Oracle® Fusion Middleware

Understanding Oracle Application Development Framework

12*c* (12.2.1.2.0)

E76683-01

October 2016

Documentation for Oracle Application Development Framework (Oracle ADF) developers that provides a conceptual and architectural overview of Oracle ADF.

ORACLE

Oracle Fusion Middleware Understanding Oracle Application Development Framework, 12*c* (12.2.1.2.0)

E76683-01

Primary Author: Ralph Gordon

Contributors: Steve Muench

Contents

Preface

Welcome to *Understanding Oracle Application Development Framework*.

Audience

This document is intended for enterprise developers who need an overview of the technologies encompassed by the Oracle Application Development Framework (Oracle ADF). This guide outlines the components and explains the concepts behind ADF Model, ADF Business Components, ADF Controller, ADF Faces, ADF Security, ADF Desktop Integration, and other parts of the ADF technology stack.

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

Related Documents

For more information, see the following documents:

- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Developing Web User Interfaces with Oracle ADF Faces*
- *Developing Applications with Oracle ADF Data Controls*
- *Developing Applications with Oracle JDeveloper*
- *Developing ADF Skins*
- *Administering Oracle ADF Applications*
- *Developing Applications with Oracle ADF Desktop Integration*
- *Tuning Performance*
- *High Availability Guide*

- *Installing Oracle JDeveloper*
- *Oracle JDeveloper Online Help*
- *Oracle JDeveloper Release Notes*, included with your JDeveloper installation, and on Oracle Technology Network
- *Java API Reference for Oracle ADF Model*
- *Java API Reference for Oracle ADF Controller*
- *Java API Reference for Oracle ADF Lifecycle*
- *Java API Reference for Oracle ADF Faces*
- *Java API Reference for Oracle ADF Data Visualization Components*
- *Java API Reference for Oracle ADF Share*
- *Java API Reference for Oracle ADF Model Tester*
- *Java API Reference for Oracle Generic Domains*
- *Java API Reference for Oracle Metadata Service (MDS)*
- *Tag Reference for Oracle ADF Faces*
- *Tag Reference for Oracle ADF Faces Skin Selectors*
- *Tag Reference for Oracle ADF Faces Data Visualization Tools*
- *Tag Reference for Oracle ADF Data Visualization Tools Skin Selectors*

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 features of Oracle JDeveloper and Oracle Application Development Framework (Oracle ADF) and other significant changes, which are described in this guide.

New and Changed Features for Release 12c (12.2.1.2.0)

For Release 12c (12.2.1.2.0), the guide has not been revised.

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

Part I

Introduction to Oracle ADF

Part I contains the following chapters:

1

Overview of Oracle ADF

This chapter provides a high-level overview of the architecture and components of Oracle Application Development Framework (Oracle ADF).

This chapter includes the following sections:

For definitions of unfamiliar terms found in this and other books, see the Glossary.

1.1 About Oracle ADF

Oracle Application Development Framework (Oracle ADF) is an end-to-end application framework that builds on Java EE standards and open-source technologies to simplify and accelerate implementing enterprise applications. Oracle ADF is suitable for enterprise developers who want to create applications that search, display, create, modify, and validate data using web, mobile, and desktop interfaces.

You can use the whole Oracle ADF framework to create an application, or you can use parts of the framework in combination with other technologies. Throughout this guide, applications that contain any ADF technologies are generally referred to as ADF applications. Web applications that incorporate ADF technologies throughout the business service, model, controller, and view layers are referred to as Fusion web applications.

1.2 Oracle ADF Key Concepts

Oracle ADF is based on the following concepts:

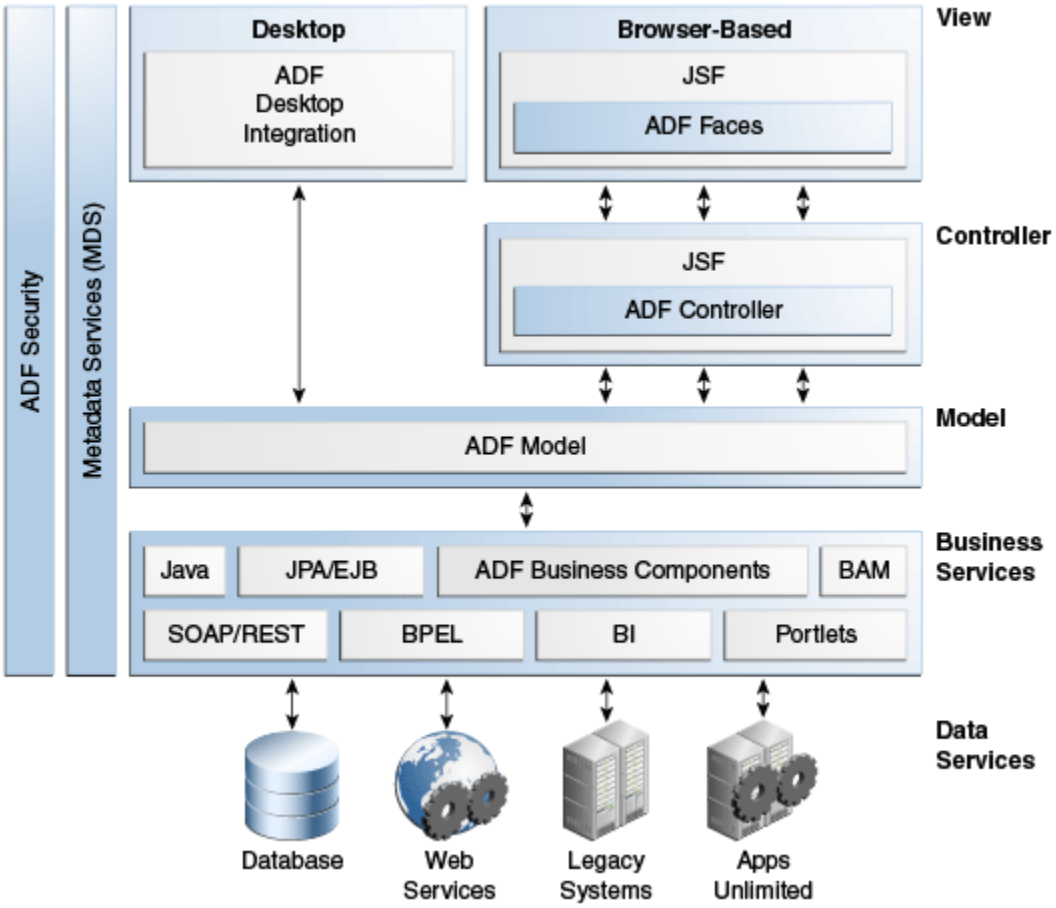
- Rich component sets for web, mobile, and desktop clients
- Declarative and reusable business logic and validation
- Declarative data binding
- Separation of UI-related and data-related elements (MVC architecture)
- Enhanced page flow functionality, including modular and reusable task flows
- Declarative security on ADF resources
- Customer level and developer level customization through metadata

For information on the key concepts of the broader Fusion Middleware stack, see Introduction to Oracle Fusion Middleware in *Understanding Oracle Fusion Middleware*.

1.3 Oracle ADF Key Components

This section provides a synopsis of the central high-level components in the ADF stack, including some of the underlying technologies such as JavaServer Faces (JSF) as well as other business and data services that are commonly part of an application. See Figure 1-1 for a visual depiction of the overall architecture, including the model, view, and controller (MVC) components.

Figure 1-1 ADF Architecture



1.3.1 ADF Model

ADF Model is a central part of Oracle ADF, enabling you to create ADF applications based on different types of business services. ADF Model implements data controls and data bindings. Data controls abstract the implementation technology of a business service by using standard metadata interfaces to describe the service's operations and data collections, including information about the properties, methods, and types involved. In Oracle JDeveloper, developers can view that information as icons that they can easily drag and drop onto a page. When the developer drags the representation of the service onto the page, Oracle JDeveloper automatically creates the bindings from the page to the services. At runtime, the ADF Model layer reads the information describing the application's data controls and data bindings from

appropriate XML files and implements the two-way connection between the user interface and the application's business service.

Oracle ADF provides ready-to-use data control implementations for common business service technologies, such as the following:

- ADF Business Components
- Enterprise JavaBeans (EJB) session beans and JPA Persistence API entities
- JavaBeans components
- Web services (SOAP and REST)

1.3.2 ADF Business Components

ADF Business Components are prebuilt application objects that are based on Java EE design patterns and best practices and which simplify the development and maintenance of complex, high-performance, and database-centric services.

When building service-oriented Java EE applications, developers implement the core business logic as one or more business services. These back-end services provide clients with a way to query, insert, update, and delete business data as required while enforcing appropriate business rules such as input validators. Using ADF Business Components, you can develop such services declaratively using wizards and visual editors in JDeveloper to generate the required metadata.

When you create an ADF Business Components application module, the services that it encapsulates are exposed through ADF Model as data controls, which you can then use to create databound components on web pages and other user interfaces. For the most common use cases, you can thus create a complete application without writing any Java code.

1.3.3 ADF Controller

In the controller layer of MVC applications, ADF Controller provides an enhanced navigation and state management model on top of JSF's controller layer. Using JDeveloper, you can declaratively create task flows that can manage application control between different types of activities, such as pages, methods on managed beans, declarative case statements, or calls to other task flows. In addition, you can create bounded task flows, which are reusable task flow segments that can be called from an overall task flow.

1.3.4 ADF Faces

ADF Faces provides the view layer for ADF applications. ADF Faces is a complete view framework that consists of over 150 Ajax-enabled JavaServer Faces (JSF) components, all built on top of the JSF standard. ADF Faces also can be used as a standalone component set that works with other non-ADF controller and model technologies.

1.3.5 ADF Mobile Browser

ADF Faces components supports recent and popular smart phones and tablets, such as Apple iPad, Samsung Galaxy S5 phones, and so on. To support older devices or feature phones that run basic HTML browsers, use the ADF Mobile browser technology. This technology uses Apache MyFaces Trinidad components to build the user interface.

This guide does not cover the ADF Mobile browser technology. For more information, see Overview of Oracle ADF Mobile Browser in *Developing Oracle ADF Mobile Browser Applications*. For information about supported mobile browsers, see "Certification Information" at http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html.

1.3.6 ADF Desktop Integration

ADF Desktop Integration enables developers to extend Fusion web applications so that end users can work with them using Microsoft Excel workbooks as a client.

1.3.7 ADF Security

The ADF Security framework uses and extends the Oracle Platform Security Services (OPSS) architecture to simplify the securing of ADF applications and enables fine-grained access control for ADF resources such as bounded task flows.

1.3.8 Oracle Metadata Services

The Oracle Metadata Services (MDS) framework allows you to create applications that your customers can further customize for their users or customers and which the end users can also customize without touching the source code or affecting the ability of the application to be patched or updated.

1.4 Key Management Tools and Processes

You can use Oracle ADF to develop components for broader middleware applications. For an overview of the tools and processes that you might use in such an application, see Understanding the Installation and Configuration Tools in *Understanding Oracle Fusion Middleware*.

To develop ADF applications, you use Oracle JDeveloper, which is an integrated development environment (IDE) that includes design-time support for ADF features. Among other things, JDeveloper contains wizards to generate working code for your business services, generates data binding code as you visually design your user interfaces, and provides a full testing and debugging environment. In addition, Oracle JDeveloper installations include a built-in copy of WebLogic Server, which enables you to test deploy your applications. For more information on using JDeveloper to develop ADF applications, see Introduction to Building Fusion Web Applications with Oracle ADF in *Developing Fusion Web Applications with Oracle Application Development Framework*. For information on JDeveloper features that are not specific to Oracle ADF, see Introduction to Oracle JDeveloper in *Developing Applications with Oracle JDeveloper*.

Note:

You can also develop ADF applications using Oracle Enterprise Pack for Eclipse (OEPE). OEPE is a set of plug-ins designed for the Eclipse IDE to support Java EE development. OEPE also includes support for ADF application development, though that support is more limited than that provided by JDeveloper. For more information, see http://www.oracle.com/technetwork/developer-tools/eclipse/overview/index.html.

For developing integrated Microsoft Excel workbooks with ADF Desktop Integration support, you need to install the ADF Desktop Integration add-in for JDeveloper. For

more information, see Installing ADF Desktop Integration in *Developing Applications with Oracle ADF Desktop Integration*.

For designing and modifying ADF skins, you can use the editors provided in JDeveloper. For more information, see Working with ADF Skins in JDeveloper in *Developing ADF Skins*.

Note:

You can also develop and deploy applications with a subset of Oracle ADF called Oracle ADF Essentials. Oracle ADF Essentials is a free packaging of key technologies from Oracle ADF that can be used to develop and deploy applications without licensing costs and to multiple application servers, such as GlassFish. For a list of the supported Oracle ADF Essentials features for GlassFish, go to the OTN site at `http://www.oracle.com/technetwork/developer-tools/adf/overview/adfessentials-1719844.html`.

2

Overview of the Oracle ADF Process Flow

This chapter contains an overview of the process for developing an application with Oracle ADF technologies and illustrates the process with a concrete use case.

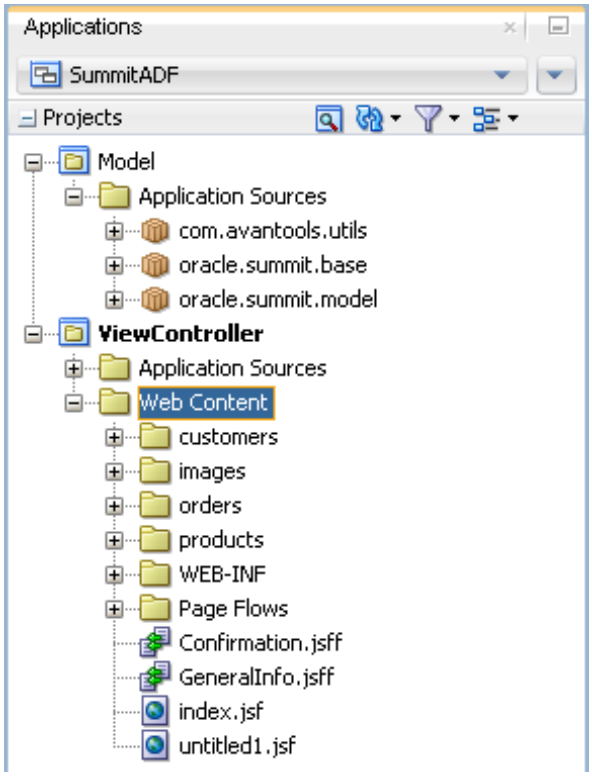
This chapter includes the following sections:

2.1 Oracle ADF High-Level Process Flow

At a high level, the development process for a Fusion web application usually involves a combination of the following steps:

- Creating an application workspace: Using a wizard, JDeveloper automatically adds the libraries and configuration needed for the technologies you select, and structures your application into projects with packages and directories.

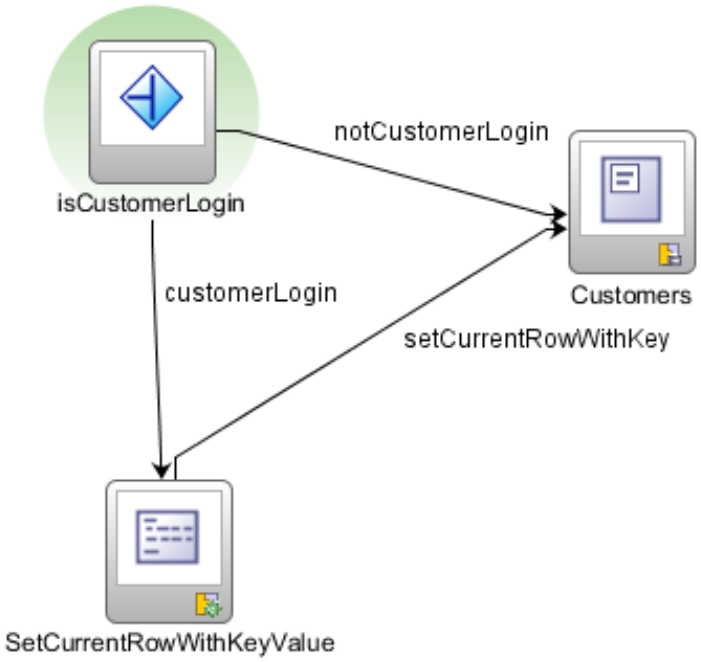
Figure 2-1 Applications Window in JDeveloper



- Obtaining a database schema or modeling the database objects from scratch: Within JDeveloper, you can create an offline replica of any database, and use JDeveloper editors and diagrammers to edit definitions and update schemas.
- Creating use cases: Using JDeveloper's UML modeler, you can create use cases for your application.
- Identifying shared resources: If you have components that can be used throughout your application or be used by multiple applications, you can develop and package those components as an ADF Library JAR and then have the JAR available in the Resources window for adding to your application.
- When using ADF Business Components, creating your own custom classes that extend the base framework classes and then configuring the model project to base any business components that you create on these custom classes: Even if you have no initial plans to put custom code into these custom classes, they provide a mechanism to later change base framework behavior and have those changes apply to all of the business components you have created in the application.
- Creating business services to access data and building a data model around those services: Based on your data source, you create business services using wizards or dialogs. Alternatively, you can create business services from scratch and then associate them with database tables later. You can also create services directly within a UML diagram.

  These business services can be ADF Business Components services, which are tightly integrated with ADF Model, which enables you to implement validation rules and other types of business logic declaratively and which enables you to easily bind data to UI components. Or they can be other types of services such as EJB session beans or web services, on top of which you can create ADF Model data controls to enable data binding and implement declarative business logic.
- Adding declarative logic to the business services, such as control hints and validation rules.
- Implementing the base user interface with ADF Faces, including page templates and layouts.
- Designing ADF task flows to define application control and navigation: You use diagrammers to visually determine navigation and control flow cases. JDeveloper creates the underlying XML for you. Figure 2-2 shows a task flow in the Summit sample application for ADF Task Flows that handles customer login.

Figure 2-2 ADF Task Flow



- Creating databound UI components: You can create databound components by dragging objects from the Data Controls panel onto a page and selecting the UI component you want to generate to display the underlying data. Figure 2-3 shows a pie chart that has been created by dropping a collection as a pie chart on a page fragment in the Summit sample application for ADF Task Flows. You can bind existing UI components to the data model (or change generated bindings) using JDeveloper's binding editors. Figure 2-4 shows the binding editor for the same pie chart. In both cases, JDeveloper generates the binding code.

Figure 2-3 Design-time View of Pie Chart

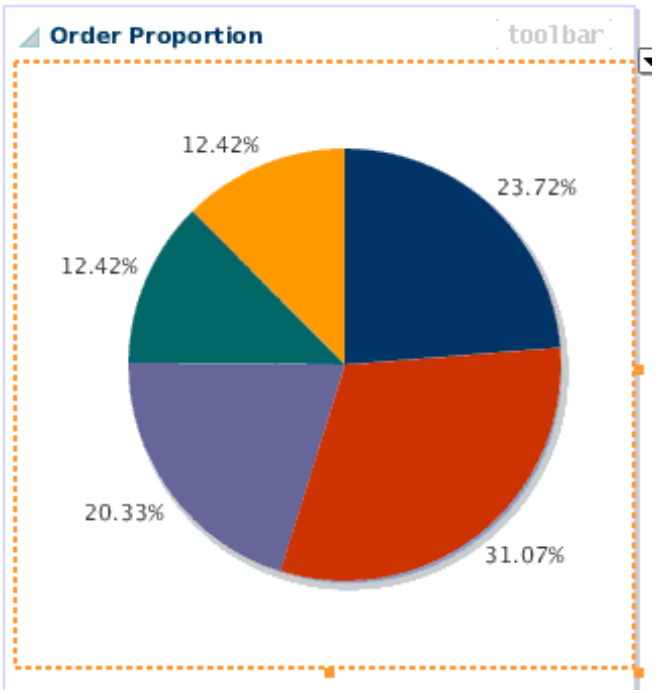
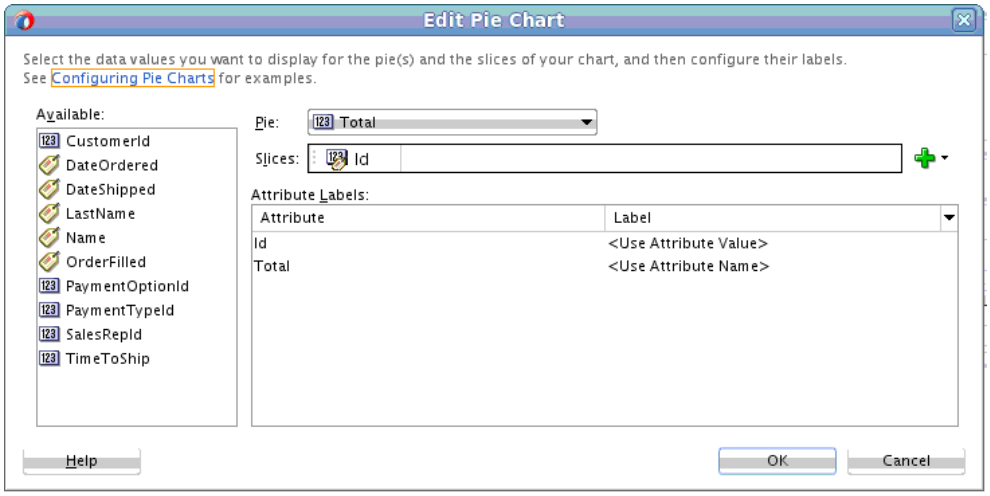
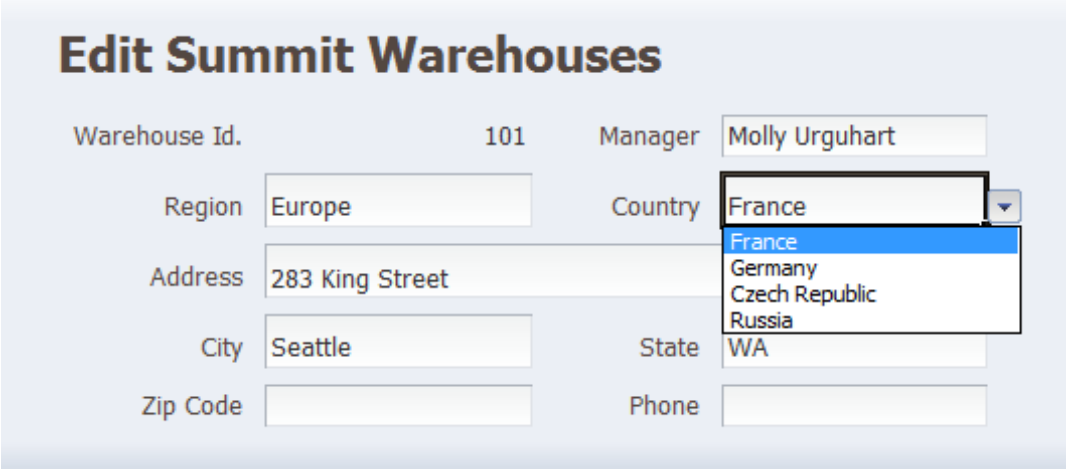


Figure 2-4 Binding Editor for Pie Chart



- Incorporating any page-specific validation and error handling: Once your application is created, you use editors to add additional validation and to define error handling.
- Securing the application: You use the Configure ADF Security wizard to enable base security features. Then you can use visual editors to define security policies for the application's resources, create roles, and populate the roles with test users.
- Enabling MDS customization of the application: If you want to enable your customers to provide further customization of the application for their users or to enable persistence of user changes to UI components within a session, you create customization classes and make them available to the application at design time and you enable seeded customization for any pages that you want to make customizable.
- Testing and debugging: JDeveloper includes an integrated application server that allows you to fully test your application without needing to package it up and deploy it to a standalone server. JDeveloper also includes the ADF Declarative Debugger, a tool that allows you to set breakpoints and examine the data.
- Refactoring: After testing your application (or even after having already deployed your application), you may need to rename components and reorganize code. JDeveloper has refactoring tools that simplify this work and protect against changes that would introduce bugs.
- Integrating the application with other applications in a service-oriented architecture (SOA): You can integrate your Fusion web application with an existing or new applications using SOA principals.
- Developing additional views: You can extend the application to provide functionality for other types of user interfaces, such as touch devices and desktop clients. Figure 2-5 shows an input form on an Excel worksheet from the Summit sample application for ADF Desktop Integration.

