

Click **Search** to search for the Oracle Enterprise Scheduler producer web application.

i. From the list of enterprise applications that is displayed, choose the relevant Oracle Enterprise Scheduler producer web application and click **OK**, as shown in Figure 7–2.

Figure 7–2 Choose the Relevant Enterprise Application



j. In the Add Enterprise Application page, fill in the details for the Oracle Enterprise Scheduler producer web application as described in Table 7–7.

k. Click **Save and Close**.

7.3.7 Granting Job Metadata Permissions to Application Roles and Users

You can use Oracle Authorization Policy Manager to manage **application roles** and resource-based policies. Identifying the application roles and users, and granting them the required **privileges** to execute Oracle Enterprise Scheduler job-related tasks is a one-time operation.

Granting Oracle Enterprise Scheduler metadata permission to the new job involves the following main steps:

- Creating a new resource for the custom job definition
- Creating a new policy

Task: Create a Resource

In Oracle Authorization Policy Manager, create an application resource instance.

To create a resource:

1. Run Oracle Authorization Policy Manager by entering the following URL in a browser.

   ```
   http://<fs-domain_url>/apm/
   ```

2. From the navigator pane, right-click the application **Resources** icon and choose **New**.

   An untitled page is displayed.

3. Define a resource with the resource type **ESSMetadataResourceType**, as well as the name and display name of the Oracle Enterprise Scheduler component using the following syntax:
   `oracle.apps.ess.`*`applicationName`*`.`*`JobDefinitionName`*`.`*`JobName`*.

4. Save the resource.

Task: Define a Policy

Define a policy that specifies the privileges allocated to a particular user when submitting the job request.

To define a policy:

1. In Oracle Authorization Policy Manager in the **Home** tab, under the Applications region, choose an application for which you want to manage the policy, for example, **MyAppEss**.
2. Click **Search Policies** to display the Search Authorization Policies tab.
3. In the **Search Authorization Policies** tab, choose the principal user on which to base the policy being created, for example, **FinUser1**.
4. In the **Functional Security** tab, choose **Resource Based Policies**.
5. Click **New Policy** to create a new policy for the selected user.
6. Add resource instances to the policy by clicking the **Add** button in the Resources table.
7. Select the resource instance created for the custom Oracle Enterprise Scheduler job definition (from "Task: Create a Resource").
8. Specify the actions `EXECUTE` and `READ` to provision Oracle Enterprise Scheduler job execution privileges to the user.
9. Click **Save**.

Task: Test Oracle Enterprise Scheduler Job Submission from the Oracle Enterprise Scheduler Central UI

Submit a job request to ensure that everything works as it should.

To submit a test job request:

1. Log in to Functional Setup Manager with the user for whom you defined an authorization policy, for example, as **FinUser1**.

   The URL for Functional Setup Manager is as follows:

   ```
   https://<HOST>/setup/faces/TaskListManagerTop
   ```

2. From the **Tools** menu, choose **Schedule Processes**.

3. Click the **Schedule New Process** button and choose a job process name when prompted. Select the job definition you created.
4. Click **OK**.

   The Oracle Enterprise Scheduler Schedule Request Submission window is displayed.
5. In the Parameters region, specify the job parameters as required.
6. Click **Submit** to schedule the job execution, and **Close** to exit the window.
7. Refresh the Search Results table to monitor the status of the submitted job.

7.4 Customizing Existing Oracle Enterprise Scheduler Job Properties

You can customize Oracle Enterprise Scheduler jobs that are associated with Oracle Fusion applications. Customizing existing Oracle Enterprise Scheduler jobs involves editing job properties using Oracle Enterprise Manager Fusion Applications Control.

An example of a customization is to set the timeout value for a scheduled job to be run asynchronously. When the job takes longer than the timeout, you can find the job that timed out in Fusion Applications Control and manually complete the job.

The job properties that can be edited are shown in Table 7–8.

For more information about editing scheduled job properties, see the "Managing Oracle Enterprise Scheduler Service and Jobs" chapter in the *Oracle Fusion Applications Administrator's Guide*.

Table 7–8 Job Properties

API	Description
`oracle.as.scheduler.SystemProperty.PRIORITY`	This property specifies the request processing priority, from 0 to 9, where 0 is the lowest priority and 9 is the highest. If this property is not specified, the system default value used is `oracle.as.scheduler.RuntimeService#DEFAULT_PRIORITY`.
`oracle.as.scheduler.SystemProperty.RETRIES`	This property defines the numerical value that specifies the retry limit for a failed job request. If job execution fails, the request retries up to the number of times specified by this property until the job succeeds. If the retry limit is zero, a failed request will not be retried. If this property is not specified, the system default used is `oracle.as.scheduler.RuntimeService#DEFAULT_RETRIES`.
`oracle.as.scheduler.SystemProperty.REQUEST_CATEGORY`	This property specifies an application-specific label for a request. The label, defined by an application or system administrator, allows administrators to group job requests according to their own specific needs.

Table 7–8 (Cont.) Job Properties

API	Description
`oracle.as.scheduler.SystemProperty.ASYNC_REQUEST_TIMEOUT`	This property specifies the time in minutes that the job request processor waits for an asynchronous request after it has begun execution. After the time elapses, the job request times out.
`enableTrace`	The property specifies a numerical value that indicates the level of tracing control for the job. Possible values are as follows: ■ 1: Database trace ■ 5: Database trace with bind ■ 9: Database trace with wait ■ 13: Database trace with bind and wait ■ 16: PL/SQL profile ■ 17: Database trace and PL/SQL profile ■ 21: Database trace with bind and PL/SQL profile ■ 25: Database trace with wait and PL/SQL profile ■ 29: Database trace with bind, wait, and PL/SQL profile
`enableTimeStatistics`	This property enables or disables the accumulation of time statistics.

8

Customizing Security for Oracle ADF Application Artifacts

This chapter describes how to customize security for custom and extended business objects and related custom and extended application artifacts defined by Oracle Application Development Framework (Oracle ADF) in Oracle Fusion applications. Developers customize security using Oracle Authorization Policy Manager and Oracle JDeveloper.

Security customization in the production environment is typically restricted to the security administrator using Oracle Authorization Policy Manager; however, during the development phase of application customization, you can perform similar security customization tasks using Oracle Authorization Policy Manager and JDeveloper.

This chapter includes the following sections:

8.1 About the Oracle Fusion Security Approach

Oracle Fusion Applications is secure as delivered. The Oracle Fusion security approach tightly coordinates various security concerns of the enterprise, including:

- The ability to define security policies to specify the allowed operations on application resources, including viewing and editing data and invoking functions of the application.
- The ability to enforce security policies by using roles assigned to end users, and not by directly enforcing those policies on the end users of the system.

A **role** is an identity that end users are anticipated to fill when interacting with Oracle Fusion Applications that specifically determines the user's permitted access to data and application functions. For example, when an end user attempts to access a task flow, whether or not the end user has the right to enter the task flow and view the contained web pages is specified by the roles provisioned to the end user and the security policies defined for those roles.

In the enterprise, the security administrator ensures end users are provisioned with the privileges to perform the duties of their various jobs. A **privilege** determines the user right to access data and application functions of Oracle Fusion applications. The provisioning tasks involve Oracle Fusion Middleware tools that integrate with Oracle Fusion Applications and allow IT personnel to extend the security reference implementation. These tools directly update a copy of the security reference implementation in the deployed application's security policy store and identity store. The **security reference implementation** provides role-based access control in Oracle Fusion Applications, and is composed of predefined security policies that protect functions, data, and segregation of duties.

From the standpoint of application developers who seek to apply the Oracle Fusion security approach to an Oracle Fusion application that they extend, the security implementation process overlaps with tasks performed by IT personnel. You may or may not need to extend the Oracle Fusion Applications security reference implementation, depending upon how end users will interact with the new resource. At the end of the process, you must ensure that any new resource you create, such as a **business object** in the data model project or a task flow in the user interface project, has sufficient security policies to grant access privileges and suitable roles to receive the access privileges.

8.1.1 How to Proceed with This Chapter

Customizing security is a complex process that involves working with several tools, familiarity with diverse technologies, and coordination between the application developer and security administrator. For a concise summary of the security customization scenarios and corresponding tasks, see Table 8–1 in Section 8.3.3, "Oracle Fusion Security Customization Scenarios."

After familiarizing yourself with the types of security customizations performed by the application developer, read the following sections to gather a more complete understanding of the security customization process:

- For an overview of the Oracle Fusion Applications security reference implementation, see Section 8.2.
- For a list of security guidelines that dictate which security artifacts in the Oracle Fusion Applications security reference implementation you may or may not modify, see Section 8.3.1.
- For an overview of the steps you follow to secure a new resource, see Section 8.3.2.
- For additional background about the type of resource customizations that require customizing security, see Section 8.3.4 and Section 8.3.5.
- For details about the security artifacts that you create to define security policies, see Section 8.3.6 through Section 8.3.9.
- For a list of tasks that may be performed only by a security administrator, see Section 8.3.10.
- For a list of prerequisite tasks to be completed before customizing security, see Section 8.3.11.
- For information about the tools involved in customizing security, see Section 8.4 through Section 8.6.

8.1.2 Related Security Documents

The following related documents contain important information specific to customizing security in Oracle Fusion Applications. References to these documents appear throughout this chapter. Consult these documents for complete details.

- *Oracle Fusion Applications Security Guide*

  Describes the concepts and best practices of the Oracle Fusion security approach. This is the main document addressing the Oracle Fusion security approach.

- *Oracle Fusion Applications Security Hardening Guide*

  Describes how security administrators proceed to implement the Oracle Fusion Applications security reference implementation for their enterprise.

- *Oracle Fusion Applications security reference manuals*

  Describes the segregation of duties in the Oracle Fusion Applications security reference implementation. Each Oracle Fusion application has its own reference manual.

- *Oracle Fusion Applications Developer's Guide*

  Describes how to secure new custom resources in Oracle Fusion Applications. Includes chapters describing how to implement **data security** and **function security** for new resources.

- *Oracle Fusion Applications Administrator's Guide*

  Summarizes available security administration tasks in a single chapter.

- *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*

  Describes how to define and modify data security policies and data role templates.

- *Oracle Fusion Middleware Application Security Guide*

  Describes the concepts and best practices of Oracle Platform Security Services (OPSS) upon which Oracle Fusion security is based. This is the main document addressing the architecture of Oracle security services.

- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

  Describes ADF Security, through which the components of Oracle ADF interact with OPSS.

- *Oracle Fusion Middleware User's Guide for Oracle Identity Manager*

  Describes role provisioning and other identity management tasks.

- *Oracle Database Security Guide*

  Describes implementing security policies at the level of the database.

- JDeveloper online help topics

  Describes the tools used to create database objects using JDeveloper.

8.2 About Extending the Oracle Fusion Applications Security Reference Implementation

The Oracle Fusion Applications security approach is embodied in the security reference implementation, which delivers predefined roles and security policies that

address the common business needs of the enterprise. The reference implementation can be extended to adjust to the needs of a specific enterprise. The predefined security policies implement role-based access control: a set of roles recognizable as jobs, a role hierarchy that contains the duties for those jobs, and a set of role provisioning events and workflows. The Oracle Fusion Applications security reference implementation represents what Oracle considers to be the general security guidelines for jobs, roles, duties, and segregation of duties.

In general, the Oracle Fusion Applications security reference implementation is designed to require only small changes to adjust Oracle Fusion security for a specific enterprise. The security reference implementation provides a comprehensive set of predefined security policies and predetermined data role templates that may be customized to generate security policies. From the standpoint of security administrators who address the specific security concerns of their organizations, typical tasks include changing or extending role definitions and role hierarchies, and managing security policies and data role templates. For example, enterprise IT security administrators eventually review the duties and access defined in the security reference implementation and specify how that matches with the job titles and tasks the enterprise expects to be performed in the deployed Oracle Fusion application.

A security administrator provisions end users with role membership, and defines the provisioning in the application's identity store. This configuration task is performed independent of security customization. The Oracle Fusion Applications security reference implementation contains four types of roles: duty, job, data, and abstract, and implements hierarchies between these roles to streamline provisioning access to end users. Each of the Oracle Fusion Applications roles is implemented in Oracle Fusion Middleware as one of the following roles:

- Internal roles are roles that are not assigned directly to end users. An internal role is also called an **application role** because it is specific to an application.

  Note that, in Oracle Fusion Applications, application roles are called **duty roles**. The security reference implementation defines a large number of duty roles that correspond to the duties of individual job roles. Duty roles are specific to applications, stored in the policy store, and shared within an Oracle Fusion Applications instance. For example, in your enterprise, the job of an application developer may also include project management duties. The duty role is a role that corresponds to a line on a job description for that job.

- External roles are roles associated with a collection of end users and other groups. They are also called **enterprise roles** because they are shared across the enterprise.

  In Oracle Fusion Applications, enterprise roles include:

  - The **job role** is a role that corresponds to a job title defined in human resources (HR).
  - The **data role** is a role that authorizes a person with a job to a particular dimension of data on which they can work. For example, the data role AP Manager - US Commercial Business Unit identifies who may access the accounts specific to the US division of the enterprise.
  - The **abstract role** is a role that is not a job title, but is a means to group end users without respect to specific jobs, for example, Employee and Line Manager are both abstract roles.

The division between internal roles and external roles is an important principle of the Oracle Fusion security approach. The principle, called **least privilege**, ensures that the end user acquires privileges specific only to the job they perform rather than to a variety of miscellaneous duties. Therefore, in adherence to the principle of least

privilege, duty roles are defined by Oracle Platform Security Services (OPSS) as internal roles and cannot be assigned directly to end users.

To understand the Oracle Fusion security approach in detail and to learn more about using the Oracle Fusion security infrastructure to implement and administer security for the enterprise, see the "Introduction" chapter in the *Oracle Fusion Applications Security Guide*.

8.3 About Extending and Securing Oracle Fusion Applications

Oracle Fusion Applications is configured by default to deny end users access to the data of the application domain and the web pages that display the data. An important principle of Oracle Fusion security ensures that end users do not have unintended access to data and application artifacts exposed in the extended application.

To enable access to custom resources in the extended application, you may define security policies to specify "who can perform what operations on what specific data and on what specific application artifacts."

> **Note:** The term **protected** in this chapter refers to the default Oracle Fusion Applications condition that denies end users access to database resources and application artifacts. In contrast, the term **secured** refers to resources that have been made accessible to end users through security policies created for this purpose. Therefore, a security policy specifically **enables access** to the resource based on the privileges it confers to the end user.

To define the security policy, you must consider the additional duties the end users of the extended application will perform, and then grant the required roles the specific privileges to:

- Access the web pages of a custom task flow that supports the duty
- Access the specific data records, or instances of a custom business object, required to complete the duty
- Perform only those operations on that data required by the duty

When you need to secure new resources, you can expect to work with two different types of security policies: **data security** policies that control access to the data records of database tables or views in the Oracle Fusion Applications schema, and **function security** policies that control access to the Oracle Fusion application artifacts used to display the data. Because the representation of data security policies and function security policies differs, the environment you will use to define security policies depends on whether data security or function security is being implemented.

In the case of access to data records, a custom business object may be secured either explicitly or implicitly. For example, the AP Manager is authorized to an explicit list of business units specified by a data role, whereas the Project Manager is implicitly authorized to the projects that he manages. When you need to secure data records, then you can:

- Implicitly grant data access to abstract and job roles through data security policies you define on custom duty roles inherited by the abstract or job role.

  You can create custom duty roles to support a new duty introduced by a custom application resource.

- Explicitly grant data access to a data role through a data security policy you apply directly to the inherited job or abstract role using a data role template.

  You can customize the data role template before running the template to generate the data roles.

8.3.1 Oracle Fusion Security Customization Guidelines for New Functionality

In general, when you create new functionality, not supported by Oracle Fusion Applications, do not include authorization to that functionality from within the security artifacts that Oracle Fusion Applications delivers in the security reference implementation.

Specifically, Oracle Fusion security guidelines suggest customization developers and security administrators *must not modify* the following security artifacts in the security reference implementation when introducing *new functionality*, through custom or extended business objects:

- Predefined duty roles, specifically:
  - Do not change the role hierarchy by removing member duty roles assigned to parent duty roles or job roles.
  - Do not remove (also called **revoke**) existing privileges granted to duty roles.
  - Do not add (also called **grant**) new privileges to duty roles.
- Predefined security policies (including data and function), specifically:
  - Do not remove existing instance sets from predefined data security policies.
  - Do not remove existing member resources from predefined function security policies.
  - Do not revoke existing actions (mapped by Oracle Fusion security to resource operations) granted on each resource or instance set.

Customization developers and security administrators *may modify* security artifacts in the security reference implementation in the following ways:

- Do modify job roles to add a custom duty role (permissible by security administrator only).
- Do modify data role templates to add a new job role as the base role or to add access privileges to a custom business object.

Customization developers and security administrators *may create* the following security artifacts and add them to the security reference implementation:

- Do create custom duty roles when a custom application resource requires a new duty role to support the segregation of duties or when a custom application resource introduces new privileges to a predefined business object.
- Do create data role templates when a custom business object is used as a data stripe and when explicit data security policies grant access to the data stripe. A **data stripe** is a dimensional subset of the data granted by a data security policy and associated with a data role. For example, create a data role template when you need to grant data roles access to a specific business unit or organization.

Note: You must not modify predefined duty roles, and you must always add custom duty roles to grant access privileges. Only the security administrator can add or remove duty roles associated with an existing job role. If a predefined job role that adequately describes the duties performed by a job does not exist, then the security administrator can also create a new job role.

8.3.2 Oracle Fusion Security Customization Process Overview

Creating a new, custom business object and exposing it in the extended application is one of the main customization tasks that you may perform. Although you may also extend existing business objects to introduce new functionality or to introduce additional data, the security customization process for new and existing business objects follows a similar pattern.