Figure 2-5 ADF Desktop Integration Form



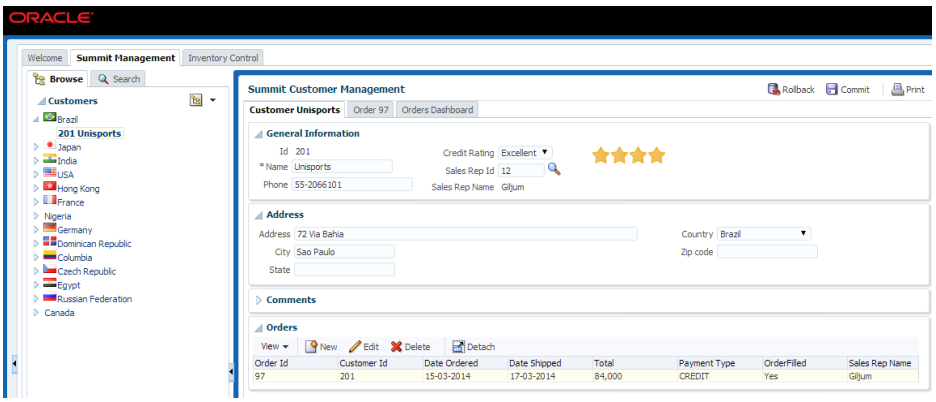
- Deploying the application: You use JDeveloper wizards and editors to create and edit deployment descriptors, JAR files, and application server connections. You can then use JDeveloper to deploy applications directly to a standalone application server or use other deployment tools.

2.2 Using Oracle ADF to Create a Rich Enterprise Application

This section walks you through the high-level steps of creating one of the Summit sample applications for Oracle ADF, which are a set of rich enterprise applications that are based on the ADF technology stack. The Summit sample applications support a sporting goods supplier with features for maintaining data on customers, customer orders, and products.

Figure 2-6 shows a page of the application showing an overview of one of its customers.

Figure 2-6 Summit Sample Application



The following are the high-level steps for creating the application:

1. Install an Oracle RDBMS on your local system or on a remote machine that you have access to.

   For information about the specific versions of Oracle Database that are supported, see "Certification Information" at `http://www.oracle.com/technetwork/developer-tools/jdev/documentation/index.html`.

2. Install the Summit schema in the database.

   Note:

   The Summit schema, along with an SQL script for installing the Summit database, was created within an JDeveloper application workspace using JDeveloper's offline database tools. You can install the database by running SQL scripts from the project for the schema in JDeveloper.

3. In JDeveloper, create an application workspace for the `Summit_Extensions` project. The output of this project is a library that contains some utility code related to calling database procedures that is later used in the Summit sample application and which might be useful for other applications as well.

4. In JDeveloper, create a new application workspace for the main application. In the New Gallery, select the **ADF Fusion Web Application** template and complete the steps in the ensuing wizard to create an application workspace that is configured with a model project to hold your ADF Business Components business services and a view-controller project to hold your ADF Faces pages and ADF Controller task flows.

5. In the application's model project, create stub custom classes that extend the ADF Business Components implementation classes (in the `oracle.jbo` package) to which extended functionality may eventually be added. Then, in the Project Properties dialog for the project, update the ADF Business Components Base Classes so that new business components that you generate are based on these custom classes. For more information, see Creating ADF Business Components Extension Classes in *Developing Fusion Web Applications with Oracle Application Development Framework*.

   This gives you the flexibility to later make changes to the base framework classes that can be picked up by their subclasses in your application. For example, the Summit sample application sub-classes `oracle.jbo.server.EntityImpl` with `SummitEntityImpl` and provides the `nextValSequence()` method, which enables entity objects to easily read the next value from a specified database sequence.

6. In the application's model project, create the ADF Business Components services that will comprise the Summit sample application's data model. These include entity objects, view objects, and application modules. You can do much of the work in the Create Business Components from Tables wizard, which also enables you generate entity associations based on foreign keys in your database tables, view links that describe relationship between view objects, and a UML diagram for the business components that you create.

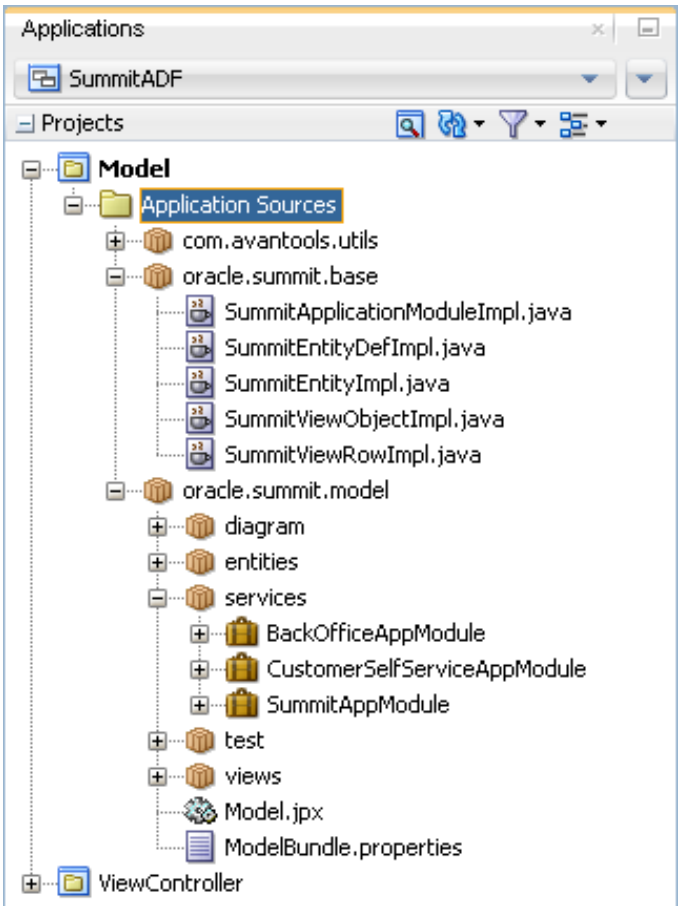
   You need entity objects for all of the database tables that you will use, view objects for SQL queries that the application needs to make, and application modules to aggregate the view objects required for a given user task. For example, the Summit sample application contains the following:

   - Entity objects for database tables that store information on customers, orders, products, warehouses, regions, countries, and so on.
   - Entity associations that correspond to each foreign key relationship between database tables that are represented by entity objects in the application.

- View objects, from which you can create databound UI components. Most of the view objects are based on columns in one or more entity objects, but a few of them are based on direct SQL queries to the database and are read-only.
- View links to describe the relationships between various view objects.
- The following three application modules:
  - `BackOfficeAppModule`, which encompasses the view objects that are needed to develop the functionality needed for the Summit employees to manage the customer and order database.
  - `CustomerSelfServiceAppModule`, which encompasses the view objects that are needed to develop the functionality needed for a customer application.
  - `SummitAppModule`, which nests the two other application modules and thus provides the services encompassed by those application modules should you later develop an application that needs all of those services.

Figure 2-7 shows the overall structure of the Summit sample application's data model project, including the extended implementation classes and the application modules.

Figure 2-7 Applications Window with Model Project

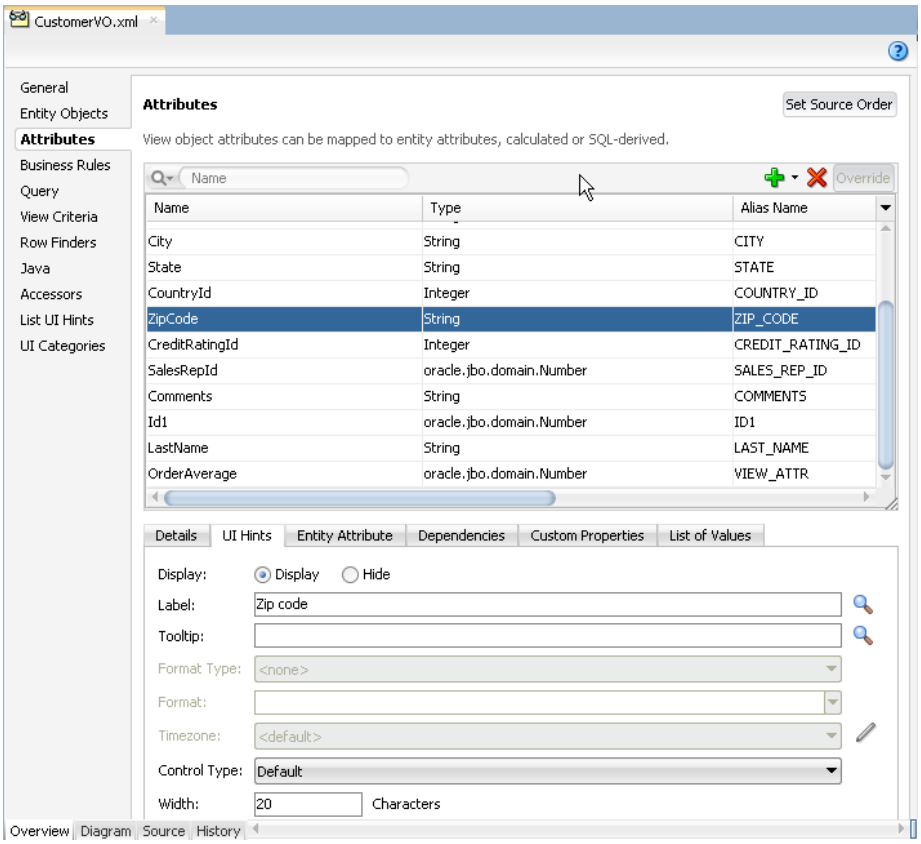


7. Add declarative logic to the business components so that the UI developer has to code less of this logic in the view layer and so that there are defaults that help

make the appearance and behavior of the generated UI components consistent among UI components generated from the same business components. For example, the Summit sample application includes the following types of declarative logic:

- UI hints for various view object attributes. For example, the `ZipCode` attribute of the `CustomerVO` view object is configured so that its default label is `Zip code`, as shown in Figure 2-8, and its default display width is 20 characters.
- Business rules that make sure that user input is valid. These rules come in the form of built-in rule types such as ranges and comparisons and in the form of expressions that you can insert yourself.
- List-of-value (LOV) objects that you can use to create selection lists and other UI components based on the contents of a database table or a static list. You can also use LOV objects to validate user input.
- Transient attributes, from which you can create components to display dynamically calculated values, such as the total price in the displayed list of order items.
- Default values for input components.
- Primary keys attributes that are set to use the `oracle.jbo.domain.DBSequence` data type so that the values of those attributes in new rows are obtained from database sequences. For example, the `Id` attribute of the `CustomerEO` entity object is set to `oracle.jbo.domain.DBSequence`, and its value is obtained from a database trigger that calls for the next value in the sequence when a new row is inserted into the database.

Figure 2-8 View Object Overview Editor



8. In the application's UI project, create the project's main page: The Summit sample application has an `index.jsf` page, which in turn is based on a `<af:pageTemplate>` component that defines the UI elements that establish the visual style for the web interface.

9. Create the task flows that define the application's flow: The Summit sample application contains an unbounded task flow that serves as the entry point to the application and bounded task flows to manage viewing and updating of customer data and managing orders.The Summit sample application includes the following task flows:

   - An unbounded task flow (represented by the `adfc-config.xml` file) that serves as the user's entry point to the application and that contains bounded task flows that demarcate the work flow for given tasks.
   - The `customer-task-flow-definition.xml` bounded task flow, which handles user navigation and management of customer information. This task flow also specifies a managed bean that provides some view-side logic for the controls.
   - The `create-edit-orders-task-flow-definition.xml` task flow, which enables a user to log new orders.

10. Create regions on the `index.jsf` page and populate them with databound UI components: This consists of the following sub-steps:

    a. Creating page fragments based on the task flow view activities.

b. Creating UI components by dragging and dropping objects from the Data Controls panel on to the page. You can create databound forms, tables, and trees, as well as DVT components such as gauges, charts, and carousels by dropping view objection collections. You can create buttons for standard navigation and CRUD commands by dragging and dropping standard operations that are built-in to the data controls by ADF Model.

c. Dragging the task flows on to the `index.jsf` page and dropping them as ADF regions.

Among others, the Summit sample application has the following fragments:

- `Customers.jsff`, which is embedded as a view activity within the `customer-task-flow-definition.xml` bounded task flow.
- `Orders.jsff`, which is embedded as a view activity within the `create-edit-orders-task-flow-definition.xml` bounded task flow.

11. Use the Configure ADF Security wizard to enable security for the application and establish roles and test users. In addition to the default roles, the Summit sample application contains roles for the application customer and the application employee, each of which is given resource grants appropriate for their roles.

Part II

Oracle ADF Back-end Components

Part II contains the following chapters:

3

ADF Model

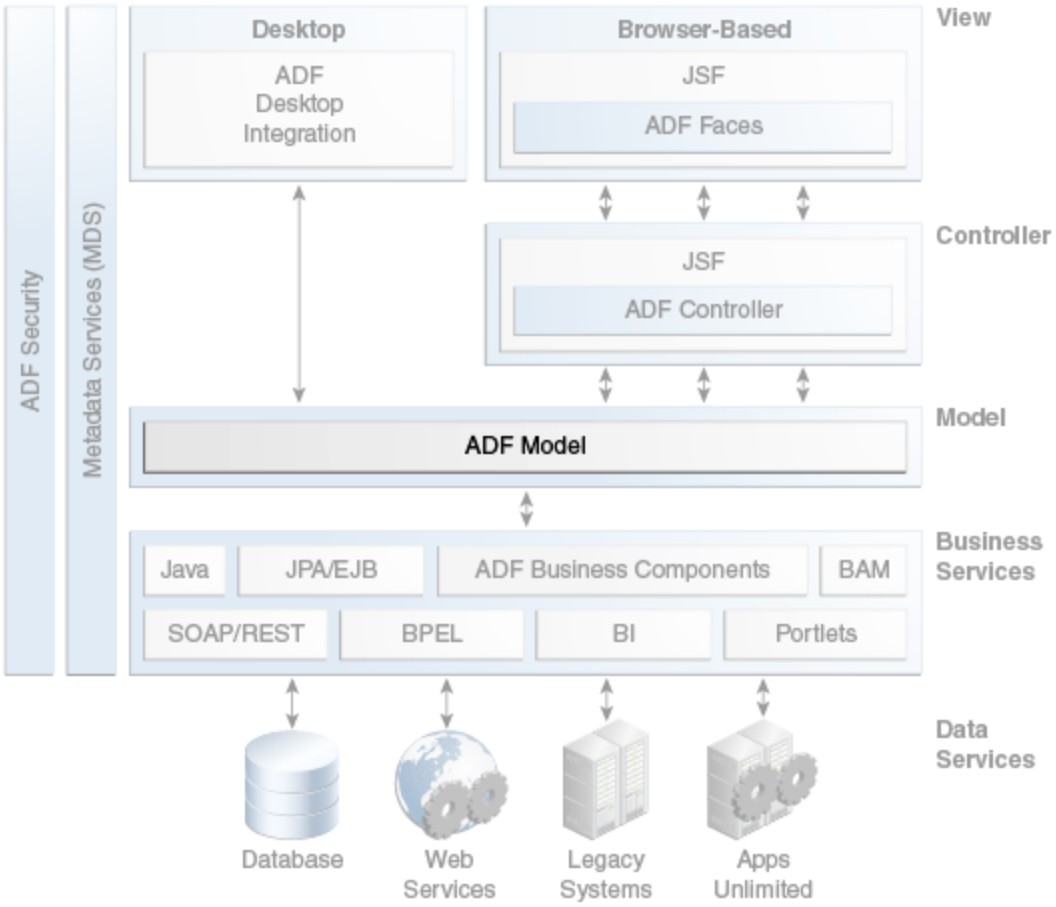
This chapter provides a high-level overview of the ADF Model technology, including data controls, declarative bindings, and the ADF binding context, and shows how those components work with other parts of a rich enterprise application.

This chapter includes the following sections:

3.1 About ADF Model

ADF Model is a declarative framework that provides an abstraction layer between business services and the view and controller layers of an enterprise application that standardizes the way that components in those layers interact with each other. Figure 3-1 shows how ADF Model fits into an enterprise application.

Figure 3-1 Overall Application Architecture with ADF Model



3.2 Core Benefits of ADF Model

ADF Model has features that benefit both UI developers and developers of application logic.

For UI developers, the core benefits are:

- Using JDeveloper, drag-and-drop creation of databound components and configuration of bindings with visual editors.
- Built-in record navigation and CRUD operations that you can add to your user interface.
- Standard use of JSF Expression Language (EL) for binding, without needing to understand the implementation of the underlying business services.
- The ability to bind business services to a variety of user interfaces, including ADF Faces components, JSF and JSP pages, and Microsoft Excel spreadsheets (through ADF Desktop Integration).

For developers who are focused on the application logic and data model, the core benefits are:

- Not needing to write Java code to handle standard interactions between the business services and the view layer.

- The ability to declaratively add validation rules, UI hints, default attribute values, and other business logic to your business services as metadata without changing the code in the business services themselves. These declarative enhancements to the data model are propagated to any components that are created from the data control.
- The ability to work with multiple types of business services in the same way. There are ADF Model data controls for ADF Business Components, EJB session beans, plain Java objects, web services, and other types of services.

3.3 Key Concepts of ADF Model

ADF Model consists of the following central features:

- *data controls*, which abstract the implementation technology of a business service by using standard metadata interfaces to describe the service's operations and data collections, including information about the properties, methods, and types involved.
- *declarative bindings*, which are used to bind services that are exposed by data controls to UI components.

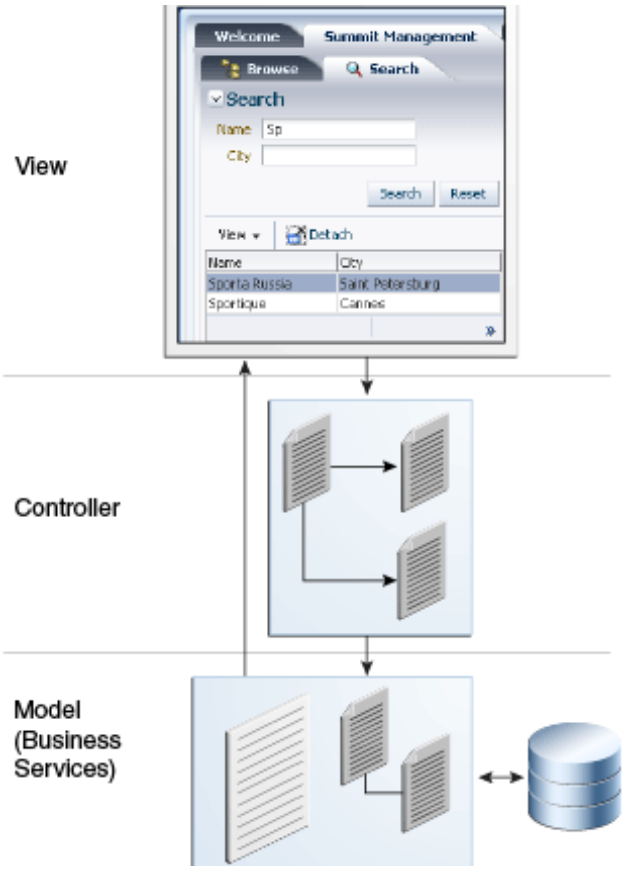
This section provides an overview of how data controls and declarative bindings work within ADF applications.

3.3.1 Abstraction of the Application's Model Layer

ADF Model builds upon the MVC (model-view-controller) design pattern, in which the code for the application's data model, visual interface, and application flow are all separated. This separation enables multiple types of client displays to work with the same business information. It also helps delineate the responsibilities of the developers working on the different layers of the application.

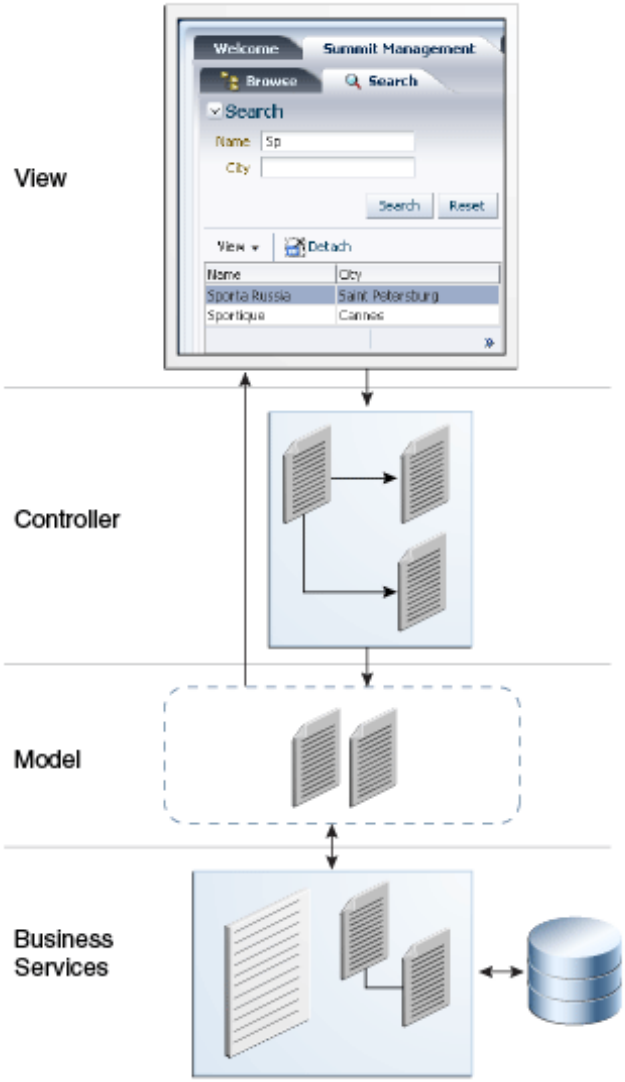
In a basic MVC architecture, the model layer consists of business services, which in turn interact with the data, as shown in Figure 3-2. With this architecture and without the help of a framework, you would need to code the business services, the controller, and the view components to properly interact with each other.

Figure 3-2 Basic MVC Architecture



When you use ADF Model in your application, you save yourself from having to write the Java code that would otherwise be necessary to coordinate the MVC layers. Similarly, the implementation details of the business services are no longer a concern to the UI developer. As shown in Figure 3-3, ADF Model serves as a conduit between the business service and the view and controller parts of the application and presents a standard way of creating bindings between the view and business services.

Figure 3-3 MVC Architecture with ADF Model



3.3.2 Declarative Data Binding

In JDeveloper, you can create declarative bindings between data services and UI components by dragging items from the Data Controls panel on to the visual editor for the given UI technology, such as a JSF page. The declarative bindings coordinate between the data controls and the controller and UI layers.

ADF data binding extends JSF data binding by enabling you to bind to ADF data controls declaratively. In a typical JSF application, you would create managed beans and then create EL expression references to them in the JSF page code. However, in an application that uses ADF Model, you can use XML configuration files instead of managed beans. Binding code in these XML files and the EL expressions in the JSF pages are automatically generated when you drag objects from the Data Controls panel on to a page. You can also manually add, delete, and modify bindings in these files.

3.4 Key Components of ADF Model

ADF Model consists of the following components

- Data controls
- Declarative bindings

3.4.1 Data Controls

There are different types of data controls, depending on what type of business service you are using. The main types of data controls are:

- Adapter data controls for common business services such as EJBs, plain Java classes (POJO), SOAP-based web services, and RESTful web services
- Data controls based on ADF Business Components application modules

3.4.1.1 Adapter Data Controls

Adapter data controls, as the name implies, act as adapters for non-ADF business services that expose their interfaces in a standard way to the binding layer.

Adapter data controls are available in JDeveloper for the following types of services:

- EJB session bean
- Bean (plain Java object)
- Web service (SOAP-based and REST-based)
- URL service
- JMX

There are also placeholder data controls, which enable a UI developer to mock up a data control for purposes of creating databound UI components before the actual business services are available.

Once they have created a data control, developers also have the option of creating data control structure files for the individual services that are encompassed by the data control in order to configure them with declarative metadata. The configuration possibilities in these structure files mirror the type of declarative configuration that you can do with ADF Business Components view objects.

The following are some of the types of metadata that can be added to adapter data controls:

- Default values of attributes.
- Transient attributes.
- UI hints for attributes.
- Validation rules (including templates for comparison operations, range, length, and the opportunity to write rules based on regular or Groovy expressions).
- Named criteria, which can be used to create UI search components based on pre-selected partial search criteria. Named criteria are only available for JPA-based adapter data controls.
- List-of-values (LOV) components, which enable UI developers to create list components that are populated by a given table in the data source.

When you create an adapter data control, a data control definition file with the name `DataControls.dcx` is created. If you subsequently add declarative metadata for objects represented by the data control, XML files are generated to hold that metadata.

3.4.1.2 ADF Business Components

ADF Business Components services are directly integrated with ADF Model. A data control in an ADF Business Components application derives from the data model that you set in one or more application modules. View objects encapsulated by the application module represent the business services and can be configured both declaratively and programmatically. The types of declarative metadata available for a view object include all of the types of metadata available for adapter data controls as well as other metadata specific to view objects.

For more information on application modules and view objects, see Key Components of ADF Business Components.

3.4.2 Declarative Bindings

Declarative bindings provide a way to call from the view layer into the model layer using EL expressions or Java code.

The following are the three categories of bindings:

- *value bindings*, which are used by UI components to display data. There are sub-types of value binding objects for trees, lists, and other components.
- *action bindings*, which are used to bind buttons and links to service methods and operations.
- *executable bindings*, which include *iterator bindings* and which generally concern background tasks, such as managing queries and row currency.

These binding objects are defined in page definition files, which are created and updated automatically when you use the Data Controls panel to create databound components. By default, a generated page definition file name takes the name of its corresponding web page, appends `pageDef`, and takes the `xml` extension. There is one page definition file for each page.

The binding objects for each page reference data control objects to provide the UI components with data. At runtime they are instantiated in a *binding container*. A page can access its corresponding binding container using expressions based on the EL namespace `bindings`. Such expressions always evaluate to the binding container for the current page. A typical expression takes the form `#{bindings.`*`BindingObject`*`.`*`propertyName`*`}` where *`BindingObject`* refers to an object or attribute defined in the page definition file and *`propertyName`* refers to a standard ADF binding property. For example, `#{bindings.Phone.inputValue}` would return the value of the `Phone` attribute.

Together, all of an application's binding containers and their references to data controls form the application's *binding context*. This binding context is represented at design time by the `DataBindings.cpx` file, which is located in an application's UI project.

3.5 ADF Model at Runtime

At runtime, the ADF Model layer does the following:

- Reads the `DataBindings.cpx` file to set up the binding context based on the listed page definition files and the data controls that they are mapped to as depicted in Figure 3-4.
- Instantiates the bindings in order to create the two-way connection between the user interface and the business services.

Figure 3-4 Files Used in Data Binding

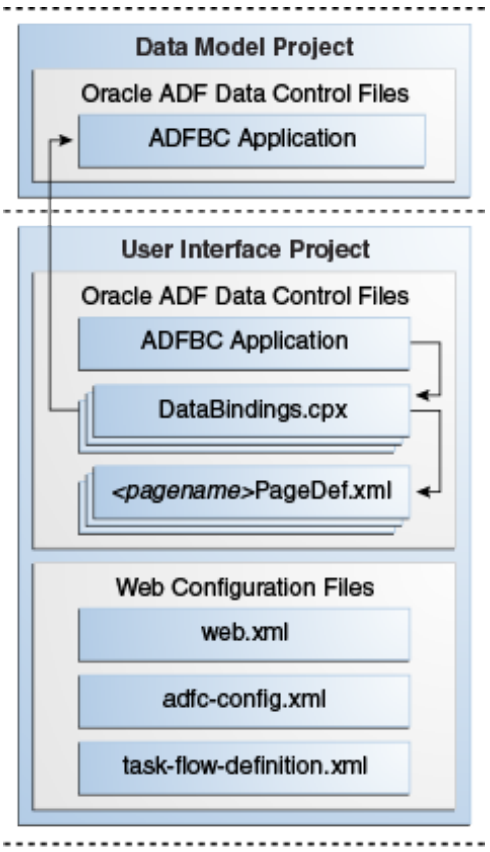
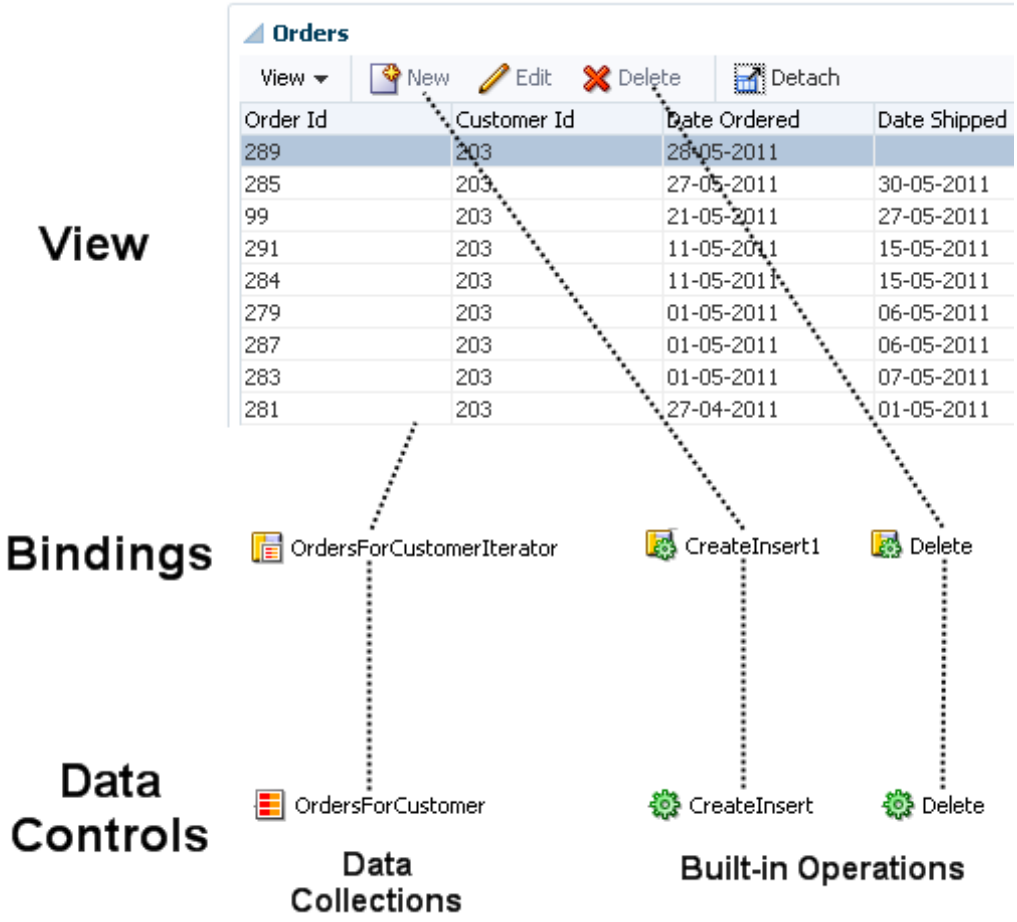


Figure 3-5 depicts the connection between the data controls, declarative bindings, and view layer.

Figure 3-5 Binding Data Sources to UI Components Using ADF Model



3.6 Overview of the ADF Model Process Flow

This section describes the process flow for working with ADF Model. The process differs depending on whether you are using ADF Business Components or other types of data controls.

3.6.1 Development Steps for Using ADF Model with ADF Business Components

Using ADF Model with ADF Business Components simply consists of building the business components. For more information, see Overview of the ADF Business Components Process Flow.

3.6.2 Development Steps for Using ADF Model with non-ADF Business Services

Using ADF Model with non-ADF business services consists of the following basic steps:

1. In JDeveloper, create an application workspace for the application.
2. In the application workspace, create or import the business services on which the data control will be based.

   These services might be EJB session facades, plain Java objects (POJOs), web services, or other types of service for which you have a data control. To create the services, you might also need to a connection to a database, URL, or a schema.

3. Using a wizard that is available in JDeveloper's New Gallery, create data controls for the business services.
4. Optionally, use JDeveloper's visual editors to declaratively specify business rules for the services that are encapsulated by the data controls.
5. Optionally, for JPA-based data controls, use the ADF Model Tester to test the business services.
6. Using the Data Controls panel and various binding editors, create databound components in the view layer.

3.7 Learning More About ADF Model

The following resources provide detailed information about using ADF Model in applications:

- For information on creating and configuring adapter data controls, see Introduction to ADF Model in *Developing Applications with Oracle ADF Data Controls*.
- For information on creating a data model using ADF Business Components, see Implementing Business Services with Application Modules in *Developing Fusion Web Applications with Oracle Application Development Framework*.
- For information on creating user interfaces that use ADF Model data binding, see Creating a Databound Web User Interface in *Developing Fusion Web Applications with Oracle Application Development Framework*.
- For Javadoc documentation related to data controls and data binding, see the *Java API Reference for Oracle ADF Model*.

4

ADF Business Components

This chapter provides a high-level overview of ADF Business Components, including a description of the key features they provide for building your business services. Features described include entity objects, view objects, and application modules.

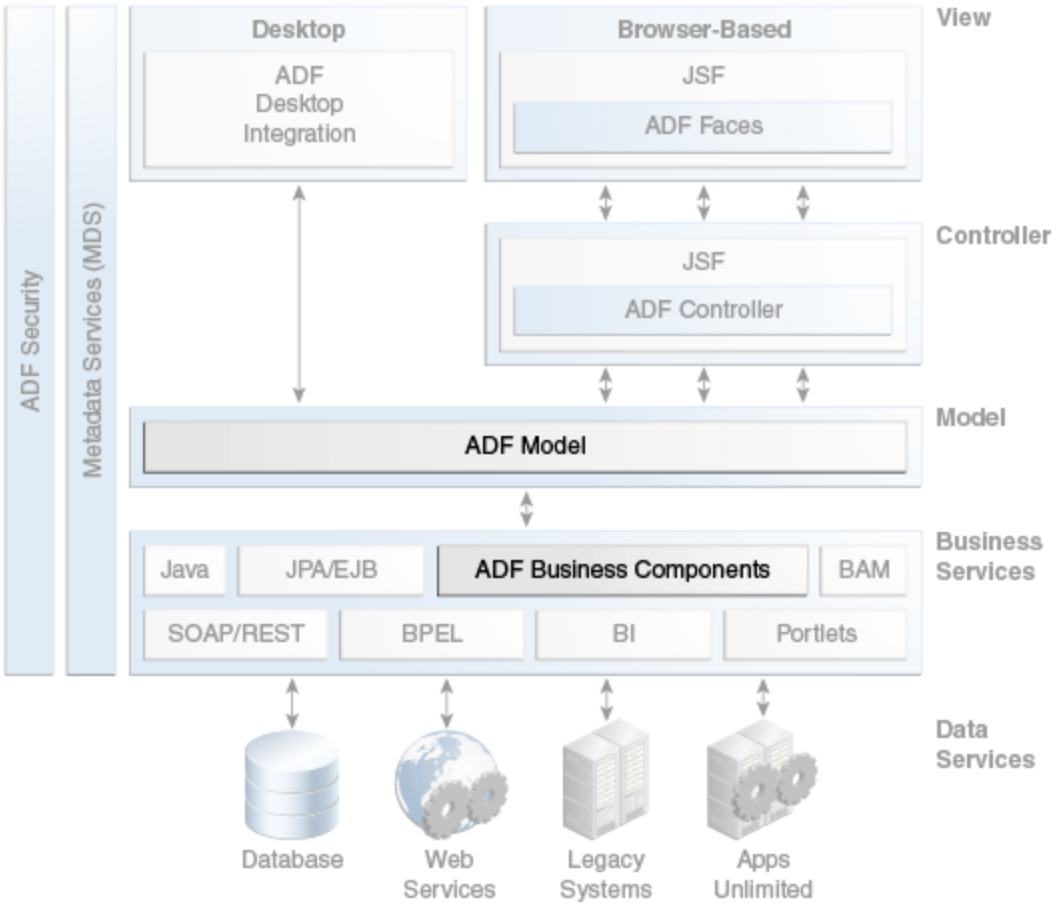
This chapter includes the following sections:

4.1 About ADF Business Components

ADF Business Components is a technology to create reusable data-aware business services with minimal developer coding. Developers can use wizards and visual editors to create ADF Business Components services without writing any Java code. It is also possible to extend the core ADF Business Components classes to create more advanced functionality. ADF Business Components services are exposed through ADF Model for use by the application's view layer.

Figure 4-1 shows how ADF Business Components fit into the ADF technology stack. Note that ADF Business Components features directly integrate with ADF Model.

Figure 4-1 ADF Architecture with Business Components



In addition, you can expose applications that you create with ADF Business Components as services that can be consumed by other Fusion web applications, composite applications that adhere to the Service Component Architecture (SCA), and other applications via web service calls.

4.2 Core Benefits of ADF Business Components

ADF Business Components provides the following benefits for developers of business services:

- Management of database access, including connection, data retrieval, locking of records, and insertion and update of records.
- Ability to create data models that are tailored for specific types of end users, with only the necessary data exposed.
- Creating of data model relationships in addition to those defined by the database.
- Ability to use declarative rules to enforce required fields, primary key uniqueness, data precision-scale, and foreign key references.
- Capturing and enforcing both simple and complex business rules, programmatically or declaratively, with multilevel validation support.
- Implementing end-user Query-by-Example data filtering without code.

- Ability to expose components as services that can be integrated with other Fusion web applications and consumed by SOA composite applications.
- Ability to raise business events to launch business processes and trigger synchronization of external systems.
- A built-in facility for application state management that enables application failover and the handling of user sessions over multiple nodes in clustered and high availability server environments.
- Features geared toward performance optimization, such as shared application modules to handle static data and application module pooling.
- Wizards and visual editors in JDeveloper that generate XML definitions for the components that you can also edit manually.

4.3 Key Concepts of ADF Business Components

ADF Business Components is an implementation of Java-based business services that directly incorporate ADF Model. This section provides an overview of the role of business services and how ADF Business Components implements business services.

4.3.1 Implementation of Business Services

Business services are behind-the-scenes components that mediate between an MVC application and a data source (usually a database). In general, business services are responsible for the following tasks:

- Retrieving data requested by the rest of the application
- Representing this data as Java objects usable by the rest of the application (object-relational ["O/R"] mapping)
- Persisting changes made by the rest of the application
- Implementing business rules, such as validation logic, calculated attributes, and defaulting logic
- Providing services that can perform large-scale batch operations on data upon request

Business services segregate the persistence and business logic of an application from the logic that governs the application's UI and control flow. Keeping persistence and business logic separate allows you to reuse them in multiple MVC applications.

4.3.2 Based on Standard Java and XML

ADF Business Components is a framework implemented in Java. Base framework classes provide generic, metadata-driven functionality. XML files store metadata that you define to configure each component's runtime behavior. You can also extend the base framework functionality to suit your needs.

Figure 4-2 shows the Applications window in JDeveloper and how it represents the files that comprise ADF Business Components services. For example, the `DeptVO` component is defined with a single XML file that relies entirely on underlying framework classes. On the other hand, the `CustomerVO` definition consists of an XML definition file that provides metadata and three Java classes that extend the underlying framework classes.