To secure a *new* business object in the extended Oracle Fusion application:

1. Create a custom duty role to serve as the grantee of the security policy privileges.

   For details about creating duty roles, see the "Managing Policies and Policy Objects" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

2. Define a database resource in the Oracle Fusion Data Security repository to protect the data records of a database table that you intend to expose in the application.

   For details about registering a database table as a database resource, see Section 8.3.6, "What You Can Customize in the Data Security Policy Store at Design Time."

   This step causes Oracle Fusion security to protect the database table records, thus rendering the data inaccessible to the end user of the application. A data security policy will be required to grant access to the data defined by the database resource and a function security policy will be required to grant access to the application artifacts that display the data in the extended application.

3. Define data security policies for the previously defined database resource to grant access to specific data records for a given role.

   For details about securing data, see Section 8.3.6, "What You Can Customize in the Data Security Policy Store at Design Time."

4. Extend the data model project (in the extended application) with a new ADF entity object to expose the database table that you defined as an Oracle Fusion Data Security database resource.

   For details about creating custom ADF business components to represent a database table, see Chapter 4, "Customizing and Extending Oracle ADF Application Artifacts."

5. Opt into the previously defined data security policies by enabling OPSS authorization checking on the operations of individual data model objects in the data model project.

   For details about enabling security, see Section 8.3.7, "What You Can Customize in the Data Model Project at Design Time."

6. Consult a security administrator to export all predefined function security policies of the application that you are customizing into a `jazn-data.xml` file.

For details about how the security administrator exports the application policy store, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

7. Copy the exported `jazn-data.xml` file into your application workspace.

   For details about adding the file to your application, see the "Implementing Function Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

8. Customize the Oracle ADF application artifacts in the user interface project to display the data records exposed by the extended data model.

   For details about creating securable custom application artifacts, see Chapter 4, "Customizing and Extending Oracle ADF Application Artifacts."

9. Define function security policies for the custom Oracle ADF application artifacts to specify the access privileges of end users.

   For details about securing application functions, see Section 8.3.9, "What You Can Customize in the Application Security Policy Store at Design Time."

10. Opt into the previously defined function security policies by running the ADF Security wizard to enable OPSS authorization checking.

    For details about enabling security on the user interface project, see Section 8.3.8, "What You Can Customize in the User Interface Project at Design Time."

8.3.3 Oracle Fusion Security Customization Scenarios

You do not need to customize security for every type of customization that you may make in the extended application. Whether or not a security policy is needed will depend on the application resource and the type of customization performed.

Table 8–1 summarizes the security customization scenarios that Oracle Fusion security supports. The "Application Developer Tasks" column of the table provides a brief description of the security artifacts involved in each scenario, but presumes some familiarity with the Oracle Fusion security approach (for guidance see Section 8.1.1, "How to Proceed with This Chapter").

Note: For simplicity, Table 8–1 does not make a distinction between explicit and implicit data security policies. You may also need to customize data role templates when a custom business object is used as a data stripe and explicit data security policies grant access to that data stripe. For more details about customizing data role templates, see Section 8.3.6, "What You Can Customize in the Data Security Policy Store at Design Time."

Table 8–1 Oracle Fusion Applications Security Customization Scenarios

Security Customization Goal	Security Policy Requirement	Application Developer Tasks
Control whether the end user associated with a particular role may access a *new task flow* and view all the web pages of the flow.	Define a new security policy. The new task flow will be inaccessible by default (also called protected) and will require a new function security policy to grant end users access.	Enable ADF Security on the user interface project to protect all task flows (and the web pages they contain). Then, in the file-based policy store, create a resource definition for the task flow and assign the definition as a member of an **entitlement** (defined in the policy store as a **permission set**) that you name. Then, define the security policy by granting the entitlement to a custom application role that you either created or consulted with a security administrator to create for you. As a security guideline, do not modify a predefined function security policy by granting additional entitlements to a predefined duty role.
Control whether the end user associated with a particular role may access a *customized task flow* and view the new or customized web pages of the flow.	Do not define a security policy. The customized Oracle Fusion application task flow will already have a function security policy defined by the security reference implementation; because this type of change does not require new duties, there is no need to grant access to a new duty role.	Consult the security administrator who can make a customized task flow accessible to additional end users through role provisioning. If the same group of end users requires access to the customized task flow, then no change to the provisioned end users is required.
Control whether the end user associated with a particular role may access a *new top-level web page*. In Oracle Fusion Applications, a top-level web page is one that is not contained by a task flow.	Define a new security policy. The new top-level web page will be inaccessible by default (also called protected) and will require a new function security policy to grant end users access. The ability to secure individual web pages in Oracle Fusion Applications is reserved for top-level web pages backed by an ADF page definition file only.	Enable ADF Security on the user interface project to protect all top-level web pages backed by ADF page definition files. Then, in the file-based policy store, create a resource definition for the web page and assign the definition as a member of an entitlement (defined in the policy store as a permission set) that you name. Then, define the security policy by granting the entitlement to a custom application role that you either created or consulted with a security administrator to create for you. As a security guideline, do not modify a predefined function security policy by granting additional entitlements to a predefined duty role.
Control whether the end user associated with a particular role may access a *customized top-level web page*. In Oracle Fusion Applications, a top-level web page is one that is not contained by a task flow.	Do not define a security policy. The customized top-level web page will already have a function security policy defined by the security reference implementation; because this type of change does not require new duties, there is no need to grant access to a new duty role.	Consult the security administrator who can make customized top-level web pages accessible to additional end users through role provisioning. If the same group of end users requires access to the web page, then no change to the provisioned end users is required.
Decide whether the end user associated with a particular role has the right to select the *create, edit, or delete button* in the displayed web page.	Do not define a security policy. Access to user interface components, such as buttons, is not controlled by a security policy, but can be controlled by rendering the button in the user interface based on the end user's role.	Conditionally render the component by entering an Expression Language (EL) expression on the `rendered` attribute of the button using ADF Security utility methods to test whether the end user has membership in a particular role.

Table 8–1 (Cont.) Oracle Fusion Applications Security Customization Scenarios

Security Customization Goal	Security Policy Requirement	Application Developer Tasks
Control whether the end user associated with a particular role may view or update a *specific set of data records for an all new business object* in the displayed web page.	Define a new security policy. After an Oracle Fusion Data Security database resource is defined for the data, the data records exposed by the new business object will be inaccessible by default (also called protected) and will require a new data security policy to grant end users read or update access on one or more specific sets of data records.	Enable authorization checking on the appropriate operations of the ADF entity object (`read`, `update`, and `removeCurrentRow`) that maps to a specific database table. Then, in the Oracle Fusion Data Security repository, add a custom duty role as the grantee of access privileges and create a named instance set of data records. Then, define the security policy by granting Oracle Fusion Data Security view or update privileges to the custom duty role for the data records. As a security guideline, do not modify a predefined data security policy by granting additional privileges to a predefined duty role.
Control whether the end user associated with a particular role may view or update a *new set of data records for an existing business object* in the customized web page.	Define a new security policy. Although an existing Oracle Fusion business object will have an existing data security policy, you must not modify privileges granted to predefined duty roles (those defined by the security reference implementation) and you must instead grant privileges only to custom duty roles that they define.	In the Oracle Fusion Data Security repository, add a custom duty role as the grantee of access privileges and create a named instance set for the new data records. Then, define the security policy by granting Oracle Fusion Data Security view or update privileges to the custom duty role for the data records. As a security guideline, do not modify a predefined data security policy by granting additional privileges to a predefined duty role.
Control whether the end user associated with a particular role may view or update *new sensitive data* exposed on a new attribute of an existing business object in the customized web page. **Sensitive data** is defined as any personally identifiable information (PII) that is considered "public within the enterprise" (also called "internally public"). Internally public PII data is secured from access external to the enterprise.	Define a new security policy. Sensitive PII data exposed by a new attribute that is added to an existing Oracle Fusion application business object will be secured by the business object's data security policies and will require a new data security policy to grant end users read or update access on a specific column of data.	Column-level OPSS authorization checking is not supported for ADF entity objects. Instead create a custom OPSS **permission** to control access to the column read or update operation, and then, in the Oracle Fusion Data Security repository, map the operation to a custom privilege and grant the privilege to the custom duty roles for the sensitive data records. Last, conditionally render the attribute by testing whether the end user has the custom privilege either 1.) by entering an EL expression on the user interface component that displays the attribute or 2.) by entering a Groovy expression on the ADF view object to which the user interface component is bound.

Table 8–1 (Cont.) Oracle Fusion Applications Security Customization Scenarios

Security Customization Goal	Security Policy Requirement	Application Developer Tasks
Control whether the end user associated with a particular role may view or update *new confidential data* exposed on a new business object that the customized web page displays. **Confidential data** is defined as any personally identifiable information (PII) that is considered "private within the enterprise." Exposure of such information outside the enterprise could result in harm, such as loss of business, benefit to a competitor, legal liability, or damaged reputation. Confidential PII data is secured from access external to the enterprise and is secured additionally to prevent access within the enterprise even by highly privileged end users (such as database administrators).	Define a new security policy. In Oracle Database, the Virtual Private Database (VPD) feature only supports securing a set of data records and therefore will require a new table in a custom schema that you create for Oracle Fusion Applications. The confidential PII data exposed by the new business object will be inaccessible by default (also called protected) and will require a new data security policy to grant end users read or update access on a specific set of data records.	Column-level policies are not supported by Virtual Private Database (VPD). Instead, the database administrator must create a new table in your custom schema for Oracle Fusion Applications, create a view for that table, and then define a VPD policy to filter the PII data records by associating a PL/SQL function with that view. Then, in the Oracle Fusion Data Security repository, create an action with the same name as the database view and define the security policy by granting Oracle Fusion Data Security view or update privileges to the custom duty role for the confidential data records. Last, in the data model project, enable OPSS authorization checking on the appropriate operations of the ADF entity object (`read`, `update`, and `removeCurrentRow`) that maps to the new PII database table.

8.3.4 Scenarios Related to Extending and Securing Data Model Components

In Oracle Fusion Applications, when you want to extend the application to expose additional data, you create an ADF entity object and implement the operations that may be performed over a particular set of data records. The ADF entity object you create encapsulates the data as business object instances, corresponding to data records from a database table or view, such as an invoice or a purchase order. Typical operations are business functions like viewing, editing, or creating an instance of the business object.

Security concerned with controlling the operations that can be performed against specific data is called **data security**. Data security policies involve granting an end user, by means of the end user's membership in a role, the ability to perform operations on specific sets of data. For example, an accounts payable manager in the enterprise's western regional office may be expected to view and edit invoice data records, but only for the customers in the western region. The Accounts Payable Manager role provisioned to the accounts payable manager authorizes access to the business functions required to view and edit invoice instances, and, in this case, the specific instances of the invoice business object that is striped for the western region.

Data security policies are implemented using Oracle Fusion Data Security, which is the technology that implements the security repository for data security policies. Oracle Fusion Data Security is implemented as a series of Oracle Fusion Applications database tables, sometimes referred to as **FND tables** (note that FND refers to resources in foundation tables) and includes tables like FND_OBJECTS that defines the protected database resource and FND_GRANTS that defines the access privileges for those database resources.

To protect the business object in the extended application, where it has been exposed as an ADF entity object, a database resource definition in the FND_OBJECTS table identifies the same table or view backing the ADF entity object. The **database resource** in Oracle Fusion Data Security is the data resource on which data security is enforced.

After the business object is defined as an Oracle Fusion Data Security database resource, then a security policy must be created to grant access to the data records. The security policies for the database resource specify access privileges such as read, update, and delete privileges on specific sets of data records exposed by the business object.

Note: When an ADF entity object exposes a business object that does *not* require security, then no database resource for that business object needs to be defined in the Oracle Fusion Data Security repository. For complete details about Oracle Fusion Data Security, see the "Implementing Oracle Fusion Data Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

As an Oracle Fusion Applications security guideline, a new data security policy must be created instead of modifying predefined data security policies of the security reference implementation. For example, a new data security policy is required to expose additional data records or operations for an existing business object. Additionally, a custom duty role must be created as the recipient of the new data security access privileges because granting privileges to a predefined duty role would alter the segregation of duties defined by the security reference implementation.

Note: Developers are not entitled to modify the role hierarchy defined by the Oracle Fusion Applications security reference implementation. Therefore, whenever you create a new duty role, you must consult the security administrator to assign the custom duty role to a job role or data role.

Additionally, the security reference implementation uses database-level security policies to protect most of the confidential **personally identifiable information** (PII), also called internally private data, that exists in the Oracle Fusion Applications schema. This type of security is implemented in Virtual Private Database (VPD) policies directly on the PII tables. In general, database administrators and other personnel with access to the database must not modify VPD policies implemented for Oracle Fusion Applications. However, when you create a business object that introduces confidential data and that data needs to be treated as internally private within the enterprise, then certain roles may be granted access to the confidential data for valid business reasons. For example, a human resources representative may require access to the employee's home addresses, while a dispatcher may require access to the home telephone numbers of on-call staff.

Whether or not you will need to define a data security policy to grant access to data records depends on the type of customization, as summarized in Table 8–1. The scenarios for defining data security policies include the following.

When a new business object is introduced and it needs to be secured:

When you seek to secure additional data records in the extended application because a new ADF entity object is introduced, then an Oracle Fusion Data Security database resource must be defined to protect the data records and a new data security policy must be created to grant end users access to the data records exposed by the business object that the ADF entity object defines. The data records exposed by the business object will be unprotected (accessible to all end users) until a database resource identifying the business object is defined in the Oracle Fusion Data Security repository.

Note that the operations to be secured on the new business object will also require enabling OPSS authorization checking for those operations on the ADF entity object in the data model project, as described in Section 8.3.7, "What You Can Customize in the Data Model Project at Design Time."

When a new business object attribute is introduced and it maps to sensitive data:

When you modify an existing ADF entity object to include a new attribute that maps to data that not all end users need to view, then a new data security policy must be defined to grant end users access to the sensitive data. This is accomplished through a combination of a data security policy that grants a custom privilege and enforcement of the privilege in the application source.

Because Oracle Fusion Data Security does not support automatic enforcement of custom data security privileges, column-level security is not supported by default. You enforce the custom privilege in the application source by enabling OPSS authorization checking at the level of the user interface component or its databound ADF view object. Otherwise, without the custom data security privilege and custom privilege check, the data records (including the sensitive fields) exposed by the business object would be secured by the data security policy that already exists for the business object.

Important: Oracle Fusion Data Security alone will not prevent sensitive data from being accessed by highly privileged end users, such as database administrators. If the data needs to be treated as internally private (confidential data), then consider implementing additional security using Virtual Private Database (VPD) policies. However, do not implement column-level VPD policies to protect sensitive data exposed by attributes, because security for attributes is not supported by VPD in Oracle Fusion Applications.

When a new business object attribute is introduced and it maps to confidential data:

When you create an ADF entity object that introduces data that is to be treated as confidential to the enterprise, then define row-level VPD policies to control access to PII data by privileged users, including database administrators. Implementing VPD policies requires saving the confidential information in a new table in a custom schema for Oracle Fusion Applications.

In this case, the database administrator first creates the database table and the VPD policy to secure the PII data records. The VPD policy the database administrator creates associates a policy function (a PL/SQL function) with a particular view or synonym definition in the database. The policy function filters the rows for any query made against the PII data. Finally, you can define the actual data security policies by granting to an action that has been created with same name as the database view where the policy is defined.

For information about creating tables in a custom schema for Oracle Fusion Applications, see Section 4.8, "Customizing and Extending the Oracle Fusion Applications Schemas."

For information about creating VPD policies, see the "Using Oracle Virtual Private Database to Control Data Access" chapter in the *Oracle Database Security Guide*.

When new operations or new data records are introduced from an already secured business object:

When you introduce new operations or additional data records exposed by an existing ADF entity object into the extended application, you must not modify the predefined data security policies or data role templates that already exist for that business object. Instead, define a new data security policy to grant end users access to the operations or data records that had previously remained protected.

Note that the operations to be secured on the business object may also require enabling OPSS authorization checking for those operations on the ADF entity object in the data model project, as described in Section 8.3.7, "What You Can Customize in the Data Model Project at Design Time."

When already exposed operations or data records need to be accessible to additional end users:

When you introduce functionality into the extended application that changes the access requirements of the operations and data records exposed by an existing business object, then those end users may be provisioned by existing job roles or data roles. Consult the security administrator to make the data accessible to additional end users through role provisioning. This type of customization does not require modifying the access privileges or the duty roles of an associated data security policy.

8.3.5 Scenarios Related to Extending and Securing User Interface Artifacts

When you want to extend an Oracle Fusion application user interface to support particular end user duties, you may either create a new ADF bounded task flow or customize an existing bounded task flow. The bounded task flow specifies the control flow that the end user is expected to follow when interacting with the web pages contained by the task flow. Similarly, top-level web pages (ones that are *not* contained by a bounded task flow) may be introduced or customized.

Security concerned with controlling access to a bounded task flow or top-level web page is called **function security**. Function security policies involve granting an end user, by means of the end user's membership in a role, the ability to access task flows and perform operations in the contained web pages. For example, the accounts payable manager must be granted access privileges to the task flow that provides the functions to manage the invoice data records. If the manager is authorized to access the task flow, then a data security policy governing the invoice records will specify the manager's right to access the actual data.

Function security is implemented at the most fundamental level as resource/action pairs that may be granted to secure specific application artifacts. Oracle ADF defines the actions needed to secure certain Oracle ADF application artifacts, including ADF bounded task flows and, in the case of top-level web pages, ADF page definitions files.

In the Oracle Fusion Applications environment, function security policies aggregate one or more resource/action pairs into an entitlement definition. The entitlement is the entity that is granted to a duty role. The function security policy for the Oracle ADF application artifact, confers the end user with function access privileges, such as view or manage, through a specific duty role.