Figure 4-2 XML and Java Objects for ADF Business Components

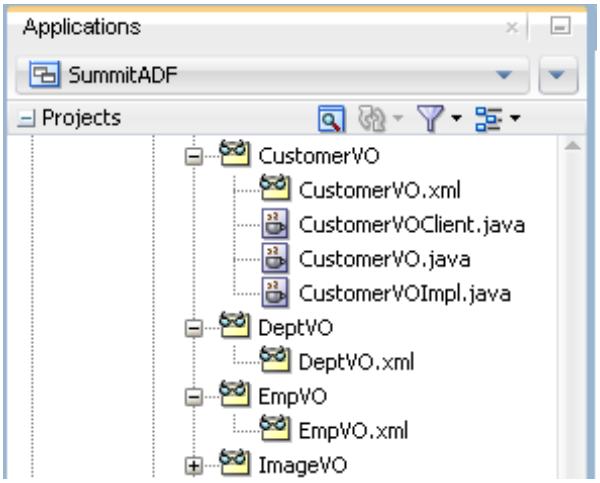
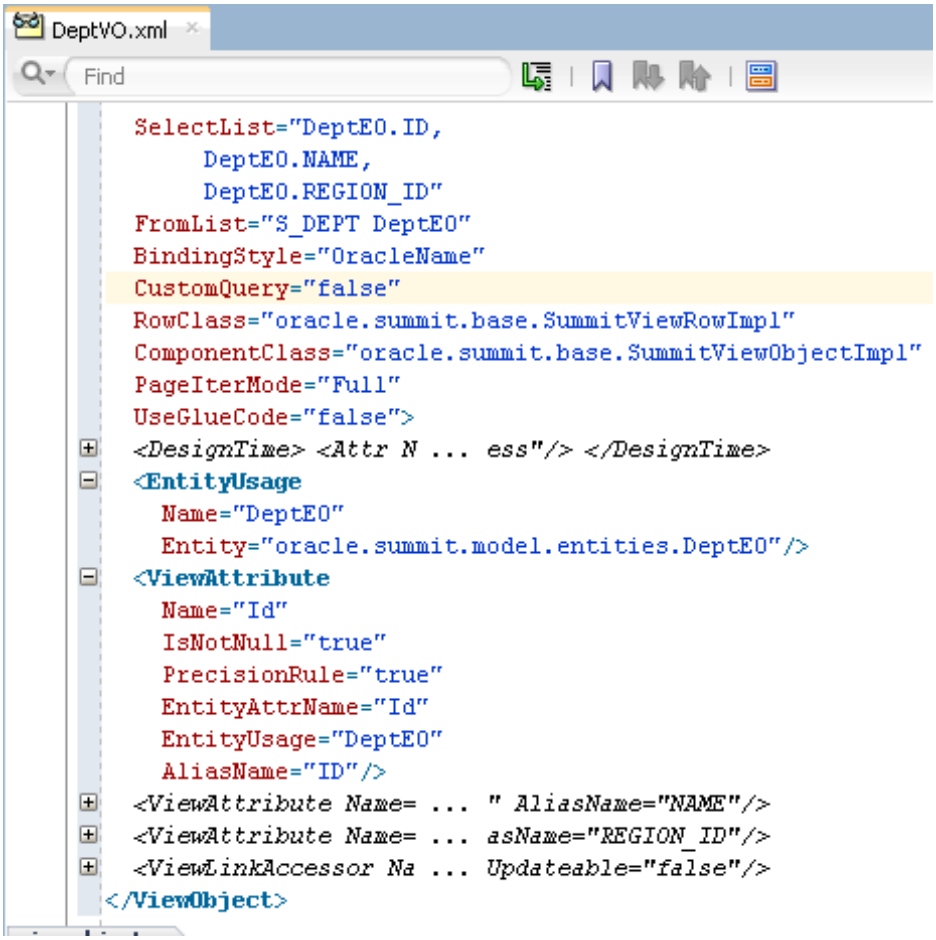


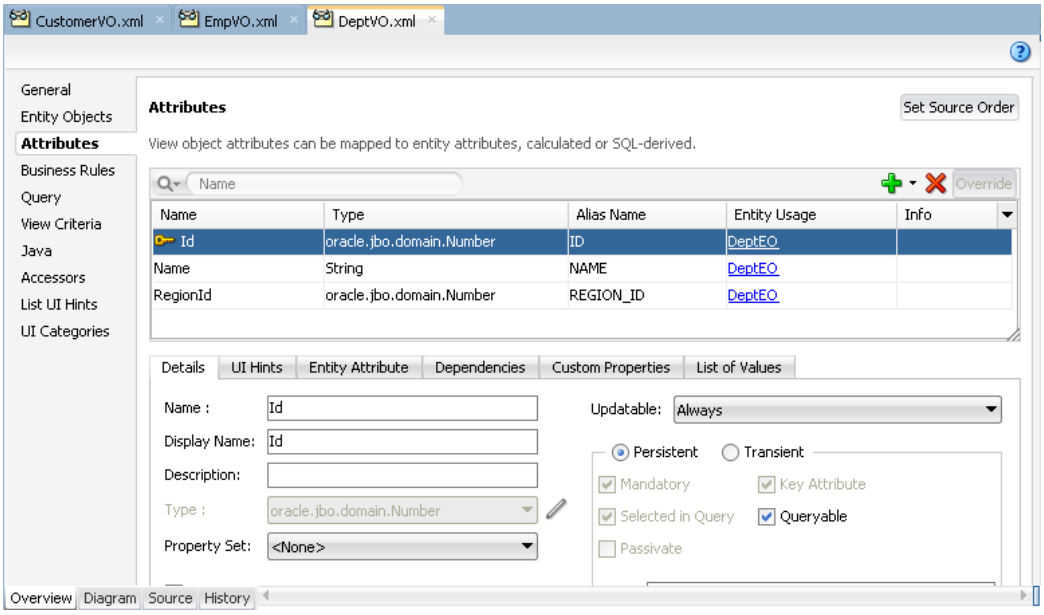
Figure 4-3 shows the source editor for an ADF Business Components view object definition file.

Figure 4-3 Source View for an ADF Business Components Definition File



JDeveloper also provides visual overview editors for ADF Business Components definition files. Figure 4-4 shows the overview editor for the same view object definition file shown in Figure 4-3.

Figure 4-4 Overview Editor for an ADF Business Components Definition File



4.3.3 Application Server and Database Independence

Because ADF Business Components services are implemented using plain Java classes and XML files, applications and services built using ADF Business Components can run on any Java-capable application server, including any Java EE-compliant application server. These applications and services can be run both within and outside of a Java EE server container.

You can use ADF Business Components components with both Oracle and non-Oracle databases. Numerous optimizations are built into ADF Business Components for use with Oracle databases.

4.3.4 Support for Java EE Design Patterns

ADF Business Components implements the Java EE design patterns that you would normally need to understand, implement, and debug yourself to create a real-world enterprise Java EE application. These patterns solve problems such as clean separation of application layers, efficient database access, and application scalability.

To cross-reference the names of these design patterns from the Java EE specifications with their ADF Business Components counterparts, you can refer to ADF Business Components Java EE Design Pattern Catalog in *Developing Fusion Web Applications with Oracle Application Development Framework*.

4.3.5 Declarative Metadata for Implementation Classes

ADF Business Components objects are based on a set of Java classes that provide built-in runtime functionality that you control through declarative settings. You use an XML component definition file to specify metadata for things like object/relation mapping for database tables, data access methods, and validation rules. At runtime, the metadata is injected into the implementation classes to create instances of the services. For typical use cases, developers do not have to write any Java code to implement the services.

4.3.6 Optional Custom Java Code

It is possible to further configure the behavior of a component by adding custom Java code to the component's definition. When you need to write custom code for a component, for example to augment the component's behavior, you can enable an optional custom Java class for the component in question.

4.3.7 Ability to Expose Services to SOA Applications

After you have developed ADF Business Components services, you can publish them as external services that can be consumed by applications that are based on a service-oriented architecture (SOA). For more information, see Service-enabled Application Modules.

4.3.8 Application State Management

ADF Business Components has a state management facility for application modules that enables you to save the state of a user session, which simplifies recovery and failover scenarios.

For more information on application module state management, see Application State Management. For more information on save points, see Save Points.

4.4 Key Components of ADF Business Components

The ADF Business Components architecture consists of the following key components:

- *Entity objects*, which encapsulate individual objects in a data source, such as tables in a database, and which add business logic for working with that data.
- *Entity associations*, which define the relationships between individual entity objects.
- *View objects*, which provide access to data in a form that can be used through ADF Model bindings in a user interface. View objects that allow updating of data are based on entity objects.
- *View links*, which define master-detail hierarchies between view objects.
- *Application modules*, which encapsulates the view objects needed for a logical unit of work related to an end-user task.

4.4.1 Entity Objects

ADF entity objects are business components that encapsulate data, persistence behavior, and business rules for items that are used in your application. For example, entity objects can represent:

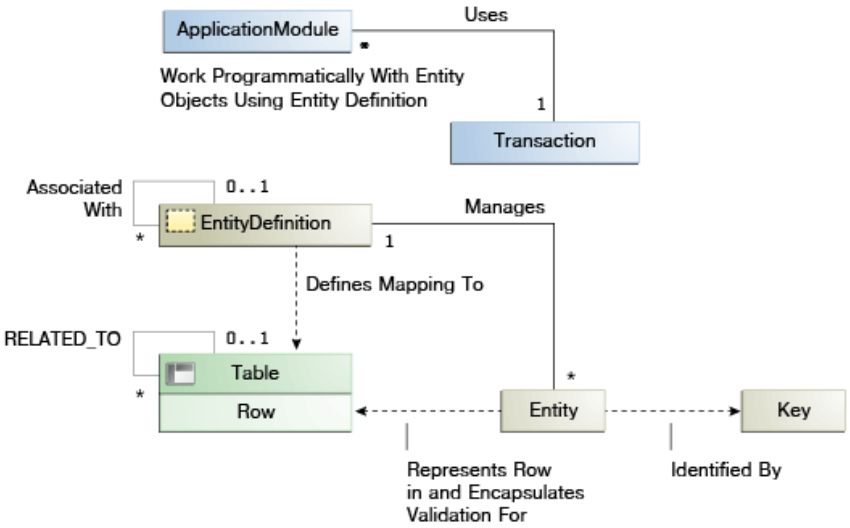
- Elements of the logical structure of the business, such as product lines, departments, sales, and regions
- Business documents, such as invoices, change orders, and service requests
- Physical items, such as warehouses, employees, and equipment

Entity objects map to single objects in the data source. In the vast majority of cases, these are tables, views, synonyms, or snapshots in a database. For example, you might create an entity object called `Departments` that represents a database table called

DEPARTMENTS. Advanced programmers can base entity objects on objects from other data sources, such as spreadsheets, XML files, or flat text files.

Figure 4-5 shows how an entity object fits in with other objects in an ADF Business Components application.

Figure 4-5 Entity Object Within the ADF Business Component Architecture



4.4.1.1 Entity Object Definition Files

When you use JDeveloper's wizards and visual editors to create and configure an entity object, JDeveloper creates an XML file that contains the declarative metadata that defines the runtime behavior of that entity object, including its O/R mapping, validation rules, UI hints, and other metadata. At runtime, this metadata is injected into an instance of the generic framework class `oracle.jbo.server.EntityImpl`.

It is also possible to add custom functionality to an entity object by writing custom classes that extend ADF Business Components framework classes. For information, see Generating Custom Java Classes for an Entity Object in *Developing Fusion Web Applications with Oracle Application Development Framework*.

4.4.1.2 Ways to Configure Entity Objects

Entity objects are part of ADF Business Components implementation of ADF Model. As such, you can add declarative metadata to an entity object definition to configure its behavior. The following are some of the things for which you can set metadata on an entity object:

- UI hints, which are settings that the view layer can use to automatically display the queried information to the user in a consistent, locale-sensitive way.
- Validation rules, which you can set at both the level of entity objects or individual attributes.
- Business events, which you can use to launch business processes and trigger external systems synchronization.

4.4.2 Entity Associations

Relationships between entity object definitions are handled by entity associations, which define a relationship between two entity object definitions based on sets of

entity attributes from each. Associations map to relationships between single objects in the data source. In the vast majority of cases, these are relationships among tables, views, synonyms, and snapshots in a database. Advanced programmers can use associations to represent relationships within other data sources, such as spreadsheets, XML files, or flat text files.

When the data source is a database, associations often map to foreign key relationships between tables in the database. Although there does not need to be a foreign key constraint between tables for you to create a one-to-one or one-to-many association between the corresponding entity objects, there should be an appropriate logical relationship between the tables.

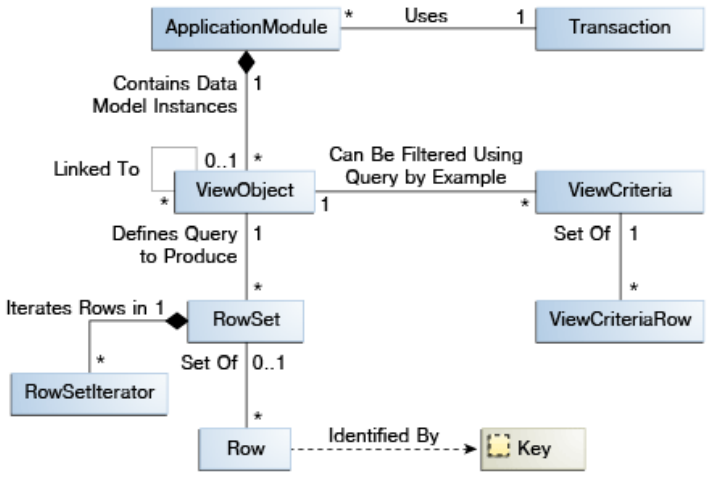
4.4.3 View Objects

ADF view objects are business components that collect data from the data source, shape that data for use by clients, and allow clients to change that data in the ADF Business Components cache. Among other things, a view object definition can gather the information needed to:

- Populate a single table element in a form
- Create and process an insert or edit form
- Create a list of values for populating a dropdown list
- Create a search form with specific search criteria

Once you have created a view object definition and included it in the data model of an application module, you use the Data Controls panel to create UI components based on the collections, attributes, and operations of that view object.

Figure 4-6 View Object Within the ADF Business Component Architecture



View object definitions must have a mechanism for retrieving data from the data source. Usually, the data source is a database, and the mechanism is a SQL query. ADF Business Components can automatically use JDBC to pass a query to the database and receive the result. When view object definitions use a SQL query, query columns map to view attributes in the view object definition. The definitions of these attributes reflect the properties of these columns, such as the columns' data types and precision and scale specifications. When view object definitions use other data sources, view object attributes map to "columns" of data from those data sources, as defined by the programmer.

Typically, when you work with a view object, you work with only a single row set of results at a time. To simplify this use case, the view object contains a default `RowSet`, which, in turn, contains a default `RowSetIterator`. The default `RowSetIterator` allows you to call all of the data-retrieval methods directly on the `ViewObject` component itself, knowing that they will apply automatically to its default row set.

In addition, you can declaratively define view criteria for a view object. With a view criteria, you specify query conditions that augment the `WHERE` clause of the target view object in order to filter the results. You can then use those view criteria to create Query-by-Example search forms, filter row sets or lists-of-values (LOVs) at runtime, or create varying view instances based on a single view object definition.

4.4.3.1 Types of View Objects

There are two main types of view objects:

- Entity-based view objects, which access data from one or more entity objects and coordinate with those entity objects to update the data source based on user actions.
- Read-only view objects, which have direct access to the data. Because read-only view objects do not require intermediary objects, they access data more quickly than entity-based view objects. Create read-only view objects if you have use cases where you need to access data without modifying it. You might have a read-only view object and an entity-based view object for the same table.

In addition, you can create view objects with other data sources such as:

- Direct SQL queries of the database
- Programmatic sources
- Static data from CSV files

You can also create polymorphic view objects, in which multiple row set types with a common inheritance hierarchy are represented in a single view object.

4.4.3.2 View Object Definition Files

Similar to working with entity objects, when you use JDeveloper's wizards and visual editors to create and configure a view object definition, JDeveloper creates an XML file that contains the declarative metadata that defines the runtime behavior of that view object and features that are used in the UI, such as UI hints and validation rules. At runtime, this metadata is injected into an instance of the generic framework class `oracle.jbo.server.ViewObjectImpl`.

It is also possible to add custom functionality to a view object by writing custom classes that extend ADF Business Components framework classes. For information, see Working Programmatically with View Objects in *Developing Fusion Web Applications with Oracle Application Development Framework*.

4.4.3.3 Ways to Configure View Objects

View objects are part of ADF Business Components implementation of ADF Model. As such, you can add declarative metadata to a view object definition to configure its behavior.

You can define the same declarative metadata for a view object as you can for an entity object (with the exception that you cannot raise business events in view objects). In addition, you can set other types of metadata for a view object, such as the following:

- View criteria, which function as further refined queries and which are represented in the Data Controls panel as named queries, from which you can declaratively create search forms.
- List UI hints, which can be used to guide how lists of values are presented in the user interface.
- UI categories, which can be used for presenting titled groups of attributes in dynamic forms.
- View accessors, which can be used to provide a data source for view instance attributes involved in either list-based attribute validation or lists of values.
- Row finders, which can be used to match view instance rows by non-key attribute values and to initiate row updates either programmatically or through ADF web services.

4.4.4 View Links

Relationships between view objects are handled by view links, which define a relationship between two view objects based on sets of entity attributes from each. Like entity associations, these can range from simple one-to-many relationships based on foreign keys to complex many-to-many relationships.

Individual instances of view objects can also be related by individual instances of view links, which create a master-detail relationship between the query result sets. For example, suppose that you have view object definitions representing a query for department information and a query for employee information, and a view link between the view objects representing the relationship between a department and its employees. If an instance of the former view object definition, `allDepartments`, is related to an instance of the latter, `employeesInDepartment`, by an instance of the view link, those instances will be synchronized: whenever a particular row of `allDepartments` is selected, `employeesInDepartment` will only display details of that row.

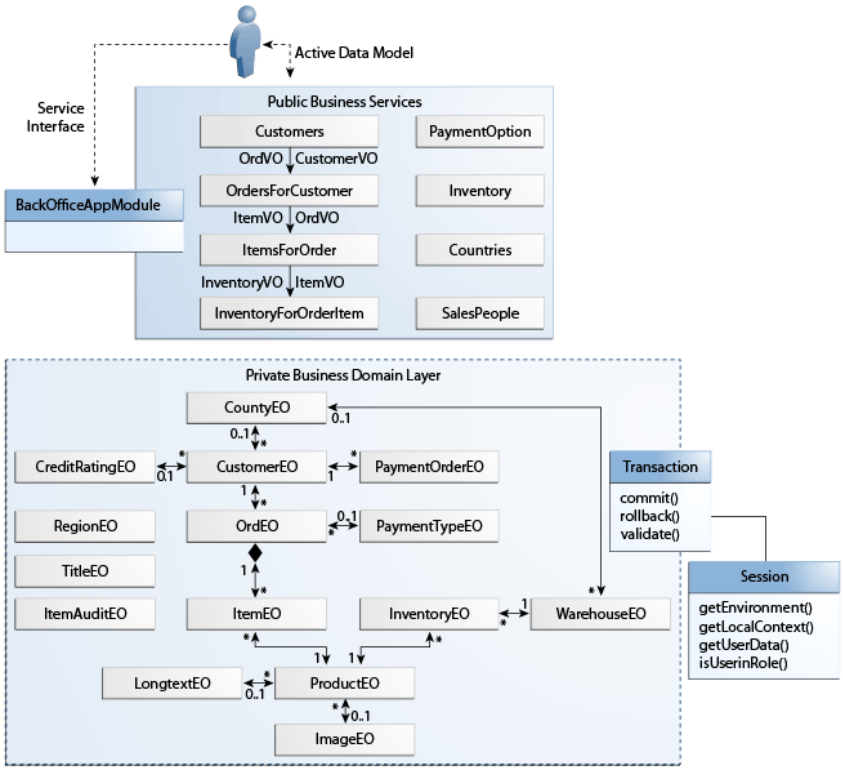
4.4.5 Application Modules

Oracle ADF application modules are the ADF Business Components implementation of ADF Model data controls. Application modules represent particular application tasks. The application module definition provides a data model for the task by aggregating the view object and view link instances required for the task. It also provides services that help the client accomplish the task. For example, an application module can represent and assist with tasks such as:

- Updating customer information
- Creating a new order
- Processing salary increases

Figure 4-7 illustrates how an application module works with other business components.

Figure 4-7 Application Module Within the ADF Business Component Architecture



In addition, application modules have pooling and state management features that simplify making applications scalable, well-performing, and able to handle failover.

4.4.5.1 Types of Application Modules

You can use application module definitions in the following ways:

- As a service object, in which case each instance of the MVC application has access to one instance of the application module. These root-level application module instances control ADF Business Components transaction objects, which in turn control the entity and view caches.
- As a reusable object for nesting, in which case you can create a data model and service methods on it and then nest one of its instances in other application module definitions. Those application module definitions can, in turn, access the nested module's methods and data model. Nested application modules share the root-level application module's transaction
- As a shared application module, in which data is cached for reuse across sessions and requests. Shared application modules are particularly useful for optimizing performance when you have data that does not change very frequently and needs to be accessed across multiple sessions and requests.

4.4.5.2 Application Module Definition Files

An application module definition can have one or two parts:

- An XML file, which represents the portion of the application that can be developed declaratively: the view object and view link instances that the application module contains and the way in which they are related. For many application modules, the XML file by itself is sufficient.

- An application module class, which lets you write custom code such as service methods that an MVC application can invoke for batch data handling. Application module classes extend the class `oracle.jbo.server.ApplicationModuleImpl`. If you do not need to write custom service methods, you need not generate an application module class—ADF can use `oracle.jbo.server.ApplicationModuleImpl` directly.

4.4.5.3 Service-enabled Application Modules

Service-enabled application modules are ADF application modules that you advertise through a service interface to service consumers. There are three scenarios for service consumers to consume a published service-enabled application module:

- web service access
- Service Component Architecture (SCA) composite access
- access by another ADF application module

The Service Component Architecture (SCA) provides an open, technology-neutral model for implementing remotable services that are defined in terms of business functionality and that make middleware functions more accessible to application developers. ADF Business Components supports an SCA-compliant solution through application modules you can publish with a service interface. The service interface is described for Fusion web application clients in a language-neutral way by the combination of WSDL and XSD.

When you service-enable your application module, JDeveloper generates the artifacts, which comprise the following files:

- the Java interface defining the service
- an EJB session bean that implements this Java interface
- a WSDL file that describes the service's operations
- an XML Schema Document (XSD) that defines the service's data structures

SCA defines two kinds of service:

- Remotable services, typically coarse-grained and designed to be published remotely in a loosely coupled SOA architecture
- Local services, typically fine-grained and designed to be used locally by other implementations that are deployed concurrently in a tightly coupled architecture

ADF Business Components services fall into the first category, and should only be used as remotable services.

ADF Business Components services, including data access and method calls, defined by the remote application modules are interoperable with any other application module. This means the same application module can support interactive web user interfaces using ADF data controls and web service clients.

Any development team can publish a service-enabled application module to contribute to the Fusion web application. The Fusion web application assembled from remote services also does not require the participating services to run on a single application server.

Although the web applications may run on separate application servers, the appearance that SCA provides is one of a unified application. Consuming client

projects use the ADF service factory lookup mechanism to access the data and any business methods encapsulated by the service-enabled application module. At runtime, the calling client and the ADF web service may or may not participate in the same transaction, depending on the protocol used to invoke the service (either SOAP or RMI). Only the RMI protocol and a Java Transaction API (JTA) managed transaction support the option to call the service in the same transaction as the calling client. By default, to support the RMI protocol, the ADF web service is configured to participate in the same transaction.

For information on enabling an application module as a service data object (SDO) component, see Integrating Service-Enabled Application Modules in *Developing Fusion Web Applications with Oracle Application Development Framework*.

4.4.5.4 Application Module Pooling

Applications you build that leverage an application module as their business service take advantage of an automatic application module pooling feature. This facility manages a configurable set of application module instances that grows and shrinks as the end-user load on your application changes. Due to the natural "think time" inherent in the end user's interaction with your application user interface, the number of application module instances in the pool can be smaller than the overall number of active users using the system. As a given end user visits multiple pages in your application to accomplish a logical task, an application module instance in the pool is acquired automatically from the pool for the lifetime of each request. At the end of the request, the instance is automatically returned to the pool for use by another user session.

To optimize your application's performance, you can tune application module pooling properties, such as initial and maximum pool size and the amount of time application module instances must be inactive before they can be removed from the pool.

4.4.5.5 Application State Management

You can use application module components to implement completely stateless applications or to support a unit of work that spans multiple browser pages. An application module supports *passivating* (storing) its pending transaction state to an XML document, which is stored in the database in a single, generic table, keyed by a unique passivation snapshot ID. It also supports the reverse operation of activating pending transaction state from one of these saved XML snapshots. This passivation and activation is performed automatically by the application module pool when needed. Activation can be triggered by server failover or simply because a user session spans multiple instances in the application module pool before it is completed.

4.5 Overview of the ADF Business Components Process Flow

Creating a business service layer based on ADF Business Components consists of the following general steps:

1. In JDeveloper, create an application workspace for the application.
2. Create custom classes that extend the base framework classes and then configure the model project to base any business components that you create on these custom classes. These classes provide a mechanism to later change base framework behavior and have those changes apply to all of the business components you have created in the application.
3. Using wizards in JDeveloper's New Gallery, create a combination of the following objects:

- Entity objects
- Entity associations
- View objects based on the entity objects
- Optionally, view objects based on queries directly to the database
- View links between view objects to establish master-detail relationships
- Create application modules and include the appropriate view objects and view links within them to establish your data model

4. Optionally, use JDeveloper's visual editors to declaratively specify business rules for the entity objects and view objects.
5. Use the ADF Model Tester to test the data model's business logic.
6. Tune the application modules for performance.
7. If participating in a SOA application, publish the services so that they can be consumed by an external application.
8. Using the Data Controls panel and various binding editors, create databound components in the view layer.

4.6 Learning More About ADF Business Components

The following resources provide detailed information about using ADF Business Components in applications:

- For information on creating business services with ADF Business Components, see Building Your Business Services in *Developing Fusion Web Applications with Oracle Application Development Framework*.
- For information on creating user interfaces that use ADF Model data binding, see Creating a Databound Web User Interface in *Developing Fusion Web Applications with Oracle Application Development Framework*.
- For Javadoc documentation related to ADF Business Components, see the `oracle.jbo` package in the *Java API Reference for Oracle ADF Model*.

5

ADF Controller Task Flows

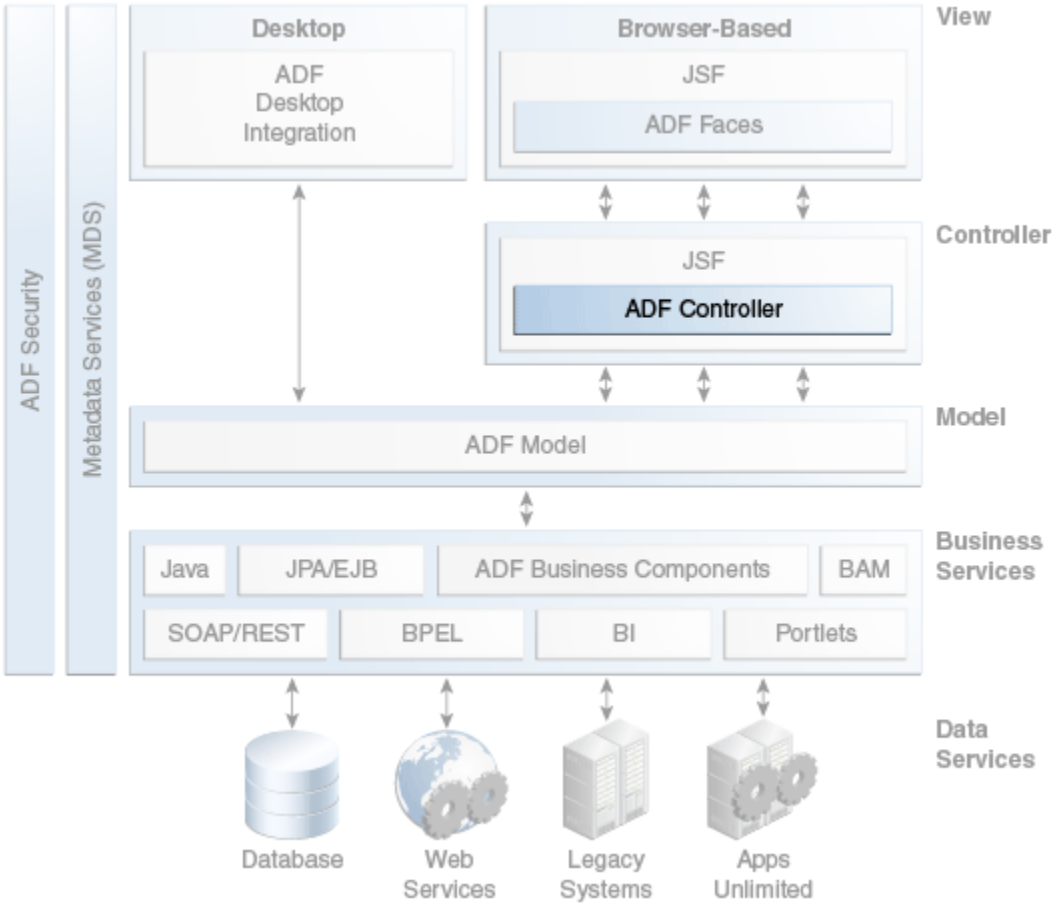
This chapter provides a high-level overview of the ADF Controller task flow technology, including its relationship to JSF's built-in controller support. In addition, it describes features and benefits of task flows, including modular and reusable task flows, flows between regions of a single page, transaction management, save points, declarative router decisions, and declarative exception handling.

This chapter includes the following sections:

5.1 About ADF Controller

ADF Controller is a declarative framework that, through the concept of task flows, builds upon the JSF page navigation support represented by the `javax.faces.webapp.FacesServlet` class and the `faces-config.xml` file. Figure 5-1 shows how ADF Controller fits into an enterprise application.

Figure 5-1 Overall Application Architecture with ADF Controller



5.2 Core Benefits of ADF Controller

ADF Controller provides the following benefits for developers:

- Applications can be broken up into a series of modular task flows that call one another.
- Task flows can contain nodes representing views, method calls, routing conditions, and calls to other task flows. (In a basic JSF application, page flow nodes can only be JSF pages.)
- You can pass parameters between task activities and different task flows.
- You can generate a page hierarchy with navigation links for your end users based on the task flow view activities in your application.
- Task flows are reusable.
- You can package a task flow into an ADF Library JAR and import it for use in another application's project.
- Task flows can be called remotely from different ADF applications.
- Task flows can also be used to secure your Fusion web application by reducing the number of access points that you expose to end users and by enforcing specific access privileges.

5.3 Key Concepts of ADF Controller

ADF Controller's central feature is the *task flow*. Task flows provide a modular approach for defining control flow in a Fusion web application. Instead of representing an application as a single large JSF page flow, you can break it up into multiple task flows, each of which contains a portion of the application's navigational graph.

A task flow consists of activity nodes, which represent simple logical operations such as displaying a page or page fragment, executing application logic, or calling another task flow. The transitions between the activities are called control flow cases.

Task flows include such features as invocation of custom business logic as part of the page flows, declarative router decisions, declarative exception handling, additional memory scopes, flows within pages, and reusable flows. Task flows also include built-in support for using flows to demarcate transaction boundaries and routing navigation to non-viewable targets such as method calls.

Using JDeveloper, you can create task flows visually using a diagram editor.

5.4 Key Components of ADF Controller

ADF Controller consists of the key components that are described in the following sections:

- Unbounded Task Flows
- Bounded Task Flows
- Task Flow Activities
- Task Flow Templates
- Save Points
- Integration with pageFlowScope, backingBeanScope, and viewScope
- Integration with the ADF Faces Train Component
- Integration with the ADF Faces Region Component

5.4.1 Unbounded Task Flows

A Fusion web application always contains a single ADF unbounded task flow, which contains the application's entry point, meaning any view activities that can be directly requested by a browser. An application's unbounded task flow can also contain other view activities (including pages with bookmarkable URLs), control flow rules, and calls to bounded task flows.

However, unbounded task flows cannot accept or return parameters or serve as transaction boundaries.

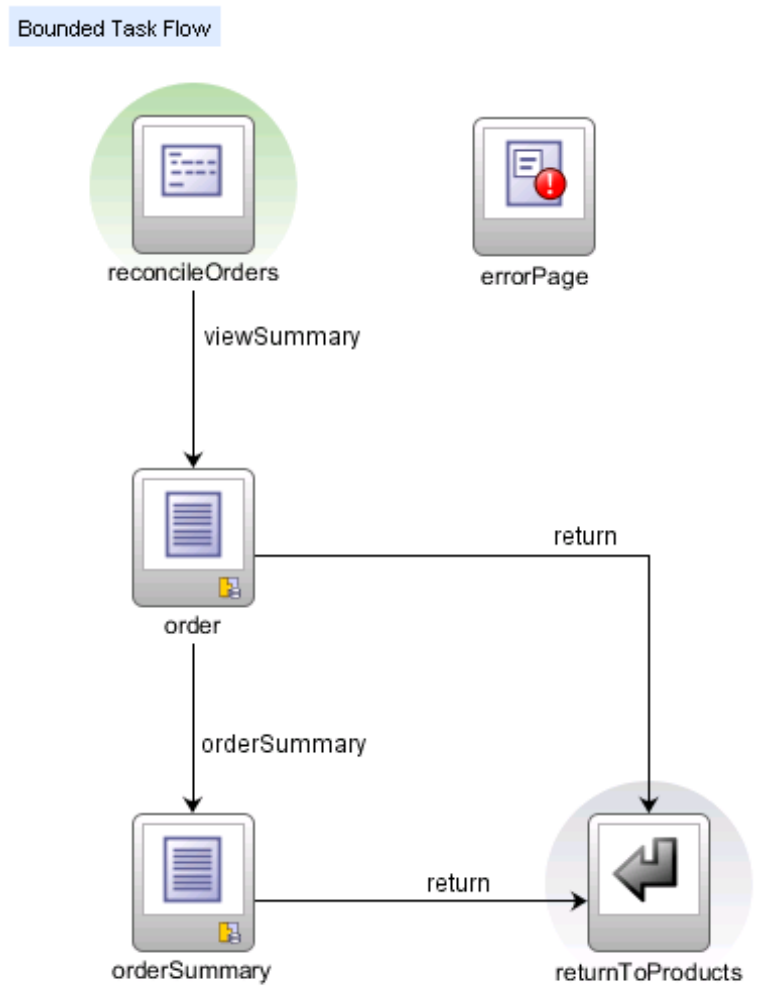
A typical application is a combination of the unbounded task flow and one or more bounded task flows. For example, JDeveloper, by default, creates an empty unbounded task flow (source file name is `adfc-config.xml`) when you create an application using the Fusion Web Application template. At runtime, the Fusion web application can call bounded task flows from activities that you added to the unbounded task flow.

5.4.2 Bounded Task Flows

Bounded task flows are private task flows that can be called from an unbounded task flow or from another bounded task flow. A bounded task flow has a single entry point and zero or more exit points. It contains its own set of private control flow rules, activities, and managed beans. It allows reuse, parameters, transaction management, reentry, and can render within an ADF region in a JSF page or page fragment. Figure 5-2 shows an example of a bounded task flow as it appears in the diagram editor.

Unlike unbounded task flows, bounded task flows cannot contain bookmarkable view activities.

Figure 5-2 ADF Bounded Task Flow



5.4.3 Task Flow Activities

Each task flow consists of some of the following actions and control cases:

- *Views*, which displays a JSF page or page fragment.

- *URL views*, which enable you to redirect the root view port (for example, a browser page) to any URL-addressable resource, such as bounded task flows, view activities in the unbounded task flow, and addresses external to the current web application.
- *Routers*, which route control to activities based on the runtime evaluation of EL expressions.
- *Method calls*, which allow you to call custom or built-in methods that invoke application logic from anywhere within an application's control flow.
- *Task flow calls*, which enable you to call a bounded task flow from either the unbounded task flow or a bounded task flow. A task flow call activity allows you to call a bounded task flow located within the same or a different application.
- *Task flow returns*, which enable you to identify the point in an application's control flow where a bounded task flow completes and sends control flow back to the caller.
- *Save point restores*, which enable you to restore a previous persistent save point in an application supporting save for later functionality. A save point captures a snapshot of the Fusion web application at a specific instance. Save point restore enables the application to restore whatever was captured when the save point was originally created. For more information, see Save Points.
- *Parent action activities*, which allow a bounded task flow running in an ADF region to generate outcomes that it passes to the parent view activity. The outcomes are used to navigate the task flow containing the parent view activity rather than navigating the task flow of the ADF region.
- *Control flow cases*, which define how control passes from one activity to another in a task flow. A control flow rule can contain one or more control flow cases to identify the activity to which control flow passes.

  Control flow rules are based on JSF navigation rules, but provide additional features. Whereas JSF navigation is always between pages, task flow control flow rules can also handle transitions between other activities, such as method calls and entry and exit points of bounded task flows.
- *Wildcard control flow rules*, which enable you to use a wildcard expression to specify which view activities are to be passed to a given control flow rule.

5.4.4 Task Flow Templates

Task flow templates are a construct that help simplify creation of bounded task flows, help enforce consistent runtime behavior of task flows, and make it easier to refactor an application's task flows.

As their name implies, task flow templates are a mechanism for creating standard task flow types. A bounded task flow created from a task flow template will have definitions for the same set of task flow activities, control flows, input parameters, and managed beans as the task flow template. You can create task flow templates for yourself or other application developers to use as a starting point when creating new bounded task flows.

In addition, you can use task flow templates at runtime. When you create a task flow (or another task flow template) based on a template, you can maintain an association between the newly created flow or template and the base template by selecting the **Update the Task Flow When the Template Changes** checkbox. If you do so, subsequent changes that you make to the base template (such as adding new view

activities) get propagated to the child flows (or templates) at runtime. (If there are conflicts between the child and parent, the child overrides the parent.) You can change, update, or disassociate the parent task flow template of a child bounded task flow or task flow template at any point during development of the child.

5.4.5 Save Points

You can add a save point to a task flow to capture the state of a Fusion web application at a particular instance. This allows you to save application state if, for example, a user leaves a page without finalizing it. The application state can be restored at a later point. The saved information includes the following:

- UI state of the current page, including selected tabs, selected checkboxes, selected table rows, and table column sort order
- State information saved in memory scopes, including session and page flow scope.
- The state of serializable managed beans
- The navigation state, which is derived from the task flow call stack, which tracks where the end user is in the application and the navigation path for getting there
- The ADF Model state, which consists of any data model updates made from when the current bounded task flow begins

5.4.6 Integration with pageFlowScope, backingBeanScope, and viewScope

ADF Faces provides `pageFlowScope`, `backingBeanScope`, `viewScope` shared memory scopes to augment standard JSF memory scopes.

Of particular relevance to task flows is `pageFlowScope`, which defines a unique storage area for each instance of a task flow. The `pageFlowScope` scope begins when the task flow begins and ends when the task flow ends.

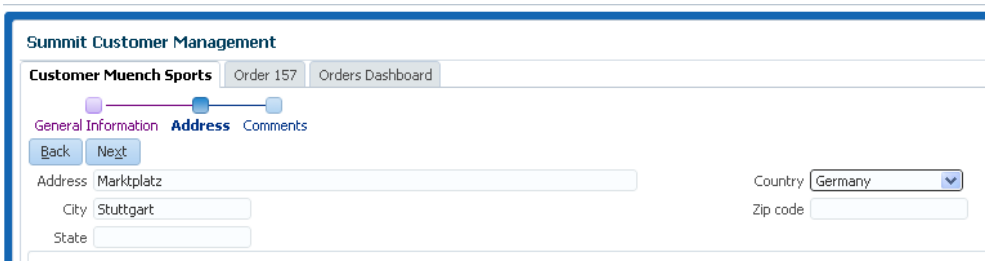
For example, a managed bean with `pageFlowScope` can be accessed within the task flow, even if its activities are spread across different pages. However, it is out of scope for anything outside of that task flow, including other task flows that call its task flow and UI components that are on the same page but in a region that is encompassed by a different task flow.

For more information on these scopes, see Key Concepts of ADF Faces.

5.4.7 Integration with the ADF Faces Train Component

When you create task flows, you can specify that they use the ADF Faces `train` and `trainButtonBar` components to guide users through the steps specified by the task flow. Figure shows a page fragment from the Summit sample application for ADF Task Flows that renders these components.

Figure 5-3 ADF Train Component



5.4.8 Integration with the ADF Faces Region Component

When you add a bounded task flow to JSF page or page fragment, it is wrapped within a `region` tag.

5.5 Overview of the ADF Controller Process Flow

Using ADF Controller consists of the following basic steps:

1. In JDeveloper, create an application workspace for the application.
2. Using wizards in JDeveloper's New Gallery, create task flow files.
3. Using JDeveloper's diagram editor for task flows, design the task flow by adding task flow activities (view activities, method call activities, and so on) and control flow cases.

   As part of this process, you can create the pages that are used in the task flow from scratch, or you can drag and drop existing JSF pages or page fragments to the diagram editor to create view activities in the task flow.
4. Test run the task flows using the Integrated WebLogic Server from within JDeveloper.

5.6 Learning More About ADF Controller

The following resources provide detailed information about using ADF Controller in applications:

- Creating ADF Task Flows in *Developing Fusion Web Applications with Oracle Application Development Framework*
- JDeveloper Tutorials on ADF Controller
- *Task Flow Design Fundamentals* white paper

Part III

Oracle ADF View Technologies

Part III contains the following chapters:

- ADF Faces
- ADF Desktop Integration

6

ADF Faces

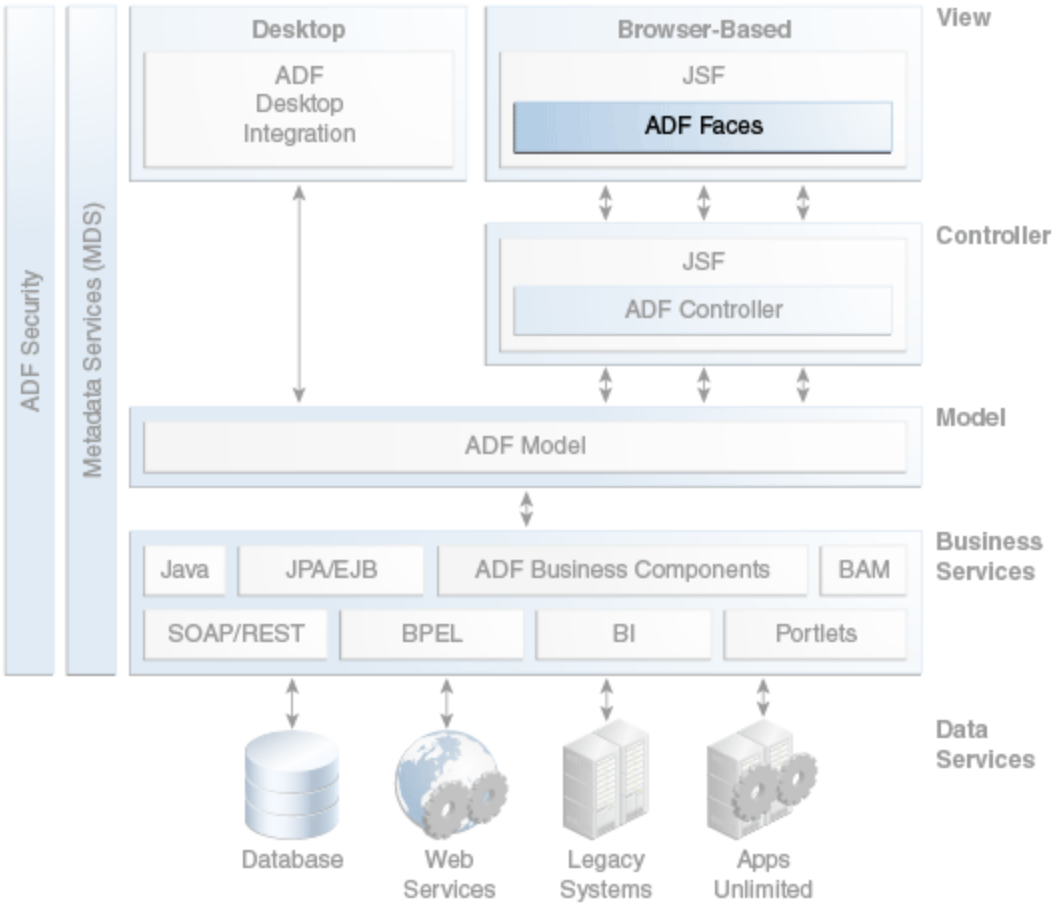
This chapter provides a high-level overview of the ADF Faces framework and component features. It includes information on the component set, including data visualization (DVT) components, and on features such as Ajax support, client-side events, user personalization, and additional memory scopes.