The function security policies for all the resources of the Oracle Fusion application form the function security repository, which is implemented as an OPSS application policy store. The OPSS policy store in a test or production environment is an LDAP server running Oracle Internet Directory. At runtime, OPSS performs authorization checks against the application policy store to determine the end user's access privileges.

> **Note:** For more information about how Oracle Platform Security Services implements function security, see the "Understanding Security Concepts" part in the *Oracle Fusion Middleware Application Security Guide*.

The security administrator for the enterprise exports the LDAP-based application policy store for a particular Oracle Fusion application into an XML file-based policy store that allows you to add security policies using the tools provided by JDeveloper. As an Oracle Fusion security guideline, you must define a new function security policy rather than modify the predefined function security policies of the security reference implementation. Additionally, a custom duty role must be created as the recipient (also called the **grantee**) of the new function security access privileges because granting privileges to a predefined duty role would alter the segregation of duties defined by the security reference implementation.

> **Note:** Developers are not entitled to modify the role hierarchy defined by the Oracle Fusion Applications security reference implementation. Therefore, whenever you create a new duty role, you must consult the security administrator to assign the custom duty role to a job role.

Whether or not you will need to define a function security policy to grant access to a task flow or top-level web page depends on the type of customization, as summarized in Table 8–1. The scenarios for defining function security policies include the following.

When a new task flow or top-level web page is introduced:

When you expose new functionality in the extended application through a new ADF bounded task flow or top-level web page that you create, then a new function security policy must be defined to grant end users access to the application artifact.

The new ADF bounded task flow and top-level web page are the only scenarios that require a new function security policy for the extended application.

When a new web page is introduced into an existing task flow:

When you modify an existing task flow to include new web pages, those web pages will be secured by the containing task flow's existing security policy. In this case, because all web pages contained by a bounded task flow are secured at the level of the task flow, there is no need to grant more function security privileges specifically for the new page. You will, however, need to define a new data security policy to grant end users access to any new data records that were introduced by the customization.

When a web page is modified to display a new field of sensitive data:

When you modify a web page to display sensitive data for a single data record field (for example, by adding a column to a table component to display salary information), access to the field displayed by the user interface component cannot be controlled by a function security policy. Authorization checking is not implemented by OPSS at the level of ADF Faces user interface components in the web page. Instead, you enter an Expression Language (EL) expression on that part of the databound ADF Faces component responsible for rendering the field and test the end user's associated role.

Note that using EL expressions to conditionally render a portion of a user interface component does not control access to the actual data; truly sensitive data must be secured on the business object with a data security policy, as described in Section 8.3.4, "Scenarios Related to Extending and Securing Data Model Components."

When a web page is modified to display UI components that must not be viewable by all end users:

When you modify a web page to display user interface components that not all end users need to view (for example, a button that deletes data records), access to the components cannot be controlled with a function security policy. Authorization checking is not implemented by OPSS at the level of ADF Faces user interface components in the web page. Instead, you enter an EL expression using ADF Security utility methods on the `rendered` property of the ADF Faces component to hide or render the entire component based the end user's associated role.

Note that using EL expressions to conditionally render a user interface component does not control access to the actual data (if that component displays data). Truly sensitive data must be secured on the business object with a data security policy, as described in Section 8.3.4, "Scenarios Related to Extending and Securing Data Model Components."

When existing task flows or top-level web pages must be accessible by additional end users:

When you introduce functionality into the extended application that changes the access requirements of an existing bounded task flow or top-level web page, then consult the security administrator to make the resource accessible to additional end users through role provisioning. This type of customization does not require changing the access privileges associated with the resource or the duties it defines.

8.3.6 What You Can Customize in the Data Security Policy Store at Design Time

Data security policies are stored in the Oracle Fusion Data Security repository and are defined and edited using Oracle Authorization Policy Manager. You have access to this tool through Oracle Fusion Functional Setup Manager, from the **Manage Data Security** task available in the Setup and Maintenance work area of any Oracle Fusion Setup application.

> **Note:** After you select the **Manage Data Security** task in Oracle Fusion Functional Setup Manager, the environment redirects to the data security customization user interface provided by Oracle Authorization Policy Manager. In this guide, although the data security customization tool is identified as Oracle Authorization Policy Manager, be aware that the tool must be accessed through Oracle Fusion Functional Setup Manager.

Data security policies control access to the database resources of an enterprise. Database resources in the security reference implementation include database tables and views and are predefined standard business objects that must not be changed. However, for cases where custom database resources must be secured business objects (defined by ADF entity objects in the data model project), you can be entitled to create custom duty roles, manage database resources, and define new data security policies using Oracle Authorization Policy Manager.

Important: As an Oracle Fusion Applications security guideline, the privileges granted by predefined data security policies assigned to duty roles of the Oracle Fusion Applications security reference implementation must not be changed by customization developers. Always define new data security policies to confer additional access privileges. Details about the security reference implementation can be found in the Oracle Fusion Applications security reference manuals.

The data security policy consists of privileges conditionally granted to a role to control access to instance sets of the business object. A privilege is a single action corresponding to an end user's intended operation on a single business object. A data security policy therefore is a grant of a set of privileges to a role on a business object for a given instance set. You can define the instance sets as a single row of data, multiple rows of a single table, or all rows of a single table.

The following security artifacts are recorded in the Oracle Fusion Data Security repository for a new data security policy:

- A database resource that references a primary key corresponding to the database table or view of the business object on which data security will be enforced.

  After the database resource is defined in the data security repository, Oracle Fusion Data Security protects the data records and operations exposed by the business object by default, and a data security policy must be defined to grant end users access to the business object.

- One or more roles that will be assigned to the end users who can perform the granted actions.

  For more details about the roles used by Oracle Fusion Applications, see Section 8.2, "About Extending the Oracle Fusion Applications Security Reference Implementation."

- A rule (also called a **named condition**) to define the available row instance sets in the form of a SQL predicate or simple filter (stored as XML) defined on the rows of the database resource.

  Instance sets may be a single row of data, multiple rows of a single table, or all rows of a single table. Only instance sets with multiple rows require creating a named condition.

- One or more actions (such as view, edit, and delete) performed on database records that correspond to the operations supported by the business object (which may include custom operations).

At runtime, data security policies make data available to end users based on their provisioned roles according to the following means:

- Action grants that specify whether the end user has the necessary privilege to perform the intended operation
- Condition evaluation for individual actions (and its corresponding operation) that specify which data records from the database resource may be accessed

> **Note:** The application developer does not enforce data security policies when creating the policies. In the case of data security, you must enable OPSS authorization checking on each business object that needs data security. This enforcement is implemented in JDeveloper, as described in Section 8.3.7, "What You Can Customize in the Data Model Project at Design Time."

Related to data security is an Oracle Fusion security feature called the data role template. Oracle Fusion Applications supplies **data role templates** to anticipate typical Oracle Fusion security scenarios and to allow the enterprise to generate data security policies based on information that is specific to the enterprise, such as the names of business units on which to apply the data security policies. Typically, the implementation manager for Oracle Fusion Applications enters the template information and then runs the templates to generate data security policies and the supporting data roles.

When you create a new business object or expose a new set of data records in the extended application, you must confirm whether a data role template exists to generate data security policies for that business object. If a data role template exists, you can update the template to supply information pertaining to the business object, such as the data records to secure and the data dimensions to express data stripes, such as territorial or geographic information used to partition enterprise data. A **data dimension** is a stripe of data accessed by a data role, such as the data controlled by a business unit.

Using Oracle Authorization Policy Manager, you may perform the following data security-related customization tasks:

- Manage database resources.

  An existing database resource must not have its primary key altered, but you can define new named conditions and add new actions to map any new operations that you implement. If you create a new business object for a database table or view, you can create an all new database resource with named conditions (see the next list entry) and actions.

- Create named conditions to filter the rows of the business object. (Optional)

  The database resource conditions are specified as SQL queries that, when added to a security policy, filter the data and generate an instance set of available data records. Conditions specify the entitlements available to the end user for specific business objects. Conditions may be static or they may be parameterized to allow instance sets to be specified generically but granted specifically. Note a condition is required only when the data security policy does not secure either a single data record or all data records: Both of these cases may be defined without named conditions when creating the security policy.

  Note that instance sets generated with parameters cannot be used for data security that is enforced declaratively. Instead, you must write code to enforce OPSS authorization checking.

- Define data security policies consisting of privileges for a specific application role, named condition (optional), and business object.

  A privilege can map a standard action to a standard operation: read, update, and delete on a condition of a business object. The standard actions and the standard operations are named similarly.

Alternatively, a privilege can map a custom action to a custom operation on a condition of a business object. The custom privilege, for example `ApprovePO`, is useful to secure a custom operation in the data model project or to secure any operation for row sets at the level of the individual ADF view object. The custom privilege also supports securing operations on columns through an EL expression in the user interface project or Groovy scripting language expressions in the data model project.

As an alternative to specifying a named condition, the data security policy can secure an instance set defined by a single data record or defined by all data records. Both of these cases may be selected when creating the data security policy.

- Generate data security policies by updating a data role template with data dimensions and data sets required to support the business object.

  A data role template generates data security policies for a business object based on supplied data dimensions to partition the data records into sets of data security policies. The template also maps instance sets for the data security policies it will generate to a particular data dimension. Instance sets are authored at the time the business object is registered as a database resource. Data dimensions and instance sets are specified as SQL clauses.

  Note that the SQL clauses cannot be modified after running the template.

For an overview of these tasks, see Section 8.4, "Defining Data Security Policies on Custom Business Objects." For detailed documentation, see the "Managing Oracle Fusion Applications Data Security Policies" and "Oracle Fusion Applications Data Role Templates" chapters in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

8.3.7 What You Can Customize in the Data Model Project at Design Time

You create a data model project in JDeveloper to map custom business objects to ADF entity objects. At runtime, the ADF entity object creates a row set of data records exposed by the business object and simplifies modifying the data by handling create, read, update, and delete operations. In the data model project, you then define one or more ADF view objects on top of the ADF entity object to shape the data to the row set required by the tasks of the application, such as populating a form that displays a customer's sales invoice.

After you map the business object to an ADF entity object, enforcement of data security policies does not occur automatically on the data records of the exposed business object. The Oracle Fusion security approach protects the business object that has been registered as an Oracle Fusion Data Security database resource to ensure that end users do not have unintended access to sensitive data. In adherence to the security principle of protected by default, Oracle Fusion security separates defining policies and enforcing policies. Thus, by default, data security policies for a business object will remain ineffective until you enable OPSS authorization checking on the operations of the ADF business component. Enforcement of OPSS authorization checking can be specified either declaratively, at the level of ADF entity objects or ADF view objects, or programmatically, on any related code.

You can modify the data model project to opt into data security in two ways:

- At the level of the ADF entity object, to enable OPSS authorization checking on standard operations. Standard operations supported by ADF entity objects include, read, update, and delete current row. In this case, all ADF view objects based on the ADF entity object will have the same level of authorization checking

enabled. The applicable data security policies will filter the data for each row set produced by these ADF view objects in exactly the same way.

- At the level of the ADF view object, to enable OPSS authorization checking on standard operations for a collection of rows. This provides a way to filter the data in the data model project based on an individual row set that the ADF view object defines. This level of authorization checking also supports defining a custom privilege (corresponding to the ADF view object read operation) in the data security policy store.

Using JDeveloper, you can perform the following security-related customization tasks in the data model project:

- Enforce row-level security for standard operations.

  Standard operations that you can secure are read, update, and remove current row. OPSS authorization checking is enabled directly on the ADF entity object to be secured. Although the ADF entity object maps to all instances of the business object, the data security policy defines conditions to filter the rows displayed to the end user.

- Enforce row-level security for custom operations.

  You may wish to enforce security for custom operations that are specific to the custom business object. Custom operations are not supported by ADF Business Components on the ADF entity object. When a data security policy defines a custom operation, you must enable it using view criteria that you set on an ADF view object. The view criteria identifies the data security policy and business object.

- Enforce security for individual attributes of business objects.

  Column-level OPSS authorization checking is not supported on the attributes of ADF entity objects or ADF view objects. You must create a custom OPSS permission for the column-level read or update operation and then map that to a custom privilege. Whether or not the user interface displays the column is specified by testing that custom privilege in the user interface using an EL expression on the secured attribute displayed by the user interface component.

For an overview of these tasks, see Section 8.5, "Enforcing Data Security in the Data Model Project." For detailed documentation, see the "Implementing Oracle Fusion Data Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

8.3.8 What You Can Customize in the User Interface Project at Design Time

Before you define function security policies, you will use JDeveloper to create a user interface project with the custom ADF bounded task flows or top-level web pages that you intend to secure.

To simplify the task of securing the functions of the extended application, ADF Security defines a containment hierarchy that lets you define a single security policy for the ADF bounded task flow and its contained web pages. In other words, the security policy defined at the level of the bounded task flow, secures the flow's entry point and then all pages within that flow are secured by the same policy. For example, a series of web pages may guide new end users through a registration process and the bounded task flow controls page navigation for the process.

Specifically, the Oracle ADF application artifacts that you can secure in the user interface project of the extended Oracle Fusion application are:

- An ADF bounded task flow that protects the entry point to the task flow, which in turn controls the end user's access to the pages contained by the flow

  The ADF unbounded task flow is not a securable application artifact and thus does not participate in OPSS authorization checking. When you must secure the contained pages of an unbounded task flow, you define policies for the page definition files associated with the pages instead.

- ADF page definition files associated with top-level web pages

  For example, a page may display a summary of products with data coordinated by the ADF bindings of the page's associated ADF page definition file.

For details about creating bounded task flows and databound top-level web pages, see the "Introduction to Building Fusion Web Applications with Oracle ADF" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Although you can define function security policies for the custom resources of the user interface project, enforcement of function security does not occur automatically. The Oracle Fusion security approach protects securable Oracle ADF application resources to ensure that end users do not have unintended access. In adherence to the security principle of protected by default, Oracle Fusion security separates defining policies and enforcing policies. Thus, by default, function security policies will remain ineffective until you enable OPSS authorization checking by running the ADF Security wizard in JDeveloper on the user interface project.

Important: After you run the ADF Security wizard, OPSS authorization checking is enforced on the bounded task flows and top-level pages. These application artifacts will be inaccessible when testing the application in JDeveloper. To enable access, you must define function security policies on the protected application artifacts, as described in Section 8.3.9, "What You Can Customize in the Application Security Policy Store at Design Time."

Using JDeveloper, you can perform the following security-related customization tasks in the user interface project:

- Enable OPSS authorization checking to protect Oracle ADF application artifacts.

  Oracle ADF application artifacts in the user interface project, including ADF bounded task flows and the top-level web pages (with a backing ADF page definition) will be protected when you configure ADF Security by running the ADF Security wizard with the **Authentication and Authorization** option selected. This ensures that end users do not have unintended access to sensitive task flows of the extended application.

- Conditionally display or hide user interface components in the web page.

  ADF Security implements utility methods for use in EL expressions to access Oracle ADF application artifacts in the security context. For example, you can use the ADF Security utility methods to specify whether the end user is allowed to access create, edit, or delete buttons. Good security practice dictates that your application must hide user interface components and capabilities for which the end user does not have access. For example, if the end user is not allowed access to a particular task flow, you can use the EL expression to evaluate the role membership of the end user to determine whether or not to render the navigation components that initiate the task flow.

For an overview of these tasks, see Section 8.6, "Defining Function Security Policies for the User Interface Project." For detailed documentation, see the "Implementing Function Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

8.3.9 What You Can Customize in the Application Security Policy Store at Design Time

You can use JDeveloper to add application security policies to a file-based policy store that the security administrator creates by exporting policies from the LDAP-based application security policy store. The file containing the exported policy store is the `jazn-data.xml` file.

As a security development guideline, use JDeveloper tools only to work on the exported file-based policy store, and do not edit the security definitions directly. JDeveloper supports iterative development of security so you can easily define, test, and edit security policies that you define for Oracle ADF application artifacts. In JDeveloper, you can also create end user identities for the purpose of running and testing the application in JDeveloper's Integrated WebLogic Server. You provision a few end user test identities with roles to simulate how the actual end users will access the secured application artifacts.

After testing in JDeveloper using Integrated WebLogic Server, you must consult with the security administrator to merge the LDAP-based application policy store in the staging environment with the security policies that you added to the exported XML file. Initially, the staging environment allows further testing using that server's identity store before deploying to the production environment. Thus, end user identities created in JDeveloper are not migrated to standalone Oracle WebLogic Server and are used only in Integrated WebLogic Server to test the extended application.

Note: For details about implementing and testing security using JDeveloper, see the "Implementing Function Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

The basic security artifact for function security is the JAAS (Java Authentication and Authorization Service) **permission**, where each permission is specific to a resource type and maps the resource with an allowed action. In general, the JAAS permission specifies the allowed operations that the end user may perform on a particular application artifact. However, from the standpoint of Oracle Fusion Applications, end users typically need to interact with multiple resources to complete the duties designated by their provisioned roles. To simplify the task of creating function security policies in the Oracle Fusion Applications environment, you work with OPSS entitlements to grant privileges to a role for a variety of securable resources, including ADF task flows, web services, and service-oriented architecture (SOA) workflows.

Function security policies that comprise entitlement grants with multiple application artifacts are called **entitlement-based policies**. Example 8–1 shows the Oracle Fusion Applications entitlement policy Maintain Purchase Orders, which groups the OPSS permissions for ADF task flows, a web service, and a SOA workflow.

Example 8–1 OPSS Entitlement-Based Policy Groups Permissions as a Set that May Be Granted to a Role

```
Resource Type: ADF Taskflow
Resource: PO Summary
Action: view

Resource Type: ADF Taskflow
```

```
Resource: PO Details
Action: view

Resource Type: ADF Taskflow
Resource: Supplier Details
Action: view

Resource Type: Web Service
Resource: SpendingLimitCheckWS
Action: invoke

Resource Type: Workflow
Resource: POApproval
Action: submit
```

You use the security policy editor in JDeveloper to define the entitlement-based policy. JDeveloper modifies the source in the exported XML file. As Example 8–2 shows, entitlement-based policies in Oracle Fusion applications are defined in the `<jazn-policies>` element. The policy store section of the file contains the following definitions:

- A `<resource-type>` definition that identifies the actions supported for resources of the selected type
- A `<resource>` definition to identify the resource instance that you selected from your application and mapped to a resource type
- A `<permission-set>` definition to define the resources and actions to be granted as an entitlement
- A `<grant>` definition with one or more entitlements (defined in the XML as a `<permission-set>`) and granted to the desired application roles (the grantee)

Example 8–2 Entitlement-Based Security Policy Definition in jazn-data.xml File

```
<?xml version="1.0" ?>
<jazn-data>
  <policy-store>
    <applications>
      <application>
        <name>MyApp</name>

      <app-roles>
        <app-role>
          <name>AppRole</name>
          <display-name>AppRole display name</display-name>
          <description>AppRole description</description>
          <guid>F5494E409CFB11DEBFEBC11296284F58</guid>
          <class>oracle.security.jps.service.policystore.ApplicationRole</class>
        </app-role>
      </app-roles>

      <role-categories>
        <role-category>
          <name>MyAppRoleCategory</name>
          <display-name>MyAppRoleCategory display name</display-name>
          <description>MyAppRoleCategory description</description>
        </role-category>
      </role-categories>

      <!-- resource-specific OPSS permission class definition -->
```

```
<resource-types>
  <resource-type>
    <name>APredefinedResourceType</name>
    <display-name>APredefinedResourceType display name</display-name>
    <description>APredefinedResourceType description</description>
    <provider-name>APredefinedResourceType provider</provider-name>
    <matcher-class>oracle.security.jps.ResourcePermission</matcher-class>
    <actions-delimiter>,</actions-delimiter>
    <actions>write,read</actions>
  </resource-type>
</resource-types>

<resources>
  <resource>
    <name>MyResource</name>
    <display-name>MyResource display name</display-name>
    <description>MyResource description</description>
    <type-name-ref>APredefinedResourceType</type-name-ref>
  </resource>
</resources>

<!-- entitlement definition -->
<permission-sets>
  <permission-set>
    <name>MyEntitlement</name>
    <display-name>MyEntitlement display name</display-name>
    <description>MyEntitlement description</description>
    <member-resources>
      <member-resource>
        <type-name-ref>APredefinedResourceType</type-name-ref>
        <resource-name>MyResource</resource-name>
        <actions>write</actions>
      </member-resource>
    </member-resources>
  </permission-set>
</permission-sets>

<!-- Oracle function security policies -->
<jazn-policy>
  <!-- function security policy is a grantee and permission set -->
  <grant>
    <!-- application role is the recipient of the privileges -->
    <grantee>
      <principals>
        <principal>
          <class>
             oracle.security.jps.service.policystore.ApplicationRole
          </class>
          <name>AppRole</name>
          <guid>F5494E409CFB11DEBFEBC11296284F58</guid>
        </principal>
      </principals>
    </grantee>
    <!-- entitlement granted to an application role -->
    <permission-set-refs>
      <permission-set-ref>
        <name>MyEntitlement</name>
      </permission-set-ref>
    </permission-set-refs>
  </grant>
```

```
      </jazn-policy>
    </application>
  </applications>
 </policy-store>
</jazn-data>
```

While OPSS permissions granted for a single resource are not typically defined in the Oracle Fusion Applications environment, function security policies that use OPSS permissions for a single resource are called **resource-based policies**. Ultimately, a function security policy may have either one or more OPSS permissions, one or more OPSS permission sets (entitlements), but not both.

> **Note:** Granting access to web pages in Oracle Fusion Applications is enforced at the level of ADF Controller components called **bounded task flows**. Task flows in Oracle Fusion Applications are ADF Controller components that assemble the application's web pages (or regions within a web page) into a workflow that supports the tasks to be performed by application end users. Defining security policies on task flows instead of individual web pages is a security best practice that blocks end users from directly accessing the pages of a task flow. Web pages that are not contained in a task flow are top-level pages and may have security policies defined individually.

Provisioning end users with role membership is defined in the application's identity store and is a configuration task to be performed by the security administrator, independent of security customization.

Using JDeveloper, you may perform the following function security customization tasks:

- Define an entitlement-based policy for all other application roles.

  An entitlement-based policy is a set of resource grants (set of OPSS permissions) that will be required by the end user to complete a task.

- Define a resource-based policy specifically for the built-in OPSS application role `authenticated-role`.

  A resource-based policy sets an OPSS permission on a single application resource and grants that permission to an application role. This type of function security is typically not used by securable resources in Oracle Fusion Applications. However, the resource-based policy must be used to make a custom resource accessible to any authenticated end user (ones who visit the site and log in). For example, granting a view privilege to the built-in OPSS application role `authenticated-role` is the way to make an employee registration task flow accessible to all employees within the enterprise.

For an overview of these tasks, see Section 8.6, "Defining Function Security Policies for the User Interface Project." For detailed documentation, see the "Implementing Function Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

8.3.10 What You Cannot Do with Security Policies at Design Time

After you define the security policies, consult a security administrator to migrate the policies to the staging environment.

The security administrator is responsible for the following tasks.

- After testing is completed in JDeveloper, the security administrator must merge the file-based policy store with the application policy store in the staging environment.

  For information about how the security administrator merges the policies using Oracle Authorization Policy Manager, see the "Upgrading Oracle Fusion Applications Policies" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

- The security administrator must provision enterprise users by mapping enterprise roles (defined in the staging environment identity store) to the custom application roles.

  For information about how the security administrator provisions enterprise users using Oracle Authorization Policy Manager, see the "Managing Policies and Policy Objects" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

- Before running the application in the staging environment, the security administrator must reconcile the application role GUID of any data security policies that were created based on new custom application roles.

  When the file-based policy store is merged, the GUIDs of application roles are not preserved. For information about how the security administrator reconciles GUIDs in a staging environment, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

- Before running the application in the staging environment, the security administrator must modify the application to use the LDAP-based policy store provided by the testing environment.

  For more information, see the "Implementing Function Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

- After testing is completed in the staging environment, the security administrator can migrate the application policy store from the staging environment to the policy store in production.

  For information about how the security administrator migrates policies to a new environment, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

8.3.11 Before You Begin Customizing Security

Before you begin customizing security, you should be familiar with the Oracle Fusion application architecture that enables customization, as described in Chapter 1, "Customizing and Extending Oracle Fusion Applications." You should also understand the typical workflows for working with customizations, as described in Chapter 2, "Understanding the Customization Development Lifecycle."

You will need to do the following before you can begin customizing security:

1. Install JDeveloper and set up your development environment.

   For more information, see Section 1.3.13, "Installing Customization Tools."

2. Create a customization application workspace.

   Before you can implement customizations using JDeveloper, you must create an application workspace that imports the necessary parts of the application you want to customize. For more information, see Chapter 3, "Using Oracle JDeveloper for Customizations."

3. Start JDeveloper in the appropriate role.

   If you are implementing customizations on existing application artifacts, you must select the **Oracle Fusion Applications Administrator Customization** role when you start JDeveloper.

   If you are creating new custom application artifacts (such as, entity objects, view objects, and pages), you must select the **Oracle Fusion Applications Developer** role when you start JDeveloper.

4. Create the database resources in a custom schema for Oracle Fusion Applications.

   The database table exposes the data in your application. You are free to use any tool you wish to create database objects in your custom schema. For example, you may choose to work with the Database Navigator in JDeveloper to model database objects. For information about creating the table in a custom schema, see Section 4.8, "Customizing and Extending the Oracle Fusion Applications Schemas." For information about creating database objects, see the Designing Databases topics in the JDeveloper online help.

5. When securing confidential personally identifiable information (PII), create a new table in a custom schema for Oracle Fusion Applications, a view corresponding to the new table, and a VPD policy to associate a PL/SQL filter function with the view.

   The VPD policy filters the view to expose the data for which data security policies may be defined. For information about creating the table in a custom schema, see Section 4.8, "Customizing and Extending the Oracle Fusion Applications Schemas." For information about creating VPD policies, see the "Using Oracle Virtual Private Database to Control Data Access" chapter in the *Oracle Database Security Guide*.

6. Obtain privileges to define or edit Oracle Fusion Data Security security policies.

   If you will be creating or editing Oracle Fusion Data Security security policies in Oracle Fusion Applications, you will need specific privileges. When you have the necessary privileges, Oracle Authorization Policy Manager allows you to access the data security customization user interface. Contact your security administrator for details.

7. In Oracle Authorization Policy Manager, create custom application roles.

   Data security and function security permit granting access privileges to Oracle Fusion Applications duty roles (also called OPSS application roles). Although Oracle Fusion Applications ships with standard duty roles, as an Oracle Fusion security guideline, you must create new duty roles rather than grant privileges to predefined duty roles.

   For information about creating application roles, see the "Managing Policies and Policy Objects" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

8. In human capital management (HCM) core applications, create job roles, as needed.

   Job roles (also called OPSS enterprise roles) provide access to application resources through the Oracle Fusion Applications role inheritance hierarchy, which specifies the inherited duty roles. Although Oracle Fusion Applications ships with standard job roles, the security administrator can create a new job role even when one does already exist that defines the new duties.

The security administrator uses integrated Oracle Identity Management pages to create and manage job roles in Oracle Fusion Applications. For information about creating job roles, see the "Managing Roles" chapter in the *Oracle Fusion Middleware User's Guide for Oracle Identity Manager*.

9. Identify the ADF business components in your application's data model project that you want to create or customize.

   You can create or customize ADF entity objects and ADF view objects using JDeveloper to expose business objects in your application and opt into data security policies. For information about creating these ADF business components, see Chapter 4, "Customizing and Extending Oracle ADF Application Artifacts."

10. Identify the application artifacts in your user interface project that you want to create or customize.

    The following application artifacts that you create or customize using JDeveloper may be secured: ADF bounded task flows and ADF page definition files for top-level web pages. For information about creating these application artifacts, see Chapter 4, "Customizing and Extending Oracle ADF Application Artifacts."

11. In JDeveloper, run the ADF Security wizard on your application.

    When you run the ADF Security wizard, it configures your application to enable authorization checking so that Oracle Platform Security Services (OPSS) running in Oracle WebLogic Server (and in JDeveloper's test environment, Integrated WebLogic Server) will utilize the security policies to authorize access to application resources by the end user. OPSS determines whether the end user (represented by the JAAS subject) has the privileges necessary to access the resources they intend.

    For information about running the wizard, see the "Implementing Function Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

8.4 Defining Data Security Policies on Custom Business Objects

In Oracle Authorization Policy Manager, the general process for defining a data security policy is as follows:

1. Register the custom business object as a database resource.
2. Define the instance set of data records that you want to associate with specific securable operations of the ADF business component.

   The security policy identifies named conditions from the security repository to specify the row instance set available to the end user provisioned to the role with the privilege to perform the intended ADF business component operation.

   In Oracle Authorization Policy Manager, a condition you create defines an instance set of multiple rows specified either by simple filters (XML-defined) or complex SQL queries whose values can be parameterized. No condition definition is needed in the case of a single row instance or all the row instances of the database resource.

3. Define the list of actions that you want to be able to grant to the role.

   Action are database equivalent create, read, update, delete (CRUD) operations and correspond to the names of securable operations of the business object that the end user may invoke. The data security policy you define will associate one or more actions with an instance set.

4. If the custom business object is not supported by a data role template, define the data security policy:

   a. Enter a name and start date for the data security policy.

   b. Select one or more job roles or duty roles to which the policy grants access. The roles you select entitle all end users assigned to those roles with access to the data.

      In Oracle Authorization Policy Manager, duty role names that you enter are identified as OPSS internal roles called application roles. Similarly, job role names are identified as OPSS external roles called enterprise roles.

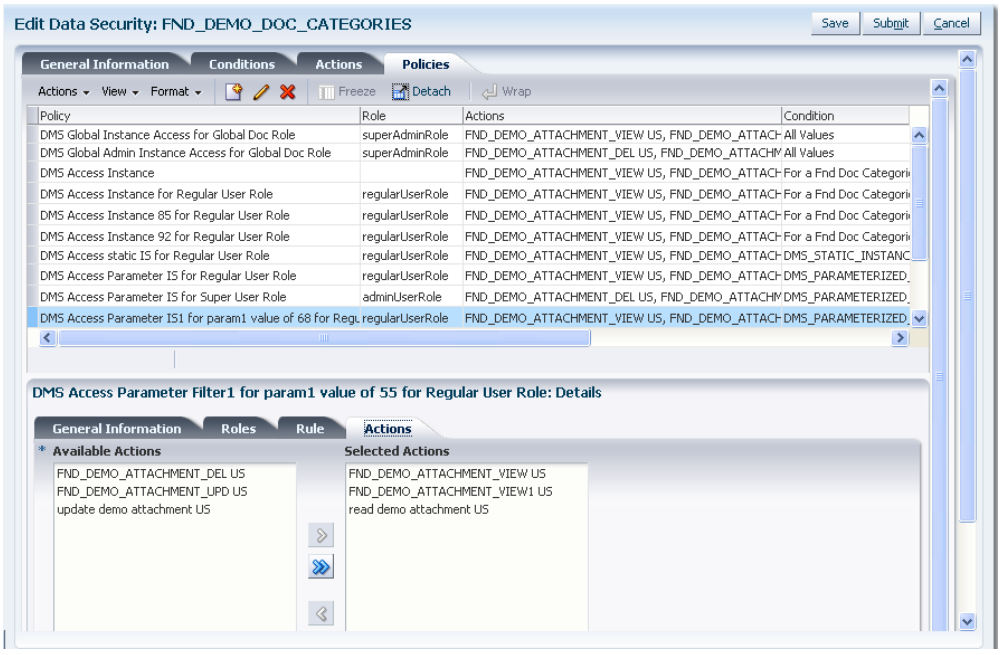
   c. Specify an instance set on the database resource for which the security policy will control access. This may be a single row, all rows, or multiple rows (specified by a previously defined named condition).

   d. Specify one or more actions to secure on the database resource for the currently specified instance set.

   e. Repeat the steps to grant actions access to additional instance sets for the current data security policy and roles.

      Figure 8–1 illustrates the **Actions** tab in the Edit Data Security page after several actions have been selected. Available actions will be limited to the actions that had been defined for the database resource.

Figure 8–1 Creating a Data Security Policy - Selecting Actions



5. If the custom business object is supported by a data role template, then update the data role template with the following information:

   a. When the job role grantees of the data security policies generated by the template are not already defined by the existing data role template, add a new external role.

      The data role template specifies which base roles to combine with which dimension values for a set of data security policies.

b. When the custom business object expresses a new data stripe to apply to the generated data security policies, modify the SQL code that identifies the dimension values of the template.

Note that the SQL code cannot be modified after running the template.

c. When the data role grantee of the data security policies generated by the template are not already defined by the existing data role template, configure a new data role name.

The data role template constrains the data roles with access privileges for specific data records with particular actions. The data role provides provisioned end users with privileges to access a dimensional subset of the data granted by a data security policy.

d. Select the database resource that you registered for the custom business object.

e. Optionally, select one or more data sets that you specified as named conditions when you created the database resource.

Alternatively, the template can generate policies based on the primary key of the database resource.

f. Specify one or more actions to secure on the database resource for the currently specified instance set.

Before you begin:

If you will be creating or editing Oracle Fusion Data Security security policies in Oracle Fusion Applications, you will need specific privileges. When you have the necessary privileges, Oracle Authorization Policy Manager allows you to access the data security customization user interface. Contact your security administrator for details.

The security reference implementation defines the job role `IT Security Manager` with a duty role hierarchy that includes the `Application Data Security Administration Duty` duty role. This duty role is entitled to manage data security policies (the entitlement is `Manage Data Security Policy`) and provides the access necessary to perform the **Manage Data Security Policies** task in Oracle Authorization Policy Manager. Contact your security administrator for details.

Additionally, collect the following information that you will use to define the data security policy in Oracle Authorization Policy Manager:

- The primary key of the database table or view that the custom business object represents

  You specified the primary key of the database table or view when you registered the database resource.

- The names of the conditions for which you want the security policy to control access

  When you registered the database resource, you may have created named conditions to control access to instance sets composed of multiple rows (Oracle Fusion Data Security does not require that you create a named condition when you want to grant access to instance sets composed either of a single row or of all rows of the database resource).

- The names of the actions for which you want to associate with a particular named condition (or instance set) to control access

When you registered the database resource, you named actions to identify the securable operations of the custom business object. Action names must be identical to the names of the operations the business object supports. For example, the names of actions corresponding to the supported standard operations are view, edit, and delete. However, if your data model project defines custom operations, actions may have names corresponding to operations named, for example, as `view_US_ONLY`, `edit_US_ONLY`, or `delete_US_ONLY` .

- The names of the custom duty roles for which you want to grant access to the conditions and actions of the database resource associated with the custom business object

  As an Oracle Fusion Applications security guideline, predefined duty roles defined by the security reference implementation must not be modified. You must use Oracle Authorization Policy Manager to create a new duty role rather than grant data security privileges to predefined duty roles. For information about creating roles, see the "Managing Policies and Policy Objects" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

Task: Create Conditions on a Business Object

A business object can define securable instance sets of data records as named conditions. The data security policy you define may identify a specific data record, all data records of the object, or multiple data records. When you want to secure specific sets of records, then conditions must be created on the business object.