This chapter includes the following sections:

6.1 About ADF Faces

ADF Faces is an implementation of JSF web components that provides enhanced functionality for developers and users of sophisticated web applications. You can use ADF Faces components in an application either with or without other Oracle ADF features, such as ADF Model and ADF Controller. Figure 6-1 shows how ADF Faces fits into an enterprise application.

Figure 6-1 Overall Application Architecture with ADF Faces



6.2 Core Benefits of ADF Faces

ADF Faces provides the following benefits for web application developers:

- Large set of fully featured rich components

  The library provides over 150 Rich Internet Application (RIA) components, including geometry-managed layout components, text and selection components, sortable and hierarchical data tables and trees, menus, in-page dialogs, data visualization components such as charts and treemaps, and general controls.

- Built-in Ajax support

  Many ADF Faces components have Ajax-style functionality implemented natively, which simplifies development of responsive user interfaces. For example, the ADF Faces table component lets you scroll through the table, sort the table by clicking a column header, mark a row or several rows for selection, and even expand specific rows in the table, all without requiring the page to be submitted to the server, and with no coding needed.

- Limited need for developers to write JavaScript

  ADF Faces hides much of the complex JavaScript from you. Instead, you declaratively control how components function. You can implement a rich, functional, attractive user interface using ADF Faces in a declarative way that does not require the use of any JavaScript at all.

- Enhanced lifecycle on both server and client

  ADF Faces extends the standard JSF page request lifecycle. Examples include a client-side value lifecycle, a subform component that allows you to create independent submittable regions on a page without needing multiple forms, and an optimized lifecycle that can limit the parts of the page submitted for processing.

- User-driven personalization

  Many ADF Faces components allow users to change the display of the component at runtime. By default, these changes live only as long as the page request. However, you can configure your application so that the changes can be persisted through the length of the user's session.

- End-user drag and drop

  The ADF Faces framework allows the user to move data from one location to another by dragging and dropping one component onto another.

- Integration with other Oracle ADF technologies

  You can use ADF Faces in conjunction with the other Oracle ADF technologies, including ADF Business Components, ADF Controller, and ADF data binding. For more information about using ADF Faces with the ADF technology stack, see Introduction to Building Fusion Web Applications with Oracle ADF in *Developing Fusion Web Applications with Oracle Application Development Framework*.

- Integrated declarative development with Oracle JDeveloper

  JDeveloper contains built-in declarative support for ADF Faces components, including a visual layout editor, a Components window that allows you to drag and drop an ADF Faces component onto a page, and a Properties window where you declaratively configure component functionality.

6.3 Key Concepts of ADF Faces

This section provides an overview of the key aspects of ADF Faces and how they work within ADF applications.

The ADF Faces framework is based on the following principles and features:

- Built to the JavaServer Faces (JSF) specification

  ADF Faces supports JSF features such as Facelets. Some JSF features that were introduced in JSF 2.0 have parallel functionality in ADF Faces. To understand the functional overlap that exists between ADF Faces and JSF, see the *JavaServer Faces 2.0 Overview and Adoption Roadmap in Oracle ADF Faces and Oracle JDeveloper 11g* whitepaper on OTN at `http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adffaces-jsf20-190927.pdf`.

- Partial page rendering (PPR)

  Many ADF Faces components have *partial page rendering*, which is an Ajax-style functionality implemented natively. For example, you can update a chart based on user input in another component on the page, all without requiring the page to be submitted to the server, and with no coding needed.

- Partial page navigation

ADF Faces applications can use PPR for navigation, which eliminates the need to repeatedly load JavaScript libraries and style sheets when navigating between pages.

- Client-side validation, conversion, and messaging

  ADF Faces validators can operate on both the client and server side. Client-side validators are written in JavaScript and validation errors caught on the client-side can be processed without a round-trip to the server.

- Additional memory scopes

  The ADF Faces framework includes additional shared memory scopes that complement other features, such as page fragments and ADF task flows.

  – `pageFlowScope`: The object is available as long as the user continues navigating from one page to another within a given task flow. If the user opens a new browser window or tab and begins navigating, that window will have its own `pageFlowScope` scope.

  – `backingBeanScope`: Used for managed beans for page fragments and declarative components only. The object is available for the duration between the time an HTTP request is sent until a response is sent back to the client. This scope is needed because there may be more than one page fragment or declarative component on a page, and to avoid collisions between values, any values must be kept in separate scope instances.

  – `viewScope`: The object is available until the ID for the current view changes. Unlike the JSF `viewScope`, objects stored in the ADF Faces `viewScope` will survive page refreshes and redirects to the same view ID.

- Server-side push and streaming

  The ADF Faces framework includes server-side push that allows you to provide real-time data updates for ADF Faces components.

- Active geometry management

  ADF Faces provides a client-side geometry management facility that allows components to determine how best to make use of available screen real-estate. The framework notifies layout components of browser resize activity, and they in turn are able to resize their children. This allows certain components to stretch or shrink, filling up any available browser space.

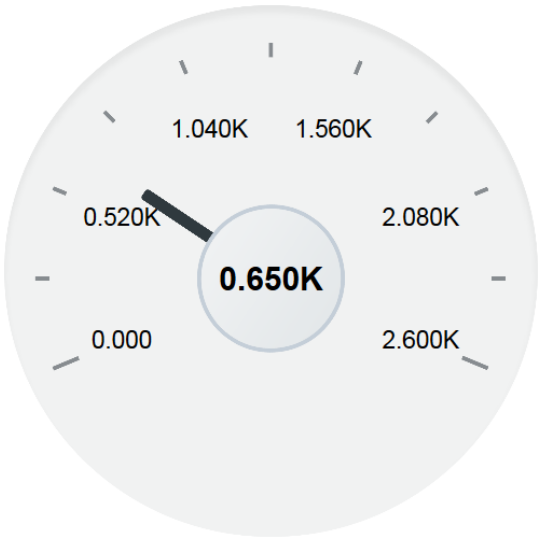
- Advanced templating and declarative components

  You can create page templates, as well as page fragments and composite components made up of multiple components, which can be used throughout your application.

- Advanced visualization components

  ADF Faces includes data visualization (DVT) components, which are capable of rendering dynamic charts, gauges, and other graphics that provide a real-time view of underlying data. Figure 6-2 shows a gauge component that graphically represents the current stock level of a given product in a label (**0.650K**) that appears in the center of the gauge.

Figure 6-2 ADF DVT Gauge Component



- Event handling

  ADF Faces adheres to standard JSF event handling techniques, as well as offering a complete client-side event model.

- ADF Skins

  You can declaratively create or modify an ADF skin, which is a type of CSS file that determines the look and feel of ADF Faces components. Oracle provides editors within JDeveloper that you can use to create and edit skins.

- Messaging and help

  The framework provides the ability to display tooltips, messages, and help for input components, as well as the ability to display global messages for the application. The help framework allows you to create messages that can be reused throughout the application. You create a help provider using a Java class, a managed bean, an XLIFF file, or a standard properties file, or you can link to an external HTML-based help system.

- Internationalization

  To simplify the process of creating and maintaining localizable text resources, JDeveloper uses resource bundles to store any component text that you add or edit using visual editors or the Properties window. In addition, ADF skins support pseudo-classes that you can use to change how an application renders in a particular locale. You can also configure your JSF page or application to use different locales so that it displays the correct language based on the language setting of a user's browser.

- Accessibility

  ADF Faces components have built-in accessibility that work with a range of assistive technologies, including screen readers. ADF Faces accessibility audit rules provide direction to create accessible images, tables, frames, forms, error messages, and popup windows using accessible HTML markup.

For a complete guide to using ADF Faces technology, see Introduction to ADF Faces in *Developing Web User Interfaces with Oracle ADF Faces*.

6.4 Key Components of ADF Faces

ADF Faces components can be broken down into the following categories:

- General controls

  General controls include the following types of components:

  – Navigation components: Allow users to go from one page to the next. ADF Faces navigation components include buttons and links, as well as the capability to create more complex hierarchical page flows accessed through different levels of menus. In addition, ADF has a train component that enables you to structure the order in which a user navigates multi-step processes.

  – Images and icon components: Allow you to display images ranging in complexity from icons to video.

- Text and selection components

  These components allow you to display output text, accept input text, or enable users to select from a pre-populated list. These components fall into the following sub-categories:

  – Output components: Display text and graphics, and can also play video and music clips.

  – Input components: Allow users to enter data or other types of information, such as color selection or date selection. ADF Faces also provides simple lists from which users can choose the data to be posted, as well as a file upload component.

  – List-of-Values (LOV) components: Allow users to make selections from lists driven by a model that contains functionality like searching for a specific value or showing values marked as favorites. These LOV components are useful when a field used to populate an attribute for one object might actually be contained in a list of other objects, as with a foreign key relationship in a database.

- Data Views

  ADF Faces provides a number of different ways to display complex data, including the following:

  – Table and tree components: Display structured data in tables or expandable trees. ADF Faces tables provide functionality such as sorting column data, filtering data, and showing and hiding detailed content for a row. Trees have built-in expand/collapse behavior. Tree tables combine the functionality of tables with the data hierarchy functionality of trees.

  – Data visualization components: Allow users to view and analyze complex data in real time. ADF data visualization components include graphs, gauges, pivot tables, geographic maps, Gantt charts, hierarchy viewers, treemaps, sunbursts, and timelines.

  – Query components: Allow users to query data. The query component can support multiple search criteria, dynamically adding and deleting criteria, selectable search operators, match all/any selections, seeded or saved searches, a basic or advanced mode, and personalization of searches.

  - Specialty display components: The calendar component displays activities in day, week, month, or list view. You can implement popup components that allow users to create, edit, or delete activities. The carousel component allows you to display a collection of images in a scrollable manner.

- Menus and toolbars

  ADF Faces provides navigation components that render items such as tabs and breadcrumbs for navigating hierarchical pages. The framework provides an XML-based menu model that, in conjunction with a metadata file, contains all the information for generating the appropriate number of hierarchical levels on each page, and the navigation items that belong to each level.

- Layout components

  Layout components act as containers to determine the layout of the page, ADF Faces layout components also include interactive container components that can show or hide content, or that provide sections, lists, or empty spaces. JDeveloper provides prebuilt quick-start layouts that declaratively add layout components to your page based on how you want the page to look.

  In addition to standard layout components, ADF Faces also provides the following specialty layout components:

  - Explorer-type menus and toolbar containers: Allow you to create menu bars and toolbars. Menus and toolbars allow users to select from a specified list of options (in the case of a menu) or buttons (in the case of a toolbar) to cause some change to the application.

  - Secondary windows: Display data in popup windows or dialogs. In addition, ADF Faces provides a dialog framework to support creating popup browser windows that are separate from the parent page and managing their interaction without JavaScript.

  - Core structure components and tags: Provide the tags needed to create pages and layouts, such as documents, forms and subforms, and resources.

- Operation Tags

  These tags work with components to provide additional functionality, such as drag and drop, validation, and a variety of event listeners.

6.5 Overview of the ADF Faces Process Flow

Creating an ADF Faces view consists of the following basic steps:

- In JDeveloper, create an application workspace from one of the templates in the New Gallery.

  Alternatively, open an existing application workspace that contains business services or other code that you want to integrate with the application and add a new project to that workspace for ADF Faces components.

- In the ADF Faces project, create page flows. These can be either basic JSF page flows or ADF Controller task flows.

- Create the pages using either JavaServer Pages (JSP) or Facelets templates and insert them into the page flows.

- Design the pages by dragging and dropping ADF Faces components from the Components window to the JSF page and using the Properties window to adjust their attributes.

  Part of this process is likely to include the creation of Expression Language (EL) expressions to define various aspects of component behavior, and possibly the creation of managed beans that you then reference from EL expressions.

- Test run the application using the Integrated WebLogic Server from within JDeveloper.

6.6 Learning More About ADF Faces

The following resources provide detailed information about using ADF Faces in applications:

- Introduction to ADF Faces in *Developing Web User Interfaces with Oracle ADF Faces*
- Creating a Databound Web User Interface in *Developing Fusion Web Applications with Oracle Application Development Framework*
- About Skinning a Web Application in *Developing ADF Skins*
- JDeveloper Tutorials on ADF Faces
- *Tag Reference for Oracle ADF Faces*
- *Tag Reference for Oracle ADF Faces Data Visualization Tools*
- The Oracle ADF Faces page on Oracle Technology Network: `http://www.oracle.com/technetwork/developer-tools/adf/overview/index-092391.html`
- The *JavaServer Faces 2.0 Overview and Adoption Roadmap in Oracle ADF Faces and Oracle JDeveloper 11g* whitepaper on OTN at `http://www.oracle.com/technetwork/developer-tools/adf/learnmore/adffaces-jsf20-190927.pdf`

7

ADF Desktop Integration

This chapter provides a high-level overview of the ADF Desktop Integration technology, which enables you to integrate Fusion web applications with Microsoft Excel workbooks.

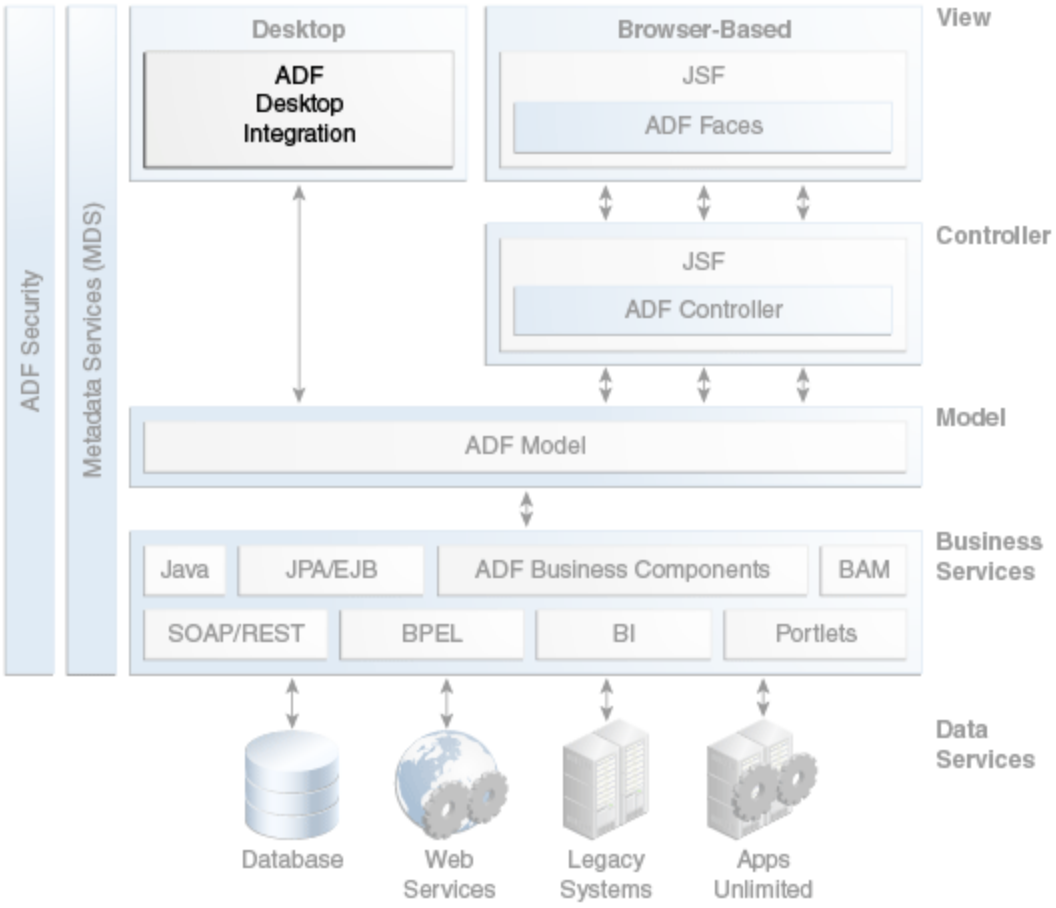
This chapter includes the following sections:

7.1 About ADF Desktop Integration

ADF Desktop Integration provides a framework for Oracle ADF developers to extend the functionality provided by a Fusion web application to desktop applications in the form of Microsoft Excel workbooks.

As shown in Figure 7-1, ADF Desktop Integration can be used with other parts of the ADF technology stack, such as ADF Model, and ADF Business Components.

Figure 7-1 ADF Architecture with ADF Desktop Integration



7.2 Core Benefits of ADF Desktop Integration

The following are the core benefits of ADF Desktop Integration:

- It allows end users to work with Oracle ADF-based applications offline.
- It allows end users to work within Microsoft Excel's user interface, with which many users are already familiar and comfortable.
- Bulk entry and update of data may be easier to accomplish through a spreadsheet-style interface.
- It allows end users to use native Excel features such as macros, calculation, validation, and styles.

7.3 Key Concepts of ADF Desktop Integration

This section outlines the key concepts of ADF Desktop Integration.

7.3.1 Integration with Microsoft Excel

The ADF Desktop Integration components allow end users to manage data retrieved from a Fusion web application in an integrated Excel workbook. You configure the ADF components and the worksheet that hosts it them so that the end user can upload changes they make to data in worksheet to a Fusion web application.

When using a workbook with ADF Desktop Integration components, you can take advantage of standard Excel features such as the following:

- Validation. You can use Excel's data validation features to control the type of data or the values that end users enter into a cell. These features allow you to restrict data entry to a certain range of dates, limit choices by using a list, or make sure that only positive whole numbers are entered in a cell.
- Styles. You can configure the appearance the application's data by using Excel's style and formatting features, including some predefined styles provided by ADF Desktop Integration. You can also use EL expressions to have styles applied dynamically.
- Formulas and calculated cells. You can write Excel formulas that perform calculations on values in an integrated Excel workbook. Formulas can be entered both in cells that reference Oracle ADF bindings and cells that do not reference Oracle ADF bindings.

  Also, you can use an EL expression to generate an Excel formula as the value of an ADF component.
- Macros. You can define and execute macros based on Excel events in an integrated Excel workbook.

7.3.2 Integration with ADF Page Definition Files

ADF Desktop Integration components are linked to page definition files in the Fusion web application. Page definition files define the bindings that populate the data in the Oracle ADF components at runtime. Page definition files also reference the action bindings and method action bindings that define the operations or actions to use on this data. You must define a separate page definition file for each Excel worksheet that you are going to integrate with a Fusion web application. The integrated Excel workbook can include worksheets that do not reference page definition files.

The link with the page definition files enables the integrated worksheets to also benefit from any validation that you have set up in the binding layer. Data that the end user enters or edits in one of the ADF Desktop Integration components, such as the ADF Table component, can be validated against these rules and conditions that are set the server side in the Fusion web application.

7.3.3 Runtime Synchronization with Fusion Web Applications

An ADF Desktop Integration integrated workbook extends and runs within the context of an ADF Fusion web application. Offline use of an ADF Desktop Integration integrated workbook is possible, but data synchronization and some user interface interactions require that a valid user session be established with the web application on which it is based at some point. ADF Desktop Integration relies on HTTP cookie-based session management for all of its interactions with the ADF Fusion web application, regardless of whether the web application is configured to enforce authentication or not.

When you deploy an ADF Desktop Integration-enabled Fusion web application from JDeveloper, references to the ADF Desktop Integration shared libraries are added to the appropriate descriptor files. For any Fusion web application that contains one or more projects referencing the ADF Desktop Integration Model API library or the ADF Desktop Integration Runtime library, a platform-dependent reference to the ADF Desktop Integration Model API shared library is added during deployment.

For any web application module (WAR) project that contains a reference to the ADF Desktop Integration Runtime library, a platform-dependent reference to the ADF Desktop Integration Runtime shared library is added during deployment.

The ADF Desktop Integration framework is composed of a client-side portion running on top of Microsoft .NET and two server components: the ADF Desktop Integration remote servlet and the ADF Desktop Integration download filter. The server components run within the context of an ADF Fusion web application. The client component acts as the view and controller and communicates with the servlet to synchronize data and execute business logic in the web application's model project. Communication between the client and server takes the form of HTTP requests and responses.

7.3.4 Security for Integrated Excel Workbooks

Whenever an integrated Excel workbook connects to a Fusion web application, the integrated workbook makes sure that a valid, authenticated user session is established before downloading any data. If you are using a Fusion web application that does not enforce authentication, the integrated Excel workbook verifies and creates a valid user session when it connects to the Fusion web application.