To create conditions for a business object:

1. From the **Administration** menu in the global area of Oracle Fusion Applications, choose **Setup and Maintenance**, and then choose **Manage Data Security Policies**.
2. In the **General Information** tab, register the business object as a database resource, and then click the **Conditions** tab and click **New**.
3. In the Create Database Resource Condition dialog, enter the SQL predicate consisting of a query on the table named by the database resource.

For more information, see the "Managing Oracle Fusion Applications Data Security Policies" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

Task: Grant Access for a Privilege to a Specific Role and Object Condition

A business object can define conditions that query only the set of data records that are relevant to the members of a particular enterprise role or application role (also called job roles or duty roles, respectively). You can secure these sets of data records by making grants on conditions of the business object for a particular application role and privilege that you define. Condition-level security lets you secure any number of subsets of the business instances defined by the business object. As an alternative to standard privileges, you can define a custom privilege to define a security policy for operations that may be specific to a particular group of end users. Custom privileges also let you enforce security in the data model project at the level of the ADF view object and perform authorization checking to secure individual business object attributes.

To define a data security policy:

1. From the **Administration** menu in the global area of Oracle Fusion Applications, choose **Setup and Maintenance**, and then choose **Manage Data Security Policies**.

2. Register the business object as a database resource (using the **General Information**, **Conditions**, and **Actions** tabs sequentially), and then click the **Policies** tab and click **New**.
3. Use the policy workflow at the bottom of the Edit Data Security page (**Roles**, **Rule**, and **Action** tabs sequentially) to define the data security policy.

For more information, see the "Managing Oracle Fusion Applications Data Security Policies" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

Task: Grant Access to a Specific Data Role and Dimension Values

A business object can be mapped to a set of dimension values and data role naming rules defined by data role templates. A data role for a defined set of data describes the job an end user does within that defined set of data. A data role inherits job or abstract roles and grants entitlement to access data within a specific dimension of data based on data security policies. The dimension expresses data stripes, such as territorial or geographic information you use to partition enterprise data. You use data role templates to generate data roles and the template applies the values of the dimension and participant data security policies to the group of base roles.

To create or revise a data role template:

1. From the **Administration** menu in the global area of Oracle Fusion Applications, choose **Setup and Maintenance**, and **Manage Data Role Templates**.
2. In the data role template workflow, use the tabbed pages (**External Role**, **Dimension**, **Naming**, and **Policies** tabs sequentially) to create a data role template or revise an existing one.

For more information, see the "Oracle Fusion Applications Data Role Templates" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

8.5 Enforcing Data Security in the Data Model Project

Data security policies secure data from business objects based on the grants made to roles. The business object participating in data security defines a database resource (a database table or view) that has been registered in the Oracle Fusion Applications FND_OBJECTS table. When you need to expose data records in the extended application, you can use JDeveloper and Oracle ADF to create a data model project with ADF entity objects based on secured database resources. However, it is not sufficient to register the business object in FND_OBJECTS and define data security policies. Additionally, you must opt into those data security policies by enabling row-level OPSS authorization checking for specific operations on ADF entity objects in the data model project.

By default, after the database table or view backing the ADF entity object has been registered as a database resource in the FND_OBJECTS table, Oracle Fusion Data Security denies end users access to the business object data. Enabling OPSS authorization checking for the operations (such as view, edit, delete) by setting metadata on the ADF entity object of the data model project, ensures that only end users with sufficient privileges are authorized to perform the actions on the database resources corresponding to the ADF entity object.

JDeveloper saves the security metadata that you define on the data model project into an Oracle Metadata Services (MDS) repository.

Before you begin:

If the ADF entity object does not appear in the data model project, then you cannot opt into data security policies that may exist for the business object. You must use JDeveloper to create the ADF entity object based on a database table or database view that you intend to register in the Oracle Fusion Data Security schema. For more information, see Chapter 4, "Customizing and Extending Oracle ADF Application Artifacts."

For OPSS to enforce security, the database table or view backing the ADF entity object must be registered as a business object with the FND_OBJECTS table provisioned by Oracle Fusion Data Security (the registered business object is also called a database resource of the Oracle Fusion Data Security schema). You must use Oracle Authorization Policy Manager to register the custom business object corresponding to the ADF entity object data source. For more information, see the "Managing Oracle Fusion Applications Data Security Policies" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

Enabling security for custom operations in the data model project requires a custom privilege in the data security policy defined on the business object. You must create the custom privilege in the data security repository. For more information, see Section 8.4, "Defining Data Security Policies on Custom Business Objects."

Task: Enforce Row Security for the Standard Operations of a Business Object

The ADF entity object in a data model project defines metadata that enables OPSS authorization checking against data security policies for view, update, or delete operations (also called standard operations) of the registered business object. You enable row-level security for standard operations by selecting the operation on the ADF entity object that maps to the business object upon which data security policies exist. Although the ADF entity object maps to all instances of the business object, the data security policy defines business object conditions to filter the records available to the end user. Filtering of the business object for standard operations supports only row-level security.

To enforce authorization checking for standard operations:

1. In JDeveloper, display the ADF entity object in the overview editor.
2. In the editor, click the **General** navigation tab and expand the Security section, and then select the list of standard operations for which you want to enforce authorization checking against data security policies.

For more information, see the "Implementing Oracle Fusion Data Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Enforce Row Security for a Custom Operation of a Business Object

The ADF entity object in a data model project does not support OPSS authorization checking against data security policies for custom operations of the registered business object. You enable row-level security for custom operations by mapping view criteria that you create in the data model project to custom privileges in the data security policies defined on the business objects. The view criteria creates a row set filter by naming the custom privilege and business object. Filtering of the business object by view criteria works only with custom operations.

To enforce authorization checking for a custom operation:

1. In JDeveloper, display the ADF view object in the overview editor and, in the editor, click the **Query** navigation tab.

2. Expand the View Criteria section and then you click the **Add** button to create a view criteria to enforce authorization checking for a custom operation.

For more information, see the "Implementing Oracle Fusion Data Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Enforce Security for Attributes of a Business Object

The ADF entity object in a data model project does not support authorization checks against data security policies for columns of the registered business object. You enable security for attributes by creating a custom OPSS permission to control access to the column read or update operation, and then, in the Oracle Fusion Data Security repository, you map the operation to a custom privilege and grant the privilege to specify the roles that are authorized to view or update the data records. Last, in the user interface, you enter an EL expression to test that custom privilege on the user interface component displaying the attribute.

For more information, see the "Implementing Oracle Fusion Data Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

8.6 Defining Function Security Policies for the User Interface Project

You can use JDeveloper to define function security policies directly in an exported version of the Oracle Fusion application security repository. The security administrator exports the policies that exist in the LDAP-based application security policy store (residing in a test environment) into an XML file that can be loaded in JDeveloper and edited using the provided security policy editor.

After editing the XML file, you must consult the security administrator to merge the security policies into the test environment.

In JDeveloper, the general process for defining function security policies is as follows:

1. Consult a security administrator to export all predefined function security policies of the application that you are customizing into a `jazn-data.xml` file.

   For details about how the security administrator exports the application policy store, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

2. Copy the exported `jazn-data.xml` file into your application workspace.

   This is the file that JDeveloper will update when you define function security policies. For JDeveloper to use the file, copy the file to your application workspace in the `<JDevAppHome>/src/META-INF` folder.

3. Create an entitlement to group one or more custom resources and their corresponding actions that together entitle end users to access the resource when needed to complete a specific duty.

   In the Oracle Fusion Applications environment, the basic security artifact for entitlement-based security polices is the entitlement (an entitlement is defined as a OPSS permission set).

4. Grant the entitlement to a custom duty role that was added to the Oracle Fusion application policy store.

   The entitlement grant to the role specifies that the end user must be a member of the role to access the resources specified by the entitlement. You must use custom duty roles and you must not grant entitlements to predefined duty roles.

In JDeveloper, duty role names that you select are identified as OPSS internal roles called application roles.

5. Enable ADF Security for the application by running the Configure ADF Security wizard.

   The wizard configures files that integrate ADF Security with OPSS on Integrated WebLogic Server.

   After you run the ADF Security wizard, any web page associated with an ADF bounded task flow will be protected. Therefore *before* you can run the application and test security, you must define the security policies that grant end users access.

Before you begin:

Consult the security administrator to obtain the file-based application policy store in the form of a `jazn-data.xml` file. The security administrator can run an Oracle WebLogic Scripting Tool (WLST) script to export the LDAP-based application policy store to the XML file. For more information about how the security administrator exports the application policy store, see the "Securing Oracle Fusion Applications" chapter in the *Oracle Fusion Applications Administrator's Guide*.

If the custom bounded task flows or top-level web pages do not appear in the user interface project of the extended application, then you cannot define application security policies. You must use JDeveloper to create the securable Oracle ADF application artifacts. For more information, see Chapter 4, "Customizing and Extending Oracle ADF Application Artifacts."

As an Oracle Fusion Applications security guideline, predefined duty roles defined by the security reference implementation must not be modified. You must use Oracle Authorization Policy Manager to create a new duty role rather than grant function security privileges to predefined duty roles. For information about creating duty roles, see the "Managing Policies and Policy Objects" chapter in the *Oracle Fusion Middleware Oracle Authorization Policy Manager Administrator's Guide (Oracle Fusion Applications Edition)*.

Task: Create Entitlement Grants for a Specific Application Role

An entitlement grant is a set of resource grants (set of OPSS permissions) that will be required by the end user to complete a task. Each permission in the entitlement grant names an OPSS permission class, a resource, and an action. Entitlements must be granted to custom application roles.

To grant end users access to enable them to perform tasks:

1. In JDeveloper, choose **Application** then **Security** and then **Entitlement Grants**.
2. In the overview editor for security, name the entitlement, add member resources, and add the actions that you want to secure.
3. Grant the entitlement to a custom application role.

For more information, see the "Implementing Function Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Create Resource Grants for the Authenticated User Role

A resource grant sets an OPSS permission on a single application resource and grants that permission to an application role.

To make a resource publicly accessible by end users:

1. In JDeveloper, to make the resource publicly accessible, choose **Application** then **Security** and then **Resource Grants**.
2. In the overview editor for security, select the Oracle ADF artifact, the built-in OPSS role **authenticated-role** (or **anonymous-role**) as the grantee, and the action that you want to make public.

For more information, see the "Implementing Function Security" chapter in the *Oracle Fusion Applications Developer's Guide*.

Task: Display or Hide User Interface Components in a Web Page

The `rendered` attribute of a user interface component controls whether or not the component is visible in the web page. You can create an EL expression using ADF Security utility methods to conditionally render the UI component based on the end user's membership in a particular role.

To hide components in a web page from unauthorized end users:

1. In JDeveloper, open the web page and, in the Property Inspector, select **Expression Builder** for the **Rendered** property of the UI component that you want to conditionally render.
2. In the Expression Builder, expand **ADF Bindings - securityContext** and then select the appropriate EL method followed by the qualified name of the ADF resource that the user will attempt to access.

For more information, see the "Enabling ADF Security in a Fusion Web Application" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

9 Translating Custom Text

This chapter describes how to localize the changes that you make to Oracle Fusion applications using Page Composer and Oracle Fusion CRM Application Composer (Application Composer). It also describes how to localize your navigator and home page menu customizations and your **descriptive flexfield** and **extensible flexfield** configurations.

This chapter includes the following sections:

9.1 About Translating Custom Text

If your Oracle Fusion Applications are running in different locales, you can localize your customizations such that end users see the custom text in the language of their locale. End users set their locale when they log in. Users can also set their locale by choosing **Set Preferences** from the **Personalization** menu in the Oracle Fusion Applications global area.

Most user interface text is made available to applications through **resource bundles**. These resource bundles are stored in an Oracle Metadata Services (MDS) repository in XML localization interchange file format (XLIFF). To provide locale translations for your Page Composer, Application Composer, navigator menu, and home page menu changes, you export, edit, and import XLIFF documents. For flexfield and value set configurations, you provide locale translations using the appropriate maintenance tasks.

For information about XLIFF documents, see the "Manually Defining Resource Bundles and Locales" section in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

9.2 Translating Resource Bundles from an MDS Repository

You use the Oracle WebLogic Scripting Tool (WLST) `exportMetadata` command to obtain XLIFF documents and you use the WLST `importMetadata` command to import XLIFF documents into an MDS repository. For information about MDS Repository and

the `exportMetadata` and `importMetadata` commands, see the "Managing the Metadata Repository" chapter in the *Oracle Fusion Middleware Administrator's Guide*.

> **Tip:** You can also use Oracle Enterprise Manager Fusion Applications Control to import and export the XLIFF documents from an MDS repository. For more information, see the "Transferring Metadata Using Fusion Middleware Control" section in the *Oracle Fusion Middleware Administrator's Guide*. The referenced procedure describes using Fusion Middleware Control, but also applies to Fusion Applications Control.

For specific information about localizing Page Composer and Application Composer customizations, see Section 9.3, "Translating Page Composer and Application Composer Customizations." For specific information about localizing navigator and home page menu customizations, see Section 9.4, "Translating Menu Customizations."

Task: Define Translations for the Custom Text in an MDS Repository

You define the translations for custom text by exporting XLIFF documents from an MDS repository, editing the documents to include the translated text, and importing the revised documents into the repository.

To localize the custom text:

1. Use the WLST `exportMetadata` command shown in Example 9–1 to export XLIFF documents from the MDS repository to a directory of your choice.

Example 9–1 WLST exportMetadata Command

```
exportMetadata(application='application', server='server',
toLocation='directory-path',
docs='xlf-classpath', applicationVersion='version')
```

   Set the `docs` attribute to the class path for the XLIFF file. For example, use `/oracle/apps/resourcebundles/xliffBundle/FusionAppsOverrideBundle.xlf` for the base file for Page Composer and Application Composer custom text. Use `/oracle/apps/menu/CustResourceBundle.xlf` for the base file for menu custom text. Use the following format for the names of locale documents:

   ```
   basename_language[_country].xlf
   ```

   Replace *language* with the ISO 639 lowercase language code, such as `fr` for France. When applicable, replace *country* with the ISO 3166 uppercase country code. Country codes are necessary when one language is used by more than one country. For example, use `CustResourceBundle_zh_CN.xlf` for custom translations for Chinese in the People's Republic of China.

   Because all Oracle Fusion applications use the same repository partition, you can use any Oracle Fusion application as an argument for the `application` attribute when you export an XIFF file for text customizations.

2. Synchronize the entries in the XLIFF documents and provide the translated text in the `<target>` tags, as shown in Example 9–2.

Example 9–2 Sample Translation

```
<trans-unit id="ACCOUNTING_DISTRIBUTION">
  <source>Accounting Distribution</source>
  <target>Ventilation comptable</target>
  <note>Accounting Distribution</note>
```

```
</trans-unit>
```

3. Use the WLST `importMetadata` command shown in Example 9–2 to import the modified documents into the MDS repository.

Example 9–3 WLST importMetadata Command

```
importMetadata(application='application', server='server',
fromLocation='directory-path',
docs='xlf-classpath', applicationVersion='version')
```

Because all Oracle Fusion applications use the same repository partition, you can use any Oracle Fusion application as an argument for the `application` attribute when you import an XIFF file for text customizations.

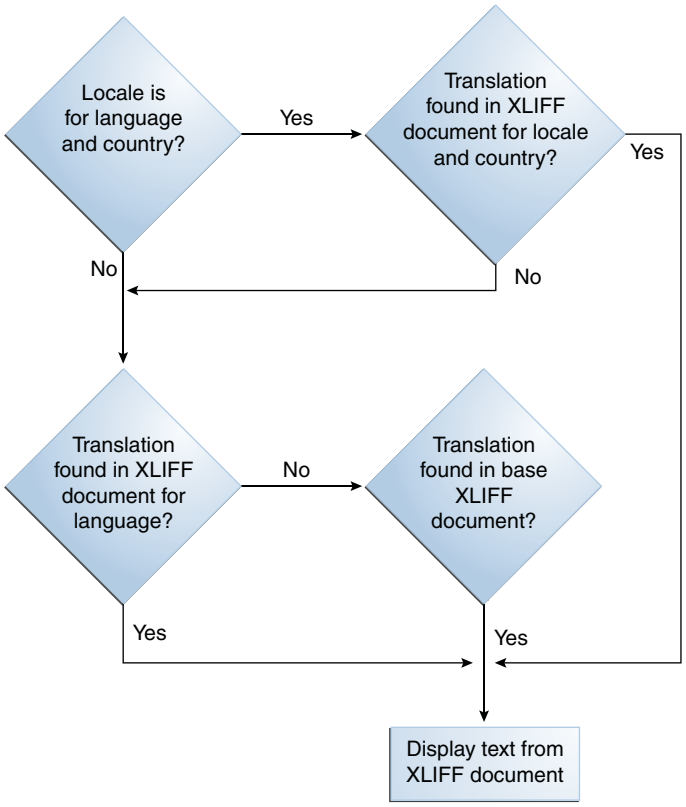
For more information about naming and editing XLIFF files, see the "Manually Defining Resource Bundles and Locales" section in the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

9.3 Translating Page Composer and Application Composer Customizations

All Page Composer and Application Composer customizations are stored in the customizations XLIFF document for the locale of the session in which you made the customizations. After you customize a page using Page Composer or Application Composer, you might want to define translations for the custom text in the base customizations file as well as the customizations files for the other supported locales. For example, you might want to define French and Chinese translations of new prompts.

As shown in Figure 9–1, when an end user accesses the customized objects, the application loads the translated custom text for the locale's language and, if applicable, country. If the user's locale is for a language in a specific country and customized text is not available for that locale, the application loads the text for the locale's language. If no translated text is found, the application loads the text from the base customizations document.

Figure 9–1 Process for Retrieving Translated Text



Note that Figure 9–1 does not show a No path for the condition where the translation is not found in the base XLIFF document. If no entries exist in the locale and base documents, the text that is displayed varies. For example, for a field label, the application displays the attribute name. In other cases, no text is displayed.

To define translations for custom text, follow the steps in Task: Define Translations for the Custom Text in an MDS Repository in Section 9.2, "Translating Resource Bundles from an MDS Repository." Export the base document `/oracle/apps/resourcebundles/xliffBundle/FusionAppsOverrideBundle.xlf` and the documents for all the locales for which you want to define translations. The locale XLIFF documents are named `/oracle/apps/resourcebundles/xliffBundle/FusionAppsOverrideBundle_`*language*`[_`*country*`].xlf`. Replace *language* with the ISO 639 lowercase language code, such as `fr` for France. When applicable, replace *country* with the ISO 3166 uppercase country code. Country codes are necessary when one language is used by more than one country. For example, use `/oracle/apps/resourcebundles/xliffBundle/FusionAppsOverrideBundle_zh_CN.xlf` for custom translations for Chinese in the People's Republic of China.

Note: The base document `/oracle/apps/resourcebundles/xliffBundle/FusionAppsOverrideBundle.xlf` is automatically generated the first time that a string is inserted or customized using Page Composer or Application Composer. Ensure that the bundle exists by inserting or customizing at least one string.

Copy the new and changed entries from the document for the locale with which you made the customizations into the base document and into the other locale documents. Provide the translations and import the modified documents into the MDS repository.

9.4 Translating Menu Customizations

All navigator and home page menu customizations are stored in the `/oracle/apps/menu/CustResourceBundle.xlf` base XLIFF document regardless of your locale setting when you customized the menu. After you customize the menu, you might want to define translations for your changes in the locales that you support, including the locale for the session in which you entered the custom text. For example, you might want to define French and Chinese translations of new menu items.

The process for retrieving translated text is the same as for Page Composer and Application Composer, as shown in Figure 9–1, with the exception that if no entries exist in the locale and base documents, no text is displayed.

To create locale translations for your menu changes, follow the steps in Task: Define Translations for the Custom Text in an MDS Repository in Section 9.2, "Translating Resource Bundles from an MDS Repository." Export the base document `/oracle/apps/menu/CustResourceBundle.xlf` and export the documents for all the locales for which you want to define translations. The locale XLIFF documents are named `/oracle/apps/menu/CustResourceBundle_`*language*`[_`*country*`].xlf`. Replace *language* with the ISO 639 lowercase language code, such as `fr` for France. When applicable, replace *country* with the ISO 3166 uppercase country code. Country codes are necessary when one language is used by more than one country. For example, use `/oracle/apps/menu/CustResourceBundle_zh_CN.xlf` for custom translations for Chinese in the People's Republic of China.

Copy the new and changed entries from the base document into the locale documents and provide the translations. Then import the modified locale documents into the MDS repository.

9.5 Translating Flexfield and Value Set Configurations

When you first configure a flexfield or **segment**, the translatable text that you enter, such as prompts and descriptions, is stored as the text for all installed locales. To translate the text for a particular locale, log in with that locale or use the **Personalization** menu in the global area to set the locale. Then, update the translatable text in the flexfield using the Manage Descriptive Flexfields task or the Manage Extensible Flexfields task as described in the "Define Flexfields" section in the *Oracle Fusion Applications Common Implementation Guide*. Your modifications change the translated values only for the current session's locale.

After you complete the translations, deploy the flexfield.

You can define translations for a **dependent value set** or an **independent value set**, if it is of type Character with a subtype of Translated text. You define the translations by setting the current session to the locale for which you want to define the translation and using the Manage Value Sets task to enter the translated values and descriptions for that locale as described in the "Define Flexfields" section in the *Oracle Fusion Applications Common Implementation Guide*.

For a **table value set** for which the underlying table supports multiple languages and for which the value set's value column is based on a translated attribute of the underlying table, you can define translated values using the maintenance task for the underlying table. For more information about enabling localization for table value sets, see the "Define Flexfields" section in the *Oracle Fusion Applications Common*

Implementation Guide. For information about multilanguage support for tables, see the "Using Multi-Language Support Features" section in the *Oracle Fusion Applications Developer's Guide*.

10

Configuring End-User Personalization

This chapter describes how you can make pages in your Oracle Fusion application personalizable by the end user. Note that mobile applications cannot be personalized by the end user.

This chapter contains the following sections:

10.1 About Configuring End-User Personalization

There are certain runtime changes that an end user can make that persist from session to session, such as changing the width of a column in a table, saving a search parameter, or redesigning an aspect of a page. This type of change is called **personalization**. Oracle Fusion applications allow end users to personalize certain pages using the **Personalization** menu. End users can set preferences, edit the current page, and reset the page to the default.

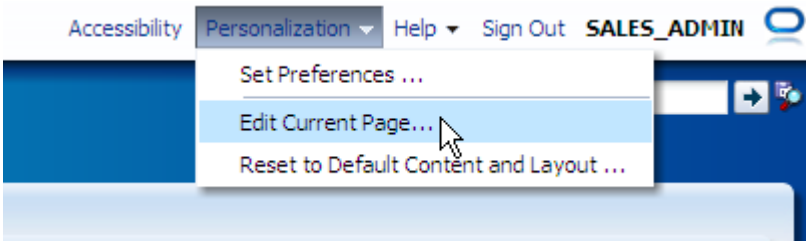
You can control what pages in an application can be personalized, including any new pages you create.

> **Tip:** If you created a page using Oracle Fusion CRM Application Composer (Application Composer), then that page is personalizable by default.

> **Note:** For a list of pages that end users can personalize, see the product-specific documentation in Oracle Fusion Applications Help.

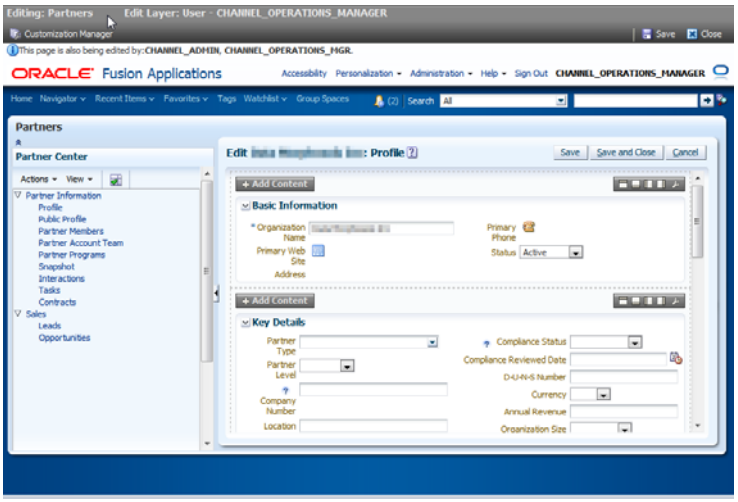
Figure 10–1 shows the Personalization menu available in all Oracle Fusion applications.

Figure 10–1 Personalization Menu in Oracle Fusion Applications



When end users choose the **Edit Current Page** menu item, Page Composer is opened. From here, they can change certain aspects of the page, such as moving or deleting a **component**. Figure 10–2 shows the Partner Profile application home page in Page Composer, ready for the end user to personalize.

Figure 10–2 Home Page Ready for Personalization



Along with using Page Composer to personalize pages, end users can change certain aspects of components, and then have those changes saved so that they remain each time the user logs in to the application. For example, end users can change the width of columns in many of the tables in Oracle Fusion applications. However, by default, when they change the width, that new width size is saved only for the current session. You can configure that column so that when the user changes the width size, it will remain at that size whenever the user logs back in to the application. For more information about configuring persistence, see the "Allowing User Customizations at Runtime" chapter in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

10.1.1 Before You Begin Allowing Pages or Components to be Personalized

Before you configure pages to be personalizable, you should be familiar with the Oracle Fusion application architecture that enables customization, as described in Chapter 1, "Customizing and Extending Oracle Fusion Applications." You should also understand the typical workflows for working with customizations, as described in Chapter 2, "Understanding the Customization Development Lifecycle."

You will also need to do the following:

- Install Oracle JDeveloper and set up your development environment. For more information, see Section 1.3.13, "Installing Customization Tools."

- Create a customization application workspace. For more information, see Chapter 3, "Using Oracle JDeveloper for Customizations."
- Start JDeveloper in the Oracle Fusion Applications Administrator Customization role.
- Select a layer value. When customizing application artifacts in JDeveloper, you first must select the layer and layer value to work in. You use the Customization Context window to make this selection. For more information about customization layers, see Section 1.2, "Understanding Customization Layers."

10.2 Allowing Pages to Be Personalized by End Users in Page Composer

You use JDeveloper to set certain attributes that allow a page to be personalized.

Task: Enable or Disable Personalization on Existing Standard Pages

Many pages in Oracle Fusion applications allow **personalization** by default. You can either disable it or enable it using the `isPersonalizableInComposer` property on a page. Set the property to `true` to allow personalizations, set it to `false` to disallow personalizations. For instructions, see the "How to Enable End-User Personalizations for a Page" section in the *Oracle Fusion Applications Developer's Guide*.

Task: Enable Page Composer Personalization on Custom Pages

For end users to be able to use Page Composer to personalize custom pages, you will need to enable your pages to work with Page Composer by doing the following:

- Set the `isPersonalizableInComposer` property to `true`.

  For instructions, see the "How to Enable End-User Personalizations for a Page" section in the *Oracle Fusion Applications Developer's Guide*.

  Note: If a page is currently available for personalization, and you do not want it to be, change the property value to `false`.

- Create a corresponding page definition file, if one does not exist.

  For instructions, see the "Ensuring Customizable Pages Have Page Definitions" section in the *Oracle Fusion Applications Developer's Guide*.

- Use Oracle WebCenter Portal components that define areas that are customizable.

  For instructions, see the "Making a JSPX Document Editable at Runtime" section in the *Oracle Fusion Applications Developer's Guide*.

10.3 Configuring End-User Personalization for Components

Certain attribute values that affect how an ADF Faces component is displayed can persist to an MDS repository. Application-wide component attribute persistence to an MDS repository is controlled by configuration in the `adf-config.xml` file. However, customizing this file is not allowed, because doing so is not upgrade-safe. Instead, you can override the application-wide persistence at the page level by setting the `persist` and `dontPersist` attributes for component instances.

For example, by default, table column attribute values do not persist. But you can configure a column in a table so that when the user changes the width, reorders columns, or selects a column, those changes will still be in effect when the user logs back in to the application, by adding those attributes to the value of the `persist`

attribute on the column component. For more information about what attribute values can persist, see the "Introduction to Allowing User Customizations" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Note: You cannot change the settings in the `adf-config.xml` file, because these changes will be overwritten anytime you apply a patch or an upgrade. Therefore, you must change the values on the individual components on a page.

Task: Persist Attribute Values on JSPX Pages

You need to add the attributes you want to persist to the `persist` attribute on the component. For more information, see the "Controlling User Customizations in Individual JSF Pages" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. You can set this attribute using Page Composer.

11 Customizing Help

This chapter describes how you can customize or extend user assistance help in your Oracle Fusion application to match your runtime and design time customizations.

This chapter contains the following sections:

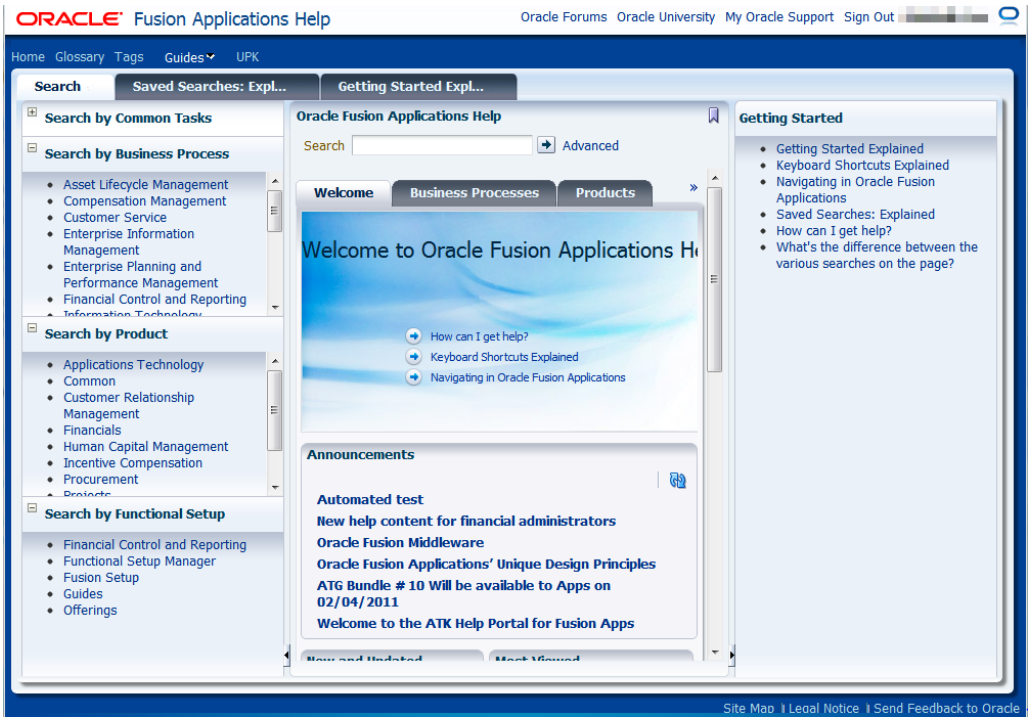
11.1 About Customizing Help

When you customize an Oracle Fusion application, you may find you also need to customize or extend the existing help to match your changes. Oracle Fusion applications provide two different types of help:

- Oracle Fusion Applications Help

  This type of help includes help topics, FAQs, examples, demonstrations and PDF guides, and is delivered with the Oracle Fusion Applications Help as shown in Figure 11–1.

Figure 11–1 Oracle Fusion Applications Help



- Embedded static page-level help

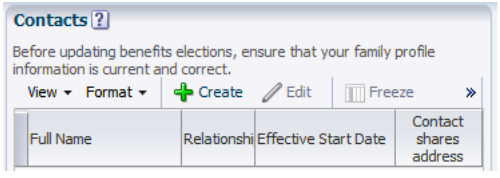
  This type of help is displayed directly on a page, using attributes of a **component**. The help text is included in the application.

  > **Tip:** Help text is stored in **resource bundles**, and so can be translated. For more information, see Chapter 9, "Translating Custom Text."

  Embedded help includes the following:

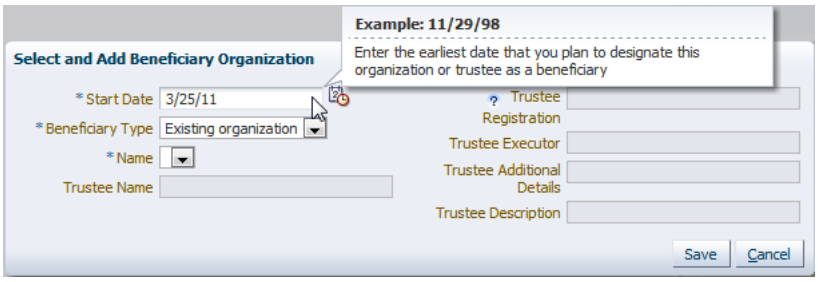
  – Static instruction text: displayed by panel components that typically contain forms or tables. This instruction guides the user in filling out the form or using the table, as shown in Figure 11–2.

Figure 11–2 Static Help Text in Oracle Fusion Applications



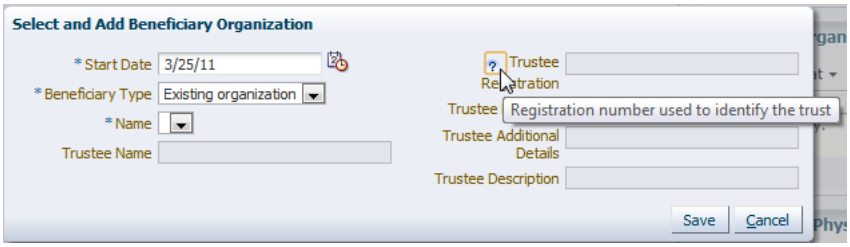
  – In-field help note: displayed by input components and guides the user in entering data into the component. Figure 11–3 shows an in-field note.

Figure 11–3 In-Field Note in Oracle Fusion Applications



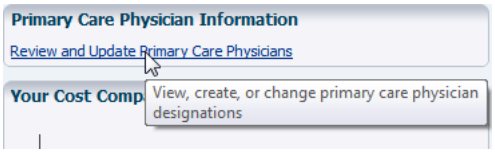
– Terminology definition: displayed by input components and defines terms used on the page, as shown in Figure 11–4.

Figure 11–4 Terminology Definition in Oracle Fusion Applications



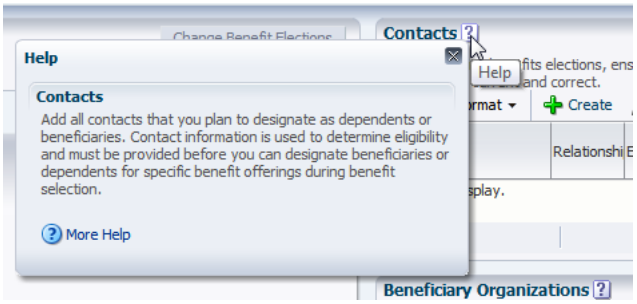
– Bubble help: displayed when the end user mouses over a button or link component, as shown in Figure 11–5.

Figure 11–5 Bubble Help in Oracle Fusion Applications



– Help window: displayed when a user clicks the help icon, as shown in Figure 11–6. This type of help is generally brief context-sensitive help, and can also provide links to help files in Oracle Fusion Applications Help.