When the corresponding Fusion web application has ADF Security enabled, the ADF Desktop Integration enforces any security policies set for the page definitions that correspond to the integrated workbook. At runtime, end users without proper permissions for a page definition (binding container) are prevented from interacting with the associated integrated Excel worksheet. Any attempt to interact with an unauthorized binding container (for example, to download or submit data) is aborted, the end user is informed of the authorization failure, and all ADF Desktop Integration activity on the worksheet is disabled. No further interaction with the ADF Desktop Integration-disabled worksheet is possible until a new user session is established. To allow end users to interact with the integrated Excel worksheet, assign them the roles that have been granted access to the page definition.

If you save an Excel workbook containing data downloaded from a Fusion web application to a location, such as a network directory, where other users can access the Excel workbook, the data stored in the Excel workbook is accessible to other users.

You can enhance the security of an integrated Excel workbook using Excel's functionality to set a password on a workbook. It prevents unauthorized users from opening or modifying the workbook. For more information about Excel security features, see Excel's documentation.

7.4 Key Components of ADF Desktop Integration

ADF Desktop Integration functionality consists of the add-in features to Microsoft Excel that are outlined in this section.

7.4.1 Table-Type Components

ADF Desktop Integration contains table and read-only table components, which enable you to display and edit large sets of structured information, such as entire database tables. These components are analogous to ADF Faces components with the same name but with properties that are specific to the Excel workbook environment. These components also have associated actions, with which you can do things such as download from and upload to the associated Fusion web application.

After you add an ADF Table component to a worksheet, you configure it and the worksheet that hosts it, so that the ADF Table component downloads data from the

Fusion web application. To achieve this, you configure an Oracle ADF component, such as ADF Button, a worksheet ribbon button, or a worksheet event to invoke an action set. The action set that is invoked must include the ADF Table component Download action among the actions that it invokes.

7.4.2 Form-Type Components

ADF Desktop Integration contains basic UI components for input text fields, output text fields, labels, buttons, and lists of values (LOVs), which enable you to create forms for displaying and entering data. These components are analogous to ADF Faces components with the same name but with properties that are specific to the Excel workbook environment.

To enable the user to commit those changes back to the Fusion web application, you configure an ADF component to invoke an action set that handles the transferring of the changes back to the web application and committing them to the data source.

7.4.3 Action Sets

To enable you to string together multiple actions that can be invoked with a single user gesture in the workbook, ADF Desktop Integration provides *action sets*. An action set is an ordered list of one or more actions that execute in a specified order. You can associate an action set with a UI element in the worksheet or with a worksheet event, such as `Startup` or `Shutdown`.

The following types of actions are available:

- `ADFmAction` - an action binding or method action binding in the underlying page definition file.
- `ComponentAction` - an action that a component on the worksheet exposes. The ADF Table and ADF Read-only Table components are the only components in ADF Desktop Integration that expose actions.
- `WorksheetMethod` - an action provided by ADF Desktop Integration that handles coordination between the data in the worksheet and the Fusion web application. The available actions are `UpSync`, `DownSync`, and `DisplayWorksheetErrors`.
- `Confirmation` - Invokes a confirmation dialog.
- `Dialog` - Invokes a web page in a popup dialog or Excel's task pane.

7.5 Overview of the ADF Desktop Integration Process Flow

Developing with ADF Desktop Integration consists of the following steps:

1. On a Microsoft Windows system, make sure Internet Explorer is installed, and install Microsoft Excel and JDeveloper.
2. Configure the Microsoft Excel installation to trust access to the VBA project object model in order to make it accessible from ADF Desktop Integration.
3. In JDeveloper, install the ADF Desktop Integration add-in.

   The ADF Desktop Integration add-in is available in two editions, the Designer edition and the Runtime edition. Use the Designer edition to create and test integrated Excel workbooks, and the Runtime edition to enable end users to use ADF Desktop Integration and integrated Excel workbooks. However, do not install both editions of ADF Desktop Integration on the same system.

4. Create a Fusion web application.
5. In the application's model project, add data controls that expose the elements you require in Microsoft Excel.
6. In the application's user interface project, create page definition files that expose the Oracle ADF bindings to use in Excel.
7. Create the Excel workbooks that you intend to configure with Oracle ADF functionality.
8. Configure the Excel workbook using the Oracle ADF bindings that you exposed in the page definition files and the Oracle ADF components that ADF Desktop Integration provides.
9. Add the integrated Excel workbook to the JDeveloper project for your Fusion web application if it is not already packaged there. This makes sure that the Excel workbooks you integrate with your Fusion web application get deployed when you deploy your finalized Fusion web application.
10. Publish the completed workbook so that it is available to users when the application is deployed.
11. Configure one or more web pages in your Fusion web application to allow end users to access the integrated Excel workbooks.
12. Deploy the Fusion web application that contains the integrated workbook.
13. Make sure that end users who want to use the functionality that you configure in an integrated Excel workbook install the Runtime edition of ADF Desktop Integration.

7.6 Learning More About ADF Desktop Integration

For more information on using ADF Desktop Integration, see the following resources:

- Introduction to ADF Desktop Integration in *Developing Applications with Oracle ADF Desktop Integration*
- The ADF Desktop Integration page on OTN: `http://www.oracle.com/technetwork/developer-tools/adf/overview/index-085534.html`

Part IV

Oracle ADF Security, Customization, and Deployment

Part IV contains the following chapters:

8

ADF Security Framework

This chapter provides a high-level overview of the ADF Security framework, including its integration with the Oracle Platform Security Services (OPSS) architecture and the securing of ADF applications with declarative resource grants.

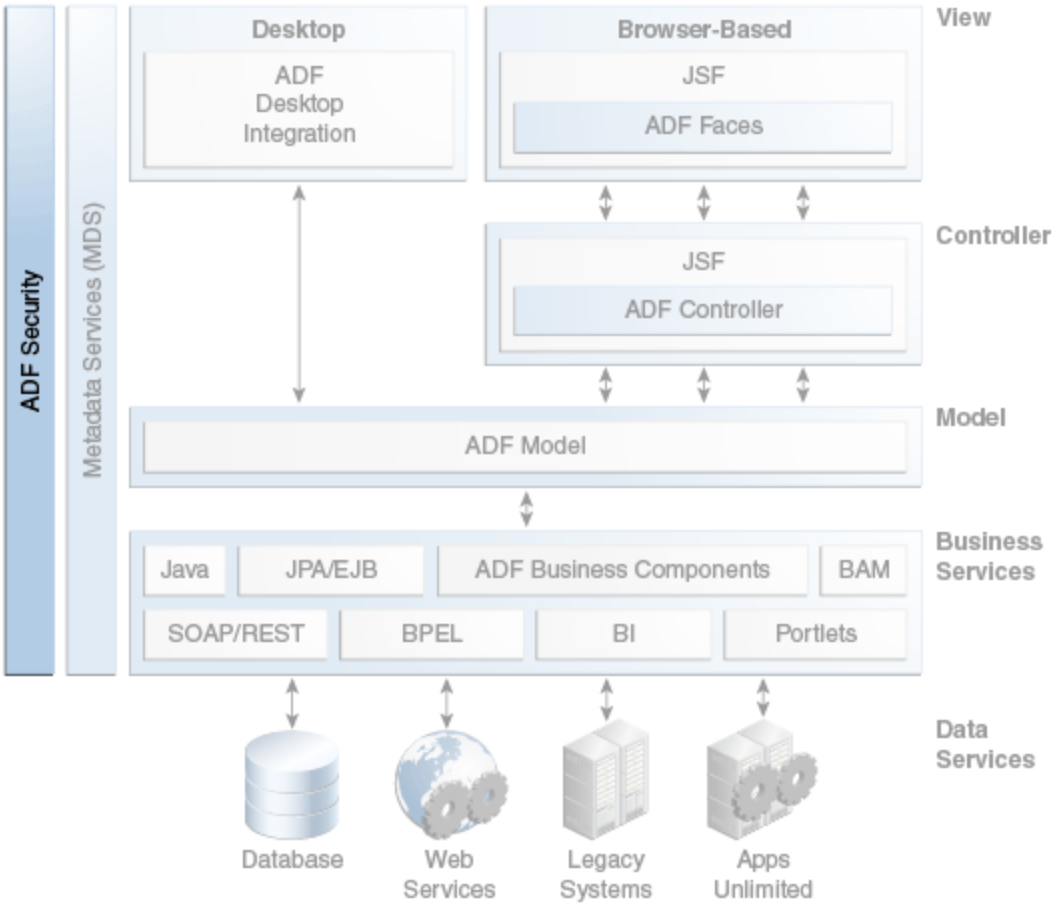
This chapter includes the following sections:

8.1 About the ADF Security Framework

In order to simplify the process of ensuring thorough application security, the Oracle Application Development Framework (Oracle ADF) provides the ADF Security framework. ADF Security is built on top of the Oracle Platform Security Services (OPSS) architecture, which in turn incorporates the Java Authentication and Authorization Service (JAAS) and Java EE container-managed security.

As shown in Figure 8-1, ADF Security encompasses the range of other components in the ADF technology stack, such as ADF Faces, ADF Controller, ADF Model, and ADF Business Components.

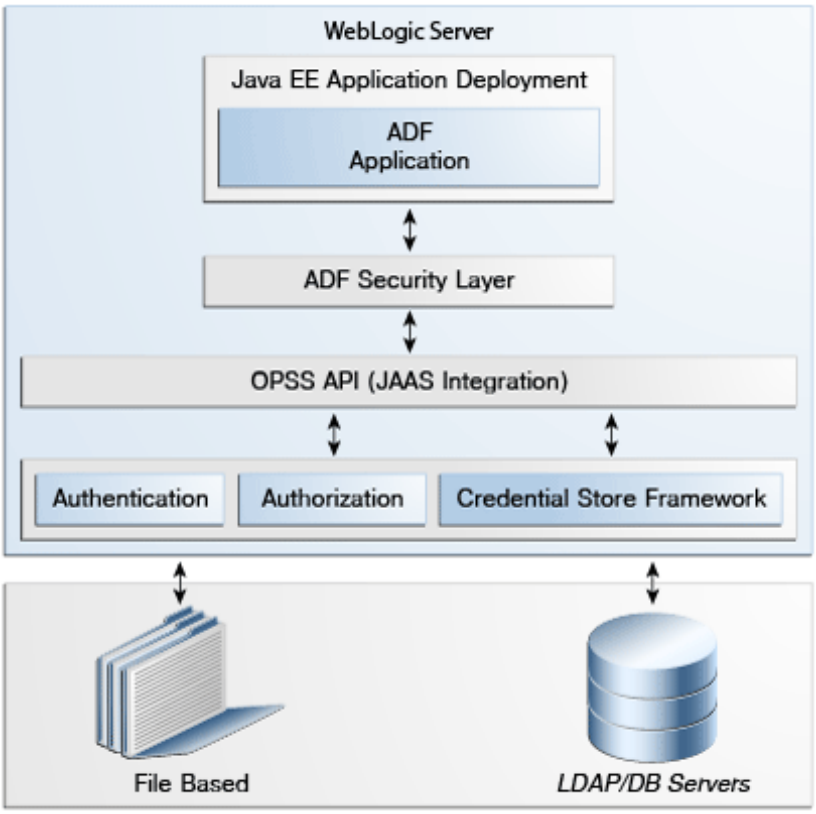
Figure 8-1 ADF Architecture with ADF Security



OPSS is the underlying security platform that provides security to Oracle Fusion Middleware, including WebLogic Server and Oracle ADF applications. OPSS is designed to be portable to third-party application servers, so developers can use OPSS as the single security framework for both Oracle and third-party environments, thus decreasing application development, administration, and maintenance costs.

Figure 8-2 conceptually illustrates the architecture of the ADF Security framework. The uppermost layer consists of the running ADF application. Below that is the ADF security layer, which implements the OPSS APIs and enables programmatic permission checks on resources from the application. The OPSS API layer serves as an abstraction layer for accessing providers of authentication, authorization, credential store framework services.

Figure 8-2 ADF Security Architecture



8.2 Core Benefits of ADF Security

ADF Security provides the following core benefits:

- Declarative, permission-based protection for ADF security-aware resources, such as ADF bounded task flows, top-level web pages that use ADF bindings, and attributes defined by ADF entity objects and their attributes.
- Dynamic user authentication. When you use ADF Security, the application dynamically prompts the user to log in if the user is not yet authenticated and tries to access a page that is not granted to the `anonymous-role` role. In the application's `web.xml` file, a security constraint is applied to the ADF authentication servlet so that login is triggered through the Java EE web container before any secured resources can be accessed. After the user successfully logs in, the ADF authentication servlet runs to verify if the authenticated user has view access to the requested page.
- Permission checking within the web page. At runtime, the security policy you define for ADF resources is enforced using standard JAAS permission authorization to determine the user's access rights. If your application requires it, you can use Expression Language (EL) to perform runtime permission checks within the web page to hide components that should not be visible to the user.
- Simplifies securing of applications by providing an abstraction layer between the application and various security providers. Calls from the application to the security layer can be made through standards-based APIs, so developers do not have to deal with implementation details of the security providers.

8.3 Key Concepts of ADF Security

The ADF Security framework consists of runtime integration with OPSS plus additional design-time features through JDeveloper that simplify the creation of secure applications. This section provides an overview of the main aspects of ADF Security, including the OPSS features it incorporates and additional ADF-specific features.

8.3.1 Authentication and Authorization

Authentication is the process of validating a user's credentials, such as through a login screen. ADF Security provides an authentication servlet which (through OPSS) delegates authentication to the Java EE web container and also allows the application to dynamically prompt the user to log in.

Authorization is the process of determining the authenticated user's access rights and permissions. ADF Security (through OPSS) offers a fine-grained, permission-based authorization model which protects a resource (such as an ADF task flow) by means of JAAS-based `checkPermission` calls. OPSS also enables you to use Java EE container-managed security, which provides a more coarse-grained authorization model.

8.3.2 Application Roles

Instead of granting access to individual users, you can group users into application roles and grant permissions to the role.

8.3.3 Security Policies

You grant users (or roles) access rights to a given resource. A security policy is an access right that you grant for a given resource. Ultimately, it is the security policy on the ADF resource that controls the user's ability to enter a task flow or view a web page.

For ease of administration, you can also create entitlement grants, under which you aggregate multiple resources in a security group. This enables you to set the security policy for multiple resources in one place.

8.3.4 Security Awareness in ADF Resources

The ADF Security framework contains permission classes that enable you to protect ADF resources. Resources for which such permission classes exist are known as *security-aware* resources. Any web page associated with an ADF security-aware resource is protected by default unless you explicitly set a security policy granting access to the resource.

For a list of the types of ADF security-aware resources, see ADF Security-Aware Resources.

8.4 Key Components of ADF Security

This section outlines the features and physical artifacts that are at the core of the ADF Security framework.

8.4.1 Design-Time Integration With OPSS

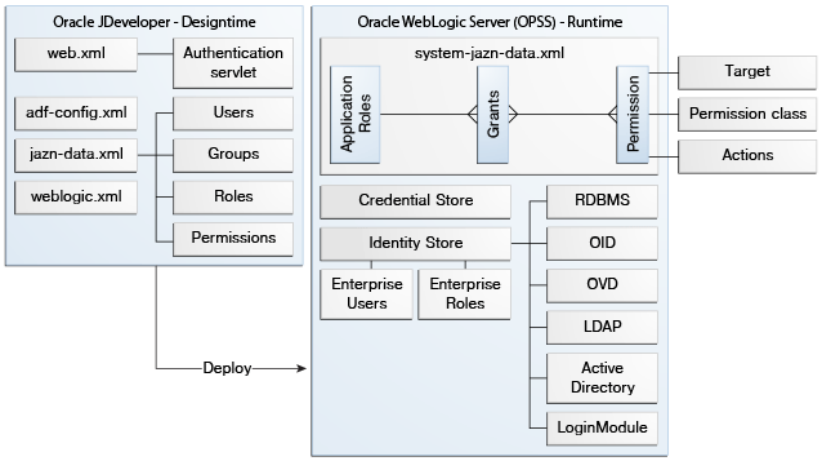
The design-time integration with OPSS for an application is configured declaratively through several metadata files, which are configured automatically when you use JDeveloper's security wizards and visual editors. The following are the key files affected:

- The user interface project's `web.xml` file, in which the following things are specified:
  - The `JpsFilter`, in order to set up the OPSS policy provider.
  - A web resource for the `oracle.adf.share.security.authentication.AuthenticationServlet` servlet and mappings to require the user to log in the first time the application is accessed and to set the appropriate security constraint to trigger user authentication dynamically.
  - The authentication method for the login configuration.
  - Required security roles, such as `valid-users`, which is used to trigger the security constraint that enables dynamic authentication.
- The application workspace's `adf-config.xml` file, in which the JAAS security context is set and the use of ADF Security security policies is enabled.
- The application workspace's `jps-config.xml` file, which holds the OPSS security platform configuration at design time. (Though this file may be deployed as part of the application EAR file, it is not used at runtime. Instead, a version of the file that is stored in the server instance's domain is used.)
- The application's `jazn-data.xml` file, which serves as the application's identity and policy store at design time. When you deploy the application to Oracle WebLogic Server, by default the server copies users and groups in `jazn-data.xml` to the server's identity store and merges policies to the server's policy store (`system-jazn-data.xml` in Integrated WebLogic Server).

  Before deploying to a production server, you should delete all users and groups that you create in the `jazn-data.xml` file to prevent the risk that malicious users could use those credentials to gain access to the application.
- The application's `weblogic.xml` file, in which the `valid-users` security role is mapped to `users`, WebLogic Server's implicit role for all users.

  Figure 8-3 shows the security artifacts that are created and configured in JDeveloper and how they map to the runtime security architecture of Oracle WebLogic Server.

Figure 8-3 ADF Security Configuration and Deployment



8.4.2 ADF Security-Aware Resources

The following types of resources in ADF applications are security-aware and can be configured with individual security policies:

- ADF bounded task flows. You can set a security policy to protect the entry point to a bounded task flow, which in turn controls the user's access to the pages contained by the flow. It is recommended that you set security policies for all bounded task flows.

  To make sure that pages contained by a bounded task flow cannot be accessed directly, you must not grant access to the contained pages through their associated page definition file. When pages require additional security within the context of a bounded task flow, wrap those pages in a sub-task flow with additional grants defined on the nested task flow.

- ADF page definition files, which contain bindings for web pages and which map to individual pages. You might need to set a security policy for a page definition file if its page is not encompassed by a bounded task flow. If you want to secure a page that does not have a corresponding page definition file, you can create an empty page definition file for the page.

- ADF Business Components entity objects and attributes of entity objects that reference rows of data and help define collections for display in the user interface. You can set permissions for the read, update, and delete operations that the entity object initiates on its data source.

  When you enable authorization for an entity object, all rows of data defined by the entity object are protected by the grant. At this level of granularity, your table component would render in the web page either with all data visible or with no data visible—depending on the user's access rights. As an alternative to securing the entire collection, you can provide security policies by individual attribute.

8.4.3 ADF Authentication Servlet

When ADF Security authorization is enabled, all user interaction with the application is mediated by the `oracle.adf.share.security.authentication.AuthenticationServlet` servlet. This servlet requires that the user successfully log in before being able to access any of the application's secured resources. It is specified as a resource in the

user interface project's `web.xml` file and referenced from filter mapping and security constraint tags.

8.4.4 ADF Security Context

Information about authenticated users can be accessed with calls to the ADF security context with Java code, Groovy expressions, or from the user interface components via EL. You can determine things such as whether the user is authenticated and whether or not a user is granted permissions for given resources. You can then use this information to determine which content and controls to display to the user. The security context is represented by the `SecurityContext` object in Java, the `securityContext` namespace in EL, and the `SecurityContext` object in Groovy.

8.5 Overview of the ADF Security Process Flow

Using the ADF Security framework consists of the following basic steps:

1. In JDeveloper, run the Configure ADF Security wizard to configure security for the application. This step enables you to set the security model, configure an authentication type for the web project, grant view access to a test-all role, and set up a welcome page for successfully authenticated users.

   It is recommended that you run this wizard early in the development cycle so that you can iteratively test security and make design decisions that best take security into account.

2. Create one or more application roles.

   Application roles you create are specific to the application and let you confer the same level of access to a set of users (also known as member users). In the test phase you create some users and add them as members to the application roles you created.

3. Set security policies to associate any ADF security-aware resources (such as bounded task flows) with one or more application roles that you have created and set the access rights for those roles.

4. Create test users for the various roles.

5. Run the application in JDeveloper and test access to the various resources using the test users that you have created.

6. Before deploying the application, remove any policy grants and users that you had added in order to test the application.

7. Migrate the finalized policy store and credentials store to the target server. Application policies and credentials can be automatically migrated to the domain stores when the application is deployed to Oracle WebLogic Server.

8.6 Learning More About ADF Security

For more information about using the ADF Security framework and OPSS, see the following.

- Enabling ADF Security in a Fusion Web Application in *Developing Fusion Web Applications with Oracle Application Development Framework*

The following resources provide more information about OPSS.

- Introduction and Roadmap in *Understanding Security for Oracle WebLogic Server*
- Introduction and Roadmap in *Administering Security for Oracle WebLogic Server*
- Introduction and Roadmap in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*
- The Oracle Platform Security Services page on OTN: `http://www.oracle.com/technetwork/middleware/id-mgmt/index-100381.html`

9

Oracle Metadata Services

This chapter provides a high-level overview of how the Oracle Metadata Services (MDS) framework can be used for seeded customizations and change persistence in an ADF application.

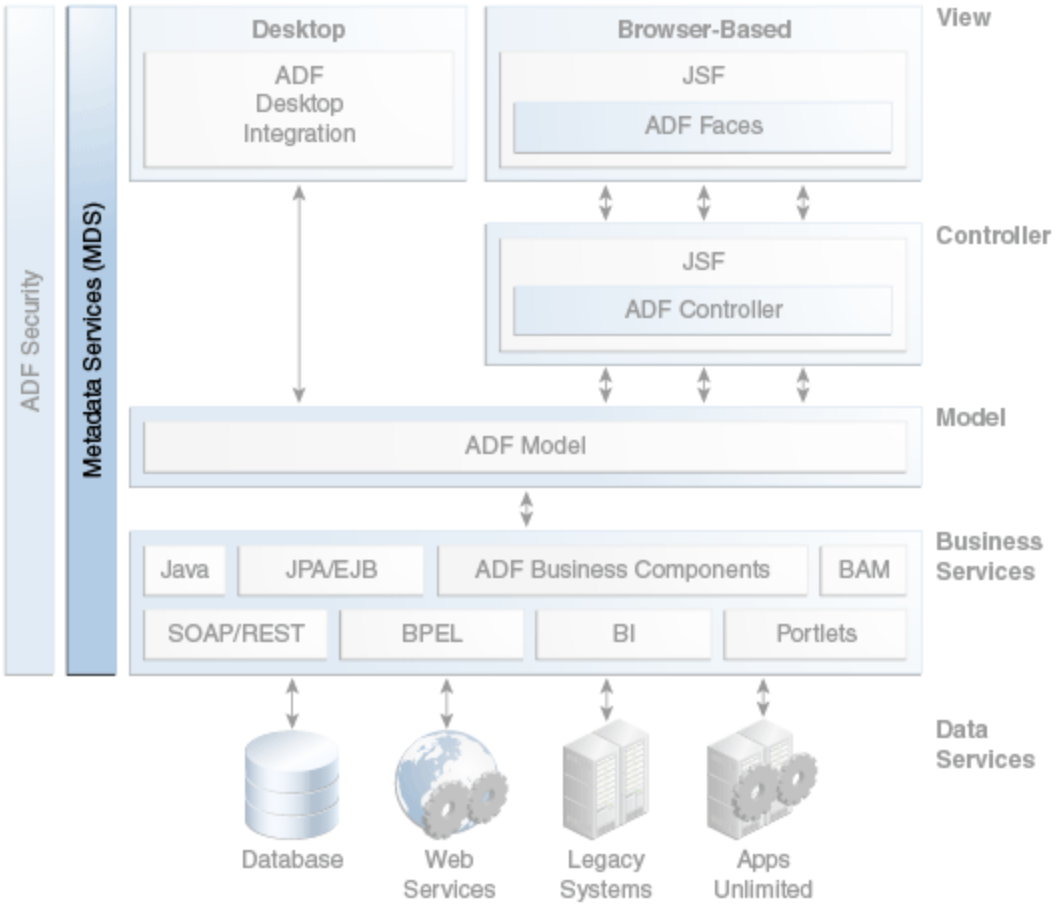
This chapter includes the following sections:

9.1 About Oracle Metadata Services (MDS)

The Oracle Metadata Services (MDS) framework allows you to create applications that your customers can further customize for their users or customers and which the end users can also customize.

As shown in Figure 9-1, MDS can be applied to the range of other components in the ADF technology stack, such as ADF Faces, ADF Controller, ADF Model, and ADF Business Components.

Figure 9-1 ADF Architecture with Metadata Services



You can use MDS to enable the following types of customizations in an application:

- Seeded customization

  Seeded customization of an application is the process of taking a generalized application and making modifications to suit the needs of a particular group, such as a specific industry or site. Seeded customizations exist as part of the deployed application, and endure for the life of a given deployment.

- User customization (change persistence).

  User customization allows an end user to change the content of the application at runtime to suit individual preferences (for example, which columns are visible in a table), and have those changes persist across that user's sessions.

9.2 Core Benefits of MDS

The architecture and features of MDS provide the following benefits:

- You can offer your customers a working application that they can further customize as they see fit.
- A single application can have different customizations for different users or user segments.
- The application can be patched or updated without changing or removing the customizations.

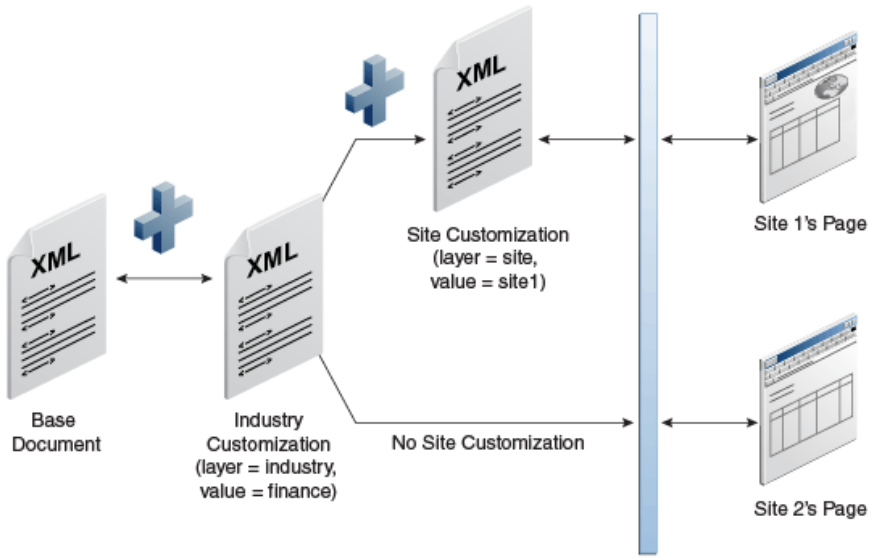
9.3 Key Concepts of MDS

The following concepts are central to the understanding of MDS:

- Customization layers and customization values.

  A customized application contains a base application (the base documents) and one or more layers containing customizations, as illustrated in Figure 9-2. Each layer can have multiple customization layer values, typically only one of which is applied at runtime. When a customized application is launched, the customization content is applied over the base application.

  Figure 9-2 Base Application and Customization Layers

  Examples of customization layers are `industry` and `site`. Examples of values for the `industry` layer might be `financial` and `healthcare`.

  Since the customizations are saved separately from the base, the customizations are upgrade safe; a new patch to the base can be applied without breaking customizations.

- Static and dynamic customizations.

  Customizations can be categorized as either static or dynamic. Static customizations have only one layer value in effect for all executions of the application, while dynamic customizations can have values that vary based on the execution context (such as the user) of the application.

  Customizations in ADF Business Components objects and data control objects for other business services can only be static. This is because these objects are loaded only once for an application and reused for the duration of the application.

  Customizations at the controller or view level can be either static or dynamic, since they can allow the layer value to be determined at runtime, based on user roles (responsibilities) or other application-specific criteria. For example, you can design an application so that users from different organizations see different sets of fields on a given screen.

- Customization Developer Role

JDeveloper provides a special development role for customization developers so that work in customization layers of an application is isolated from the base application code. When you switch JDeveloper to the Customization Developer role, only customizable parts of the application are editable and new objects cannot be created.

- Change Persistence

  ADF Faces incorporates a change persistence feature, which enables users to make changes to UI components, such as selecting which columns to display in a table, and have those changes persist as long as the session is active. Using MDS, you can extend the change persistence features to work across sessions, so that a user can exit the application with the UI in a given state and then restart the application and see the UI in the state that it was previously.

9.4 Key Components of MDS

The following are the key components of MDS:

- MDS metadata repository.

  MDS stores the customizations in a metadata repository and retrieves them at runtime to merge the customizations with the base metadata to reveal the customized application.

  An MDS repository can be file-based or database-based. For production environments, a database-based MDS repository is preferable because of advantages such as efficient set-based queries, atomic transaction semantics, versioning, and the ability to isolate and test metadata changes on selected users in a running environment. For more information on setting up a metadata repository, see Managing the Metadata Repository in *Administering Oracle Fusion Middleware*.

- Customization classes.

  A customization class is the interface that MDS uses to define which customization applies to the base definition metadata. Each customization class defines a customization layer (for example, `industry` or `site`) and can contain multiple layer values.

- Metadata Archive (MAR).

  A MAR file is a compressed archive of selected metadata that is used to deploy an application's customizations to the MDS Repository.

9.5 Overview of the MDS Process Flow

Using MDS in an application consists of the following general steps:

1. Create the customization classes that will be used and make them available to your application at design time in JDeveloper.
2. Enable seeded customization for any pages or other artifacts that you want to make customizable.
3. Specify the customization classes in the `adf-config.xml` file
4. Optionally, set any restrictions on runtime customizations on the application.

5. Optionally, enable end-user customizations and specify which components and which of their properties can be customized. In addition, you can set page-specific customization configurations and implement programmatic customizations for things such as reordering child components.
6. Specify the customization layers and their values in the `CustomizationLayerValues.xml` file (to determine the layer values to make available to the customization developer in the Customization Context window).
7. In JDeveloper, using the Studio Developer or the Customization Developer role, implement any seeded customizations that you want to add.
8. If it has not already been done, configure an MDS repository on the server on which the customizable applications will be deployed.
9. Package the customizations in a MAR file and include that MAR as part of the application's EAR file that you deploy to the application server.

9.6 Learning More About MDS

The following resources provide detailed information about using MDS:

- Customizing Applications with MDS in *Developing Fusion Web Applications with Oracle Application Development Framework*
- Allowing User Customizations at Runtime in *Developing Fusion Web Applications with Oracle Application Development Framework*
- Allowing User Customization on JSF Pages in *Developing Web User Interfaces with Oracle ADF Faces*
- Managing the Metadata Repository in *Administering Oracle Fusion Middleware*

10

Deployment of Applications Containing Oracle ADF Features

This chapter describes the concepts behind and the high-level process of deploying Oracle ADF applications to an application server.

This chapter includes the following sections:

10.1 About Deployment of Applications that Contain Oracle ADF Features

Deployment is the process of packaging application files as an archive file and transferring that file to a target application server. You can use JDeveloper to deploy ADF applications directly to the application server or indirectly to an archive file as the deployment target, and then install this archive file to the target server.

10.2 Key Concepts of Deploying ADF Applications

This section outlines key aspects of the process of preparing and completing deployment of ADF applications.

10.2.1 Test Deployment with Integrated WebLogic Server

For test running an application during development, you can use JDeveloper to run an application in Integrated WebLogic Server. When running an application in the Integrated WebLogic Server, you do not have to manually complete many of the steps that are necessary for deployment to a standalone server. For example, you do not have to create a deployment profile or an EAR file, and the ADF Runtime libraries are automatically included in the Integrated WebLogic Server instance.

10.2.2 Deployment Tools

For deployment to standalone servers, you can choose from among the following tools and approaches:

- Oracle Enterprise Manager Fusion Middleware Control
- Scripting tools that are specific to a given application server, such as WebLogic Scripting Tool (WLST)

- Administration consoles, such as Oracle WebLogic Administration Console
- Command scripts and Ant scripts
- JDeveloper

10.2.3 Test Deployment on a Standalone Server

Before you deploy your application to a production server, you may wish to test the application on a standalone server instance that is not in a production environment. Doing so enables you to make sure that the application deploys as expected in a remote environment. This step is recommended to help identify and fix potential problems in the deployment that otherwise would not be revealed by running the application in an Integrated WebLogic Server instance, where most of the steps are automated to benefit the development workflow and are not geared toward a production server environment.

10.3 Key Components of Deploying ADF Applications

This section outlines the key elements that are necessary for deploying an ADF application to an application server.

10.3.1 Enterprise Archive (EAR) File

ADF applications are generally deployed to application servers as EAR files. EAR files are composite archives that contain one or more other archives, such as WAR, JAR, and MAR files, and one or more deployment descriptors.

10.3.2 ADF Runtime Libraries

Applications that use any ADF components or features need ADF Runtime libraries installed on the application server in order to run. Such components and features includes ADF Business Components, ADF Model data binding, ADF task flows, or ADF Faces components.

10.4 Overview of the ADF Application Deployment Process Flow

Once your application is developed, you follow the steps below to deploy the application to a standalone server. These steps do not include running the application in Integrated WebLogic Server from JDeveloper.

1. Create *deployment profiles* to define the way the application contents are packaged into archive files that will be deployed to the target environment. In addition to specifying the format and contents of the archive file, a deployment profile includes dependency information, platform-specific instructions, and other information.

2. Create or edit the necessary *deployment descriptors* for the target server. Deployment descriptors are XML server configuration files that define the configuration of an application for deployment and that are deployed with the application as needed.

   When developing the application in JDeveloper, the necessary deployment descriptor files for Oracle WebLogic Server are generated. JDeveloper also provides visual editors for these files that you can use to view and edit properties.

3. Prepare the application's security policies and credentials for migration to the standalone server. This includes ensuring that any policies and credentials that you have set up for testing purposes are removed from the application's configuration files and setting up application roles that map to standard roles on the target server.

4. In the user interface project's `web.xml` file, register any ADF MBeans that you want to use.

   ADF MBeans correspond to various configuration files. After the application has been deployed, you can change configuration properties by accessing the ADF MBeans using the Enterprise Manager Fusion Middleware Control MBean browser.

5. In JDeveloper, generate an EAR file from the deployment profile.

6. Set up a standalone instance of the target application server for test deploying and install the ADF runtime in that instance.

7. Migrate the application's policy store to the domain level on the standalone server instance.

   You typically handle the migration task outside of JDeveloper using tools like Oracle Enterprise Manager Fusion Middleware Control. For details about using tools outside of JDeveloper to migrate the policy store to the domain-level in a standalone environment, see Migrating from a Test to a Production Environment in *Securing Applications with Oracle Platform Security Services*.

8. Test deploy the application using any of the approaches cited in Deployment Tools and fix any problems that arise.

9. On the target server, make sure that the ADF runtime libraries are installed and add the application's policy store.

10. Deploy the application to the target server.

10.5 Learning More About Deploying ADF Applications

For more information on preparing applications for deployment, see Deploying Fusion Web Applications in *Developing Fusion Web Applications with Oracle Application Development Framework*.

For more information on setting up standalone servers on which to deploy applications, deploying the applications, and then configuring and managing the deployed applications, see Part II Basic Administration in *Administering Oracle ADF Applications*.

For more information on migrating security policies and credentials to the target server, see Migrating from a Test to a Production Environment in *Securing Applications with Oracle Platform Security Services*.

For more information on deploying Oracle ADF Essentials applications to GlassFish see Deploying ADF Applications to GlassFish in *Developing Fusion Web Applications with Oracle Application Development Framework*.

Part V

Appendix

This part contains the following appendix:

A

ADF Business Components and Familiar 4GL Tools

This appendix compares key components in ADF Business Components to conceptually similar functionality offered by the enterprise 4GL tools PeopleTools, Siebel Tools, and ADO.NET.

This appendix includes the following sections:

A.1 Comparison to PeopleTools

If you have developed solutions in the past with PeopleTools, you are familiar with the PeopleTools component structure. ADF Business Components implement the data access functionality you are familiar with from PeopleTools.

Table A-1 Familiar Concepts for PeopleTools Developers

This concept	Maps to Oracle ADF	With these similarities
Headless Components	Application Modules	Oracle ADF adheres to an MVC pattern and separates the model from the view. Pages, which you are familiar with in the PeopleTools Component, are defined in the view layer, using standard technologies like JSF and ADF Faces components for web-based applications or Swing for desktop-fidelity client displays. The ADF application module defines the data structure, just like the PeopleTools Component Buffer does. By defining master-detail relationships between ADF query components that produce row sets of data, you make sure that any application module that works with the data can reuse the natural hierarchy as required, similar to the scroll levels in the Component Buffer. Similar to the Component Interface you are familiar with, the application module is a service object that provides access to standard methods, as well as additional developer-defined business logic. In order to present a "headless" data service for a particular user interface, the Component Interface restricts a number of PeopleTools functions that are related to UI interaction. The application module is similar to the Component Interface in that it provides a "headless" data service, but in contrast it does not do this by wrapping a restricted view of an existing user interface. Instead, the application module is designed to deal exclusively with business logic and data access. Rather than building a Component Interface on top of the component, with ADF Business Components you first build the application module service that is independent of a user interface, and then build one or more pages on top of this service to accomplish some end-user task in your application. The application module is associated with a transaction object in the same way that the PeopleTools Component Buffer is. The application module also provides a database connection for the components it contains. Any logic you associate today with the transaction as Component PeopleCode, in ADF Business Components you would define as logic on the application module. Logic associated with records in the transaction, that today you write as Component Record PeopleCode or Component Record Field PeopleCode, should probably *not* be defined on the application module. ADF Business Components has view objects that allow for better re-use when the same record appears in different components. In summary, PeopleTools uses the component for the container concept, whereas ADF Business Components uses the application module. That is where the similarity ends. Do not assume that all of your component code will migrate to an application module. First, understand the concept of the view object, which is the layer between the entity object and the application module. Then, decide which of your component code is suitable for an application module and which is suitable for view objects.

Table A-1 (Cont.) Familiar Concepts for PeopleTools Developers

This concept	Maps to Oracle ADF	With these similarities
Record Definition	Entity Object	The entity object is the mapping to the underlying data structure, just like the PeopleTools Record Definition maps to the underlying table or view. You'll often create one entity object for each of the tables that you need to manipulate your application. Similar to how you declare a set of valid values for fields like "Customer Status" using PeopleTools' translate values, in ADF Business Components you can add declarative validations to the individual attributes of an entity object. Any logic you associate with the record that applies throughout your applications, which today you write as Record PeopleCode or Record Field PeopleCode, can be defined in ADF Business Components on the entity object.
Row Set	View Object	Just like a PeopleTools row set, a view object can be populated by a SQL query. Unlike a row set, a view object definition can contain business logic. Any logic which you would find in Component Record PeopleCode is a likely candidate to define on the view object. Component Record PeopleCode is directly tied to the component, but a view object can be associated with different application modules. Whereas you can use the same record definition in many PeopleTools components, Oracle ADF allows you to reuse the business logic across multiple applications. The view object queries data in exactly the "shape" that is useful for the current application. Many view objects can be built on top of the same entity object. You can define relationships between view objects to create master-detail structures, just as you find them in the scroll levels in the PeopleTools component.

A.2 Comparison to Siebel Tools

If you have developed solutions in the past with Siebel Tools version 7.0 or earlier, you will find that ADF Business Components implements all of the familiar data access functionality you are familiar with, with numerous enhancements.

Table A-2 Familiar Concepts for Siebel Tools Developers

This concept	Maps to Oracle ADF	With these similarities
Table Object	Entity Object	Like the Siebel Table object, the ADF entity object describes the physical characteristics of a single table, including column names and physical data types. Both objects contain sufficient information to generate the DDL (data definition language) statements to create the physical tables in the database. In ADF Business Components you define associations between entity objects to reflect the foreign keys present in the underlying tables. These associations allow view object queries used by user interface pages to automatically join business information. ADF Business Components handles list of values (LOV) objects that you reference from data columns through a combination of declarative entity-level validation rules and view object attribute-level LOV definitions. You can also encapsulate other declarative or programmatic business logic with these entity object "table" handlers that is automatically reused in any view of the data you create.
Business Component	View Object	Like the Siebel Business Component, the ADF view object describes a logical mapping on top of the underlying physical table representation. Both the Siebel Business Component and the ADF view object allow you to provide logical field names, data, and calculated fields that match the needs of the user interface. As with the Siebel Business Component, with the ADF view object you can define view objects that join information from various underlying tables. The related ADF view link is similar to the Siebel Link object and allows you to define master-detail relationships. In ADF Business Components, your view object definitions can exploit the full power of the SQL language to shape the data as required by the user interface.
Business Object	Application Module	The Siebel Business Object lets you define a collection of business components. The ADF application module performs a similar task, allowing you to create a collection of master-detail view objects that act as a "data model" for a set of related user interface pages. In addition, the application module provides a transaction and database connection context for this group of data views. You can make multiple requests to objects obtained from the application module and these participate in the same transaction.

A.3 Comparison to ADO.NET

If you have developed solutions in the past with Visual Studio 2003 or 2005, you are familiar with using the ADO.NET framework for data access. ADF Business Components implements all of the data access functionality you are familiar with from ADO.NET, with numerous enhancements.

Table A-3 Familiar Concepts for ADO.NET Developers

This concept	Maps to Oracle ADF	With these similarities
Data Set	Application Module	The application module component plays the same role as the ADO.NET data set. It is a strongly typed service component that represents a collection of row sets called view object instances, which are similar to ADO.NET data tables. An application module exposes a service interface that surfaces the rows of data in a developer-configurable set of its view instances as an SDO-compatible service (accessible as a web service, or as an SCA composite). The application module works with a related transaction object to provide the context for the SQL queries that the view objects execute. The application module also provides the context for modifications saved to the database by the entity objects, which play the role of the ADO.NET data adapter.
Data Adapter	Entity Object	The entity object component is like a strongly typed ADO.NET data adapter. It represents the rows in a particular table and handles the find-by-primary-key, insert, update, delete, and lock operations for those rows. In ADF Business Components, you don't have to specify these statements yourself, but you can override them if you need to. The entity object encapsulates validation or other business logic related to attributes or entire rows in the underlying table. This validation is enforced when data is modified and saved by the end user using any view object query that references the underlying entity object. One difference in ADF Business Components is that the arbitrary, flexible querying is performed by SQL statements at the view object instance level, but the view objects and entity objects coordinate automatically at runtime.
Data Block	View Object	The view object component performs the "data retrieval" portion of the data block functionality. Each view object encapsulates a SQL query, and at runtime each one manages its own query result set. If you connect two or more view objects in master-detail relationships, that coordination is handled automatically. While defining a view object, you can link any of its query columns to underlying entity objects. By capturing this information, the view object and entity object can cooperate automatically for you at runtime to enforce your domain business logic, regardless of the "shape" of the business data required by the user's task.