Figure 11–6 Help Window in Oracle Fusion Applications



11.1.1 What You Can Do with Help

In Oracle Fusion Applications Help, you can change the content in existing help windows or you can create new help windows. Within a page of an application, you

can customize or create bubble help, static instructions, in-field notes, terminology definitions, and help windows. This help text is stored either as a value for an attribute, or in translatable resource bundles.

Note: You can customize the help menu to add a link to a privacy statement and to make Oracle User Productivity Kit (UPK) content available for users. For information, see the "Setting Up Help" section in the *Oracle Fusion Applications Post-Installation Guide*.

11.1.2 Before You Begin Customizing Help

Before you customize help, you should be familiar with the Oracle Fusion application architecture that enables customization, as described in Chapter 1, "Customizing and Extending Oracle Fusion Applications." You should also understand the typical workflows for working with customizations, as described in Chapter 2, "Understanding the Customization Development Lifecycle."

You will also need to do the following before you can begin customizing help:

- If you will be adding or customizing Oracle Fusion Applications Help, then you will need specific privileges. Contact your security administrator for details.
- Install Oracle JDeveloper and set up your development environment. For more information, see For more information, see Section 1.3.13, "Installing Customization Tools."
- Create a customization application workspace. For more information, see Chapter 3, "Using Oracle JDeveloper for Customizations."
- Start JDeveloper in the Oracle Fusion Applications Administrator Customization role.
- Select a layer value. When customizing application artifacts in JDeveloper, you first must select the layer and layer value to work in. You use the Customization Context window to make this selection. For more information about customization layers, see Section 1.2, "Understanding Customization Layers."

11.2 Customizing or Extending Oracle Fusion Applications Help

You can customize existing help files in Oracle Fusion Applications Help, or you can extend Oracle Fusion Applications Help by adding custom topics.

After they are created, custom help files are distinguished by an icon in search results, and they are displayed at the top of help listings when you navigate.

Task: Customize Oracle Fusion Applications Help Windows

When you have the necessary privileges, help windows in Oracle Fusion Applications Help display a **Manage Custom Help** link, which allows you to change the content and specify in which help windows in the application your custom help will appear, and where it will appear in the help site navigators. For more information, see the "Define Help Configuration" section in the *Oracle Fusion Applications Common Implementation Guide*.

Task: Add Custom Help Files to Oracle Fusion Applications Help

You can add new custom help files to Oracle Fusion Applications Help. Custom help files will appear like standard help files and can be searched and included in help

windows and navigators. For more information, see the "Define Help Configuration" section in the *Oracle Fusion Applications Common Implementation Guide*.

11.3 Customizing or Adding Bubble Embedded Help

For bubble help, you can use Oracle Fusion CRM Application Composer (Application Composer) or Page Composer to customize or create the help text.

The following components use bubble help.

- `Butcon`
- `Button`
- `Link`
- `Tab`

Task: Customize or Add Bubble Help

The text displayed in bubble help is the value of the component's `shortDesc` attribute. Usually, the value resolves to a key in a **resource bundle**. If you are customizing a CRM application, use Application Composer to customize the value of the attribute. For other applications, use Page Composer to customize the attribute.

11.4 Customizing or Adding Static Instructions, In-Field Notes, and Terminology Definitions

Oracle Fusion Applications embedded help (aside from bubble help) uses two types of ADF Faces help: *instruction* and *definition*. Instruction-type help displays static text, either in a specified area on a component (like static instruction help, shown in Figure 11–2), or in a note window, as in-field notes do, shown in Figure 11–3. Definition-type help displays a help icon, and is what terminology definition embedded help uses, as shown in Figure 11–4. When the user mouses over the help icon, the help text is displayed in a message box. UI components display the instruction and definition help text using the `helpTopicId` attribute. For more information about the ADF Faces help framework, see the "Displaying Help for Components" section of the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

It is important that for the type of help you want to add or customize, you understand which component actually displays the help, and which type of ADF Faces help is being used. Table 11–1 shows the different types of Oracle Fusion Applications embedded help, the corresponding ADF Faces help, and the components that display that type of help.

Table 11–1 Oracle Fusion Applications Help and Corresponding ADF Faces Help and UI Components

Oracle Fusion Applications Help Type	ADF Faces Help Type	Component
Static instruction	instruction	Page header
		Subheader
		Sub-subheader

Table 11–1 (Cont.) Oracle Fusion Applications Help and Corresponding ADF Faces Help and UI Components

Oracle Fusion Applications Help Type	ADF Faces Help Type	Component
In-field note	instruction	Multiselect checkbox group
		Single-select choice list
		Multiselect choice list
		Single-select list box
		Multiselect list box
		Text box
		Single-select radio groups
		Items in true/false radio groups
		Items in true/false checkbox groups
		Color picker
		Date/time picker
		Flexfield
		LOV
		Spin box
		Slider
		File upload
		Shuttle
		Rich Text Editor
Terminology definition	definition	Checkbox prompt
		Checkbox group prompt
		Single-select choice list
		Multiselect choice list
		Single-select list box
		Multiselect list box
		Text box
		Radio group prompt
		Color picker
		Date/time picker
		Flexfield
		LOV
		Column headers
		Spin box
		Slider
		File upload
		Shuttle
		Rich Text Editor

You perform the following tasks in JDeveloper in the Oracle Fusion Applications Administrator Customization role.

Note: You cannot directly customize the existing help text strings. If you want to change text that currently appears, you must create a new text string and associate the component with that new text.

Task: Add Help Strings to Resource Bundle

Add custom help text strings to an existing custom resource bundle or create a new resource bundle to hold your customized help text (Oracle Fusion applications use XLIFF files for resource bundles). If you create a new resource file, you must register that file with the project. For information about creating and using resource bundles for an Oracle Fusion application, see Section 4.12, "Customizing or Adding Resource Bundles."

The help text must use the following syntax:

- `<trans-unit>`: Enter the topic ID. This must contain a unique prefix, the topic name, and the help type, either `INSTRUCTION` or `DEFINITION`.

  Note: You prefix must unique. You must use this prefix for all your custom help strings.

  For example:

  ```
  MYCUSTHELP_NEWHELPTOPIC_DEFINITION
  ```

  In this example, `MYCUSTHELP` is the prefix used to access the XLIFF file. `NEWHELPTOPIC` is the topic name, and `DEFINITION` is the type of ADF Faces help.

  UI components access the help content based on the topic name. Therefore, if you use the same topic name for two different types of help (instruction and definition), then both types of help will be displayed by the UI component.

- `<source>`: Create as a direct child of the `<trans-unit>` element and enter the help text.
- `<target>`: Create as a direct child of the `<trans-unit>` element and leave it blank. This will hold translated text populated by translation tools.
- `<note>`: Create as a direct child of the `<trans-unit>` element and enter a description for the help text.

Example 11–1 shows a resource file that contains two topics.

Example 11–1 XLIFF Resource Bundle

```
<?xml version="1.0" encoding="UTF-8" ?>
<xliff version="1.1" xmlns="urn:oasis:names:tc:xliff:document:1.1">
  <file source-language="en" original="this" datatype="xml">
    <body>
      <trans-unit id="MYCUSTHELP_NEWHELPTOPIC_DEFINITION">
        <source>Credit Card Definition</source>
        <target/>
        <note>This is the credit card definition text.</note>
      </trans-unit>
      <trans-unit id="MYCUSTHELP_NEWTOPIC2_INSTRUCTIONS">
        <source>Credit Card Instructions</source>
        <target/>
        <note>This is the credit card instruction text.</note>
```

```
      </trans-unit>
    </body>
  </file>
</xliff>
```

Task: Associate the Component with the Help Strings

In JDeveloper, select the component to display the help. Associate that component with the `<trans-unit>` element in the resource bundle, using the component's `helpTopicID` attribute. Ensure that the component supports the type of help (that is, definition or instruction) defined for the `id` attribute. For instructions, see the "How to Access Help Content from a UI Component" section of the *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

12

Customizing the Oracle Fusion Applications Skin

This chapter describes how to use Oracle Application Development Framework (Oracle ADF) Skin Editor to change the look and feel of Oracle Fusion applications.

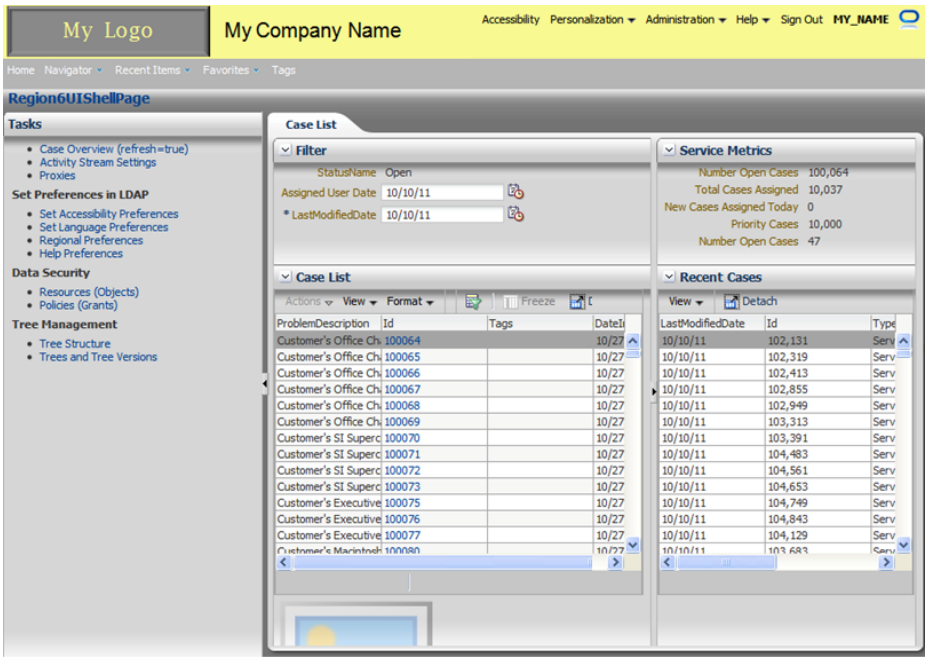
This chapter includes the following sections:

12.1 Introduction to Skinning Oracle Fusion Applications

If you want to make changes to the appearance of Oracle Fusion Applications pages, such as changing the logo or changing the colors to make the pages adhere to your company's corporate brand, use ADF Skin Editor to create a custom **skin** based on the Oracle Fusion Applications Skin Extension (fusionFx-simple) and apply that skin to your Oracle Fusion applications. You can apply a custom skin to the whole site, to specific products, or specific end users. The changes that you make using custom skins are maintained through future patches and upgrades of Oracle Fusion Applications.

The fusionFx-simple skin extension is a special type of cascading style sheet (CSS) that enables you to customize the appearance of Oracle Fusion Middleware Extensions for Applications (Applications Core) components, ADF Faces components, and ADF Data Visualization components. Figure 12–1 shows an example of an application that has been skinned using the fusionFx-simple skin extension.

Figure 12–1 Example of a Skinned Application



12.1.1 Before You Begin Customizing the Oracle Fusion Applications Skin

Before you implement customizations in applications, you should be familiar with the ADF skinning framework and how to use it to create a custom skin. You should also be familiar with how to use the editor to work with Applications Core components. For information about the ADF skinning framework and working with Applications Core components, as well as how to install and start ADF Skin Editor, see the Downloads for Oracle ADF 11g page at
http://www.oracle.com/technetwork/developer-tools/adf/downloads/index.html

You will need to do the following before you can begin customizing the Oracle Fusion Applications skin:

1. Download ADF Skin Editor from the Downloads for Oracle ADF 11g page at http://www.oracle.com/technetwork/developer-tools/adf/downloads/index.html
2. Install and start ADF Skin Editor.
3. Choose **Check for Updates** from the **Help** menu to install the most current release of the Oracle Fusion Applications Skin Extension (fusionFx-simple).

12.2 Creating a Custom Oracle Fusion Applications Skin

To create and modify a custom skin for your Oracle Fusion applications, use ADF Skin Editor to create a project, add a skin based on the fusionFx-simple skin extension, and modify the ADF Faces **component**, ADF Data Visualization component, and Applications Core component styles.

Task: Create a Custom Oracle Fusion Applications Skin

You create a skin by creating a new application in ADF Skin Editor and then creating a new ADF skin file in the project. Ensure that you set the project's target application release to the Oracle Fusion Applications release. When you create the ADF skin file,

select the appropriate fusionFx-simple version from the **Extends** dropdown list. Make a note of the family name from the Create ADF Skin File dialog. You use this name when you apply the skin to your Oracle Fusion applications.

Task: Modify the Component Styles in the Custom Skin

Use ADF Skin Editor to change the look and feel of ADF Faces and ADF Data Visualization components. The fusionFx-simple skin extension additionally enables you to modify Applications Core components, which define how the Oracle Fusion Applications template and extensible components appear. You find the Applications Core components by expanding **Style Classes** in the Design view.

For information about modifying the look and feel of Applications Core components, see *Skinning Oracle Fusion Applications*, which is available at `http://www.oracle.com/technetwork/fusion-apps/tools/downloads/index.html`

12.3 Applying a Custom Skin to Your Oracle Fusion Applications

When you are ready to apply your custom skin to your Oracle Fusion applications, you deploy the custom skin to an ADF Library JAR file. You then copy the custom skin JAR file plus supporting JAR files to the installation directories of the applicable Oracle Fusion applications, restart the applications, and set the profile option to use the custom skin.

Task: Deploy the Custom Skin to an ADF Library JAR File

Deploy the skin project to a JAR file that can be included in an application.

To create the JAR file:

1. Right-click the skin project, choose **Deploy**, and choose **New Deployment Profile** to display the Create Deployment Profile dialog.
2. Select **ADF Library JAR File** from the **Profile Type** dropdown list.
3. Set the **Deployment Profile Name** to a name that begins with `Xx_`. The `Xx_` prefix signifies to future patches and upgrades that this deployment is customer-provided and must not be touched.
4. Click **OK**.
5. Right-click the skin project, choose **Deploy**, and choose the profile name to display the Deploy *profile name* dialog.
6. Click **Finish**.
7. Right-click the skin project, choose **Deploy**, and choose *profile name* **to JAR file**.

Task: Add the Custom Skin JAR Files to Your Oracle Fusion Applications

You must make the custom skin JAR file and skin support JAR files available to Oracle Fusion Applications before you can apply the skin.

Copy the following JAR files to the `WEB-INF/lib` directory of every Oracle Fusion application:

- *skin-editor-installation-dir*`/jlib/adf-richclient-fusion-simple-`*version*`.jar`
- `XxApplCoreSkin.`*version*`.jar`. Download this JAR file from OTN at `http://www.oracle.com/technetwork/fusion-apps/tools/downloads/index.html`
- The ADF Library JAR file for your custom skin.

After you add the custom JAR files, you must stop and restart the Oracle Fusion applications as described in the "Starting and Stopping" section in the *Oracle Fusion Applications Administrator's Guide*.

Task: Apply the Custom Skin to Your Oracle Fusion Applications

You use the Manage Profile Option Values page in the Manage Profile Options task from the Setup and Maintenance work area to apply your custom skin to Oracle Fusion applications. You can set this value at the site, product, or user level. You typically set the option at the site level, however, if you want to test the skin, you can set it at the user level.

To use your custom skin, change the value of the FND_CSS_SKIN_FAMILY profile option to the skin family attribute that you set when you created your skin. If you do not know the skin family attribute, you can find it in the skin project's `trinidad-skins.xml` file.

Note: If you set the profile option at the site level, but you did not copy the necessary skin JAR files into the `WEB-INF/lib` directory of every Oracle Fusion application, the applications with the missing files will be displayed with a simple skin that has a basic black-and-white look.

Part III

Appendixes

This part contains information about troubleshooting Oracle Fusion Applications extensions and customizations. It contains the following appendix:

- Appendix A, "Troubleshooting Customizations"

A

Troubleshooting Customizations

This appendix describes common problems that you might encounter when using design time tools to customize and extend Oracle Fusion Applications and explains how to solve them.

This appendix contains the following topics:

- Section A.1, "Introduction to Troubleshooting Customizations"
- Section A.2, "Getting Started with Troubleshooting and Logging Basics for Customizations"
- Section A.3, "Resolving Common Problems"
- Section A.4, "Using My Oracle Support for Additional Troubleshooting Information"

In addition to this appendix, review *Oracle Fusion Middleware Error Messages Reference* for information about the error messages you may encounter.

A.1 Introduction to Troubleshooting Customizations

Use the following guidelines and process within this appendix to help focus and minimize the time you spend resolving problems.

Guidelines

When using the information in this appendix, consider the following guidelines:

- After performing any of the solution procedures in this appendix, immediately retry the failed task that led you to this troubleshooting information. If the task still fails when you retry it, perform a different solution procedure in this appendix and then try the failed task again. Repeat this process until you resolve the problem.
- Make notes about the solution procedures you perform, symptoms you see, and data you collect while troubleshooting. If you cannot resolve the problem using the information in this appendix and you must log a service request, the notes will expedite the process of solving the problem.

Process

Follow the process outlined in Table A–1 when using the information in this appendix. If the information in a particular section does not resolve your problem, proceed to the next step in this process.

Table A–1 Process for Using the Information in this Appendix

Step	Section to Use	Purpose
1	Section A.2	Get started troubleshooting customizations. The procedures in this section quickly address a wide variety of problems.
2	Section A.3	Perform problem-specific troubleshooting procedures for customizations. This section describes: ■ Common problems ■ Solution procedures corresponding to each of the possible problems
3	Section A.4	Use My Oracle Support to get additional troubleshooting information about Oracle Fusion Applications or Oracle Business Intelligence. My Oracle Support provides access to several useful troubleshooting resources, including Knowledge Base articles and Community Forums and Discussions.
4	Section A.4	Log a service request if the information in this appendix and My Oracle Support does not resolve your problem. You can log a service request using My Oracle Support at `https://support.oracle.com`.

A.2 Getting Started with Troubleshooting and Logging Basics for Customizations

This section provides the following general approaches for managing and diagnosing customization issues:

- Exporting Customizations
- Backing Up and Restoring Customizations
- Choosing the Right Customization Layer
- Determining the Full Path for a Customizations Document
- Determining Whether a Customization Layer is Active
- Logging Customizations that Are Applied to a Page

A.2.1 Exporting Customizations

Customizations are stored in XML files. You can export the customizations for diagnosing issues in a number of ways, as described in Section 2.3, "Exporting and Moving Customizations."

A.2.2 Backing Up and Restoring Customizations

Before you make customizations, you can create a backup of a known good state by creating a label. If an issue occurs after creating the label, you can revert back to that label by promoting it to the tip as described in the "Creating Metadata Labels" and "Promoting Metadata Labels" sections of the *Oracle Fusion Middleware Administrator's Guide*.

Another way to back up and restore customizations is by exporting and importing customization files, as described in Section A.2.1, "Exporting Customizations."

A.2.3 Choosing the Right Customization Layer

When you make customizations, be careful to choose the correct layer:

- Use the site layer for customizations that affect all end users.
- Use the global layer for ADF Business Components customizations.
- Use product-specific layers appropriately as documented.

A.2.4 Determining the Full Path for a Customizations Document

The following string shows the structure of the full document path for a customization document:

`/`*`package`*`/mdssys/cust/`*`layer-name`*`/`*`layer-value`*`/`*`document-name`*`.`*`suffix`*`.xml`

For example, the full document path for the Visa Work Permit Expiration region is `/oracle/apps/hcm/dashboard/hrSpecialist/publicUI/page/mdssys/Site/SITE/VisaWorkPermitExpirationRegion.jsf.xml`.

You can obtain the full document path of a customized region on a page by completing the following steps:

1. Go to the page that contains the customized region and choose **Customize** *page_name* **Pages** from the **Administration** menu in the global area of Oracle Fusion Applications to open Page Composer.
2. If you have more than one layer available for customization, the Layer Picker dialog is displayed. In the **Edit** column, select the desired layer.
3. Choose **Source** from the **View** menu.
4. In the hierarchical list, drill down to and hover over the customized region to display the full document path of the JSF fragment that contains the customization, such as `/oracle/apps/hcm/dashboard/hrSpecialist/publicUI/page/mdssys/Site/SITE/VisaWorkPermitExpirationRegion.jsf.xml`. Make a note of this path.

For descriptive flexfield configurations, you can use the Register Descriptive Flexfields task to find the name of the flexfield's package.

A.2.5 Determining Whether a Customization Layer is Active

Customizations do not appear if the customization layer is not active in an application. To determine if a customization layer is active, open the `adf-config.xml` file for the application and look for the `<cust-config>` tag, as shown in Example A–1. The nested `<customization-class>` tags show the active layers.

Example A–1 Active Customization Layers

```
<adf-mds-config xmlns="http://xmlns.oracle.com/adf/mds/config">
<mds-config xmlns="http://xmlns.oracle.com/mds/config" version="11.1.1.000">
<cust-config>
  <match path="/">
    <customization-class name ="oracle.apps.fnd.applcore.customization.GlobalCC"/>
    <customization-class name ="oracle.apps.fnd.applcore.customization.SiteCC"/>
    <customization-class name ="oracle.apps.fnd.applcore.customization.UserCC"/>
  </match>
</cust-config>
</mds-config>
</adf-mds-config>
```

A.2.6 Logging Customizations that Are Applied to a Page

To turn on runtime logging for customizations that are applied to a page, set the log level for the `oracle.mds.custmerge` module to `FINEST`. You can set the application's log level by choosing **Troubleshooting** from the **Help** menu. You might need to ask your administrator to give you privilege to set the log level.

If you have administration privileges, you can also use Fusion Applications Control to set the log level.

A.3 Resolving Common Problems

The following are common problems and solutions:

- User Interface is not Displaying the Active Sandbox Customizations
- Customizations Context Table Is Empty in Oracle JDeveloper
- Application Is Not Displayed Correctly After Applying a Customized Skin
- Finding the EAR File for an Application

A.3.1 User Interface is not Displaying the Active Sandbox Customizations

Problem

The customizations that were made in the active sandbox are not appearing in the user interface.

Solution

Sign out and sign back in.

To ensure that the sandbox customization cache is cleared, log out and log back in before you enter a sandbox and after you perform any of the following sandbox-related actions:

- Exit a sandbox
- Publish a sandbox
- Delete a sandbox

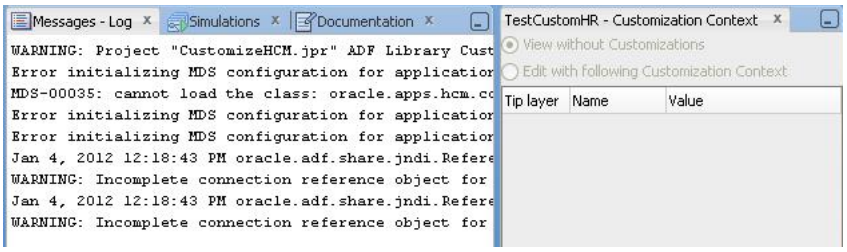
A.3.2 Customizations Context Table Is Empty in Oracle JDeveloper

Problem

You are using JDeveloper in the Oracle Fusion Applications Administrator Customization role. The Customization Context table does not display the customization classes, as shown in Figure A–1, and the messages log displays an error message similar to the following text:

```
Error initializing MDS configuration for application
"file:/somepath/TestCustomHR.jws". Customizations disabled for this application.
MDS-00035: cannot load the class: oracle.apps.hcm.common.core.HcmCountryCC
```

Figure A–1 Empty Customization Context Table



Solution

Enable JDeveloper to see the customization classes that define the customization layers as described in Section 3.1.3, "Before You Begin Using JDeveloper to Customize."

A.3.3 Application Is Not Displayed Correctly After Applying a Customized Skin

Problem

After applying a customized skin that is based on the Oracle Fusion Applications Skin Extension (fusionFx-simple), the application does not show the expected customizations. For example, one or more of the following might occur:

- The background is not in the expected color.
- The user interface pages have a simple, minimal appearance instead of the expected skin.
- Expected images do not appear.

Solution

Verify that you used the correct target application version when you created the custom skin. Try repackaging and redeploying the JAR file and ensure that no problems occur during the packaging process.

Ensure that you copied the necessary JAR files to all the Oracle Fusion applications and that you spelled the name of the skin correctly in the profile option.

For more information, see Section 12.3, "Applying a Custom Skin to Your Oracle Fusion Applications."

A.3.4 Finding the EAR File for an Application

Use Oracle WebLogic Server Administration Console to locate a deployed application's enterprise archive (EAR) file.

To find the EAR file for a deployed application:

1. Make a note of the string that follows the host name in the URL of the deployed application. For example, if the URL is `http://myhost.mycompany.com:7401/myProd/faces/MyPage`, make a note of `myProd`.
2. Open Oracle WebLogic Server Administration Console.
3. Select **Deployments**.
4. In the **Overview** tab, click the entry with a name similar to the text that you noted in Step 1, such as `MyProdApp (V2.0)`, to display the settings for that deployed application.

5. In the **Overview** tab, find the **Path** setting to see the path to the EAR file.

A.4 Using My Oracle Support for Additional Troubleshooting Information

You can use My Oracle Support (formerly MetaLink) to help resolve Oracle Fusion Middleware problems. My Oracle Support contains several useful troubleshooting resources, such as:

- Knowledge base articles
- Community forums and discussions
- Patches and upgrades
- Certification information

Note: You can also use My Oracle Support to log a service request.

You can access My Oracle Support at https://support.oracle.com.

Glossary

application role

A **role** specific to applications and stored in the policy store.

application stripe

A collection of Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider policies that are applicable to the application with which it is associated. Out of the box, an application stripe maps to a Java EE application. Oracle Platform Security Services also supports mapping multiple Java EE applications to one application stripe.

artifact (SAR file)

A file included in the **SAR file** of the **SOA composite application**. Examples of artifacts include **binding components** and **service components**, references to B2B agreements, Oracle Web Services Manager (Oracle WSM) policies, **human task** flows, and metadata such as WSDL and XSD files.

binding component

A component that establishes the connection between a **SOA composite application** and the external world. There are two types of binding components:

- services: provide the outside world with an entry point to the SOA composite application
- references: enable messages to be sent from the SOA composite application to the external services in the outside world.

BPEL

Business Process Execution Language. An XML-based markup language for composing a set of discrete web services into an end-to-end process flow.

BPEL process

A **service component** that integrates a series of business activities and services into an end-to-end business process flow. See also **BPEL**.

bucketset

A container for defining a list of values or a range of values of a specified type. After you create a bucketset, you can associate the bucketset with a fact property of matching type. **Business rules** use the bucketsets that you define to specify constraints on the values associated with fact properties in rules or in **decision tables**.

business event

A message sent as the result of an occurrence or situation, such as a new order or completion of an order. You can raise business events when a situation of interest occurs. When an event is published, other applications can subscribe to it. Definitions for business events are stored in an Oracle Metadata Services (MDS) repository, and then published in the Event Delivery Network (EDN).

business object

A resource in an enterprise database, such as an invoice or purchase order.

Business Process Execution Language (BPEL)

See **BPEL**.

business rule

A statement that describes business policies or describes key business decisions.

component

An individual piece of an application, for example, a task flow, portlet, page, or layout element such as a box or image.

configuration plan

As you move projects from one environment to another (for example, from testing to production), you typically must modify several environment-specific values, such as JDBC connection strings, hostnames of various servers, and so on. Configuration plans enable you to modify these values using a single text (XML) file. During process deployment, the configuration plan searches the SOA project for values that must be replaced to adapt the project to the next target environment.

custom object

A high-level artifact, which manages data that resides in a database table, that you create using Oracle Fusion CRM Application Composer.

customize

To change a standard (existing) Oracle Fusion Applications artifact.

data dimension

A stripe of data accessed by a data **role**, such as the data controlled by a business unit.

data security

The control of access to data. Data security controls what action an end user can take against which data.

data stripe

A dimensional subset of the data granted by a **data security** policy and associated with a data **role**. The data dimension expresses stripes of data, such as territorial or geographic information, that you can use to partition enterprise data.

decision table

An alternative business rule format that is more compact and intuitive when many rules are needed to analyze many combinations of property values. You can use a decision table to create a set of rules that covers all combinations or when no two combinations conflict.

dependent value set

A list of values whose availability and meaning depend on the value that the end user provides for a prior **segment**, where the prior segment is associated with an **independent value set**.

descriptive flexfield

A type of **flexfield** used to give additional attributes to a data model. A descriptive flexfield can support only a set amount of **segments**.

design time customizations and extensions

Customizations and extensions that include more complex changes, such as creating Oracle SOA Suite composite applications or creating new batch jobs. Design time customizations are most often done by Java developers using Oracle JDeveloper (a comprehensive IDE), or may be done in other tools, such as Oracle SOA Composer. The customizations are then uploaded or deployed to a running instance of Oracle Fusion Applications.

domain value map

A set of value mappings that enables you to associate values from one application with values from another. For example, one value can represent a city with a long name (Boston), while another value can represent a city with a short name (BO). In such cases, you can directly map the values by using domain value maps.

entitlement

A set of grants of access to functions and data. This is an Oracle Fusion Middleware term for **privilege**.

extend

To create a completely new artifact, such as a custom **business object** or custom view page.

extensible flexfield

A type of **flexfield** that is similar to a **descriptive flexfield**, but does not have a fixed number of **segments**, allows grouping of segments into contexts, allows entities to inherit segments from their parents, and supports one-to-many relationships between an entity and its extended attribute rows.

flexfield

A set of placeholder fields (**segments**) that is associated with a business object. Oracle Fusion Applications provides three types of flexfields: descriptive, extensible, and key. Implementors use descriptive and extensible flexfields to add custom attributes to **business objects**. Implementors use key flexfields to define keys, such as part numbers.

flexfield sandbox

A **sandbox** to which you deploy **flexfield** configurations for testing purposes before deploying to the mainline code.

format-only value set

A **value set** that conforms to formatting rules. This is used when you want to allow end users to enter any value so long as that value conforms to formatting rules. For example, if you specify a maximum length of 3 and numeric-only, then users can enter

456 but not 4567 or 45A. Use a format-only value set only when no other types of validation are required.

function security

The mechanism by which user access to application functionality is controlled.

global layer

A customization layer in which customizations affect all end users of the application. This layer's XML files are added for everyone, whenever the artifact is requested. Customizations made to ADF Business Components in Oracle JDeveloper must be made in the global layer.

human task

A Business Catalog component that enables you to define how end users interact with your Oracle Business Process Management (Oracle BPM) processes. Human tasks are implemented in an Oracle BPM process using the user task. Human tasks are also used in **SOA composite applications**, where they are known as **service components**.

independent value set

A predefined list of values for a **flexfield segment**. These values can have an associated description. The meaning of a value in this **value set** does not depend on the value of any other segment.

key flexfield

A non-optional type of **flexfield**. This type of flexfield is used to define the parts of a key structure such as the parts of a product key or the parts of a customer key.

metadata sandbox

The type of **sandbox** that supports making changes to the application's metadata stored in an Oracle Metadata Services (MDS) repository.

multi-tenant environment

An environment where a single application instance serves multiple client organizations by partitioning the data and configurations into separate virtual application instances.

navigator menu

The global menu that is accessible from the Oracle Fusion Applications global area.

partner link

Characterizes the conversational relationship between two services in a **BPEL process** by defining the roles played by each service in the conversation and specifying the port type provided by each service to receive messages within the conversation.

permission

A security artifact in the policy store that maps a specific application resource with an allowed action. For example, a permission may be granted to a **role** to confer access rights to users.

personalization

The changes that every end user of the Oracle Fusion Applications product suite can make to certain artifacts in the user interface (UI) at runtime. These changes remain for that user each time that user logs in to the application. Personalization includes

changes based on user behavior (such as changing the width of a column in a table), changes the user elects to save, such as search parameters, or composer-based personalizations, where an end user can redesign aspects of a page.

privilege

A grant of access to functions and data. A privilege is a defined by a single, real world action on a single business object.

production-to-test movement

The process of completely refreshing the test environment by copying configuration and data from the production system.

resource bundle

A collection of locale-specific objects. When a program needs a locale-specific resource, a `String` for example, it can load the resource from the resource bundle appropriate for the current user's locale. In this way, the program code is largely independent of the user's locale, isolating most, if not all, of the locale-specific information in resource bundles.

role

An identity that determines permitted access to application functions and data.

rule dictionary

A **business rules** container for facts, functions, globals, **bucketsets**, links, decision functions, and **rulesets**. A dictionary is an XML file that stores the application's rulesets and the data model. Dictionaries can link to other dictionaries.

ruleset

A **business rules** container for rules and **decision tables**. A ruleset provides a namespace, similar to a Java package, for rules and decision tables.

runtime customizations and extensions

Customizations and extensions that can be made to Oracle Fusion Applications at runtime using browser-based components. These customizations and extensions are available to all or to a subset of Oracle Fusion Applications end users, and range from changing the look and feel of a page, to customizing standard **business objects**, adding a new business object and associated pages and application functionality, changing workflows, defining security for new objects, and customizing reports.

sandbox

A testing environment that separates sections of an application so that changes and modifications are kept within the sandbox and do not affect the mainline code or other sandboxes. Different users can create their own sandboxes to test their own sections. After the changes made in the sandbox have been tested, the sandbox user has the option to publish the changes to the mainline code.

SAR file

A SOA archive deployment unit. A SAR file is a special JAR file that requires a prefix of sca_. (For example, sca_OrderBookingComposite_rev1.0.jar). The SAR file packages **binding components** and **service components**, such as **BPEL processes**, **business rules**, **human tasks**, and mediator routing services, into a **SOA composite application**.

security reference implementation

Provides **role**-based access control in Oracle Fusion Applications, and is composed of predefined security policies that protect functions, data, and segregation of rules. The reference implementation supports identity management, access provisioning, and security enforcement across the tools, data transformations, access methods, and the information lifecycle of an enterprise.

security sandbox

The type of **sandbox** that supports making **data security** changes.

segment

A subdivision of a **flexfield**. A segment captures a single atomic value, which is represented in the application database as a single column. In the application UI, segments can be presented as individual table columns, as separate fields, or as a concatenated string of values.

service component

A component that implements the business logic or processing rules of a **SOA composite application**. Service components include **Oracle Business Process Execution Language processes**, **business rules**, **human tasks**, and mediator routing services.

site layer

A customization layer in which customizations affect end users at a particular location.

skin

A style sheet based on the CSS 3.0 syntax and that is specified in one place for an entire application. Instead of providing a style sheet for each component, or inserting a style sheet on each page, you can create one skin for the entire application.

SOA composite application

An assembly of service **binding components**, **service components**, and reference binding components designed and deployed in a single application. Wiring between the components enables message communication. The details for a composite are stored in the composite.xml file.

subset value set

A set of values taken from an existing **independent value set**. For example, if you have an independent **value set** for the days of the week, a weekend subset can be composed of its entries for Saturday and Sunday.

table value set

A **value set** with the Table validation type. The valid values in the value set are obtained from a specified column in a database table.

UI Shell template

The template that is use for the base UI for all Oracle Fusion Applications pages.

user layer

The customization layer in which all **personalizations** are made. The user layer is automatically selected when you use the Personalization menu.

value set

A list of values used to specify the validation rules for a **flexfield segment**.

wire

Wires connect service **binding components**, **service components**, and reference binding components into a complete **SOA composite application**.