

Oracle® Fusion Middleware

Developing Swing Applications for Oracle Application
Development Framework

12c (12.1.3)

E41267-01

May 2014

Describes how to create and deploy Swing desktop
applications using ADF Swing and Oracle Application
Development Framework (Oracle ADF).

Oracle Fusion Middleware Developing Swing Applications for Oracle Application Development Framework 12c (12.1.3)

E41267-01

Copyright © 2013, 2014 Oracle and/or its affiliates. All rights reserved.

Primary Author: Ralph Gordon

Contributing Author: David Mathews, Cindy Hall

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

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

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Contents

Preface	vii
Audience	vii
Documentation Accessibility	vii
Related Documents	vii
Conventions	viii
1 Introduction to ADF Swing Applications	
1.1 About ADF Swing	1-1
1.1.1 Advantages of Using ADF Swing	1-1
1.1.2 ADF Swing Architecture	1-2
1.1.2.1 Swing MVC	1-2
1.1.2.2 Oracle Application Development Framework	1-3
1.2 Creating a Desktop Application That Works with Oracle ADF	1-3
1.3 What Happens When You Create a Desktop Application with ADF Swing	1-4
1.4 Connecting to Business Components	1-5
1.4.1 How to Modify the Configuration Name	1-5
1.4.2 How to Modify the Configuration File	1-6
1.4.3 What You May Need to Know About the Client Application Libraries	1-6
2 Creating ADF Swing Forms and Panels	
2.1 About Creating ADF Swing Forms and Panels	2-1
2.1.1 ADF Swing Design Time Wizards	2-2
2.1.2 A Typical ADF Swing Form	2-2
2.1.3 Navigation in an ADF Swing Form	2-3
2.2 Process for Creating ADF Swing Panels and Forms	2-4
2.3 What Happens When You Create an ADF Swing Form	2-5
2.4 What You May Need to Know About ADF Swing Code Generation	2-5
2.5 What You May Need to Know About Business Components Attribute Settings	2-6
2.6 How to Create a Client Data Model Definition	2-6
2.7 How to Create a Single Table ADF Swing Form	2-7
2.8 How to Create a Master-Detail ADF Swing Form	2-8
2.9 How to Create an Empty ADF Swing Form	2-9
2.10 How to Create an Empty ADF Swing Panel	2-10
2.11 How to Create ADF Swing Edit Forms from the Data Controls Panel	2-11
2.12 How to Create ADF Swing Forms from the Databases Window	2-12

3 Modifying ADF Swing Forms and Panels

3.1	About Modifying ADF Swing Forms and Panels	3-1
3.1.1	Value Bindings for the Entire Collection or Data Object	3-1
3.1.2	Value Bindings for Individual Data Object Attribute Values	3-2
3.2	How to Assemble ADF Swing Forms Using the Java Visual Editor	3-3
3.3	How to Insert UI Components into ADF Swing Panels	3-4
3.4	How to Change Client Data Model References	3-5
3.5	How to Open an ADF Swing Form with an Action Handler	3-6
3.6	How to Drop Data Panels Onto an Empty ADF Swing Form	3-7
3.7	How to Lay Out Data Panels in an Empty Swing Form	3-7
3.8	Binding a Method with Parameters in an ADF Swing Form	3-7
3.8.1	How to Populate the Data Controls Panel with JavaBean Methods	3-8
3.8.2	How to Create an ADF Swing Form with Method Bindings	3-8
3.8.3	What You May Need to Know About Displaying a Method Result Using a JTable Component	3-9

4 Working with Data Binding

4.1	About Working With Data Binding	4-1
4.1.1	ADF Swing Containers	4-2
4.1.2	Standard Java Containers	4-2
4.2	Navigating the UI Using ADF Swing Controls	4-2
4.2.1	How to Navigate Using the Navigation Bar	4-3
4.2.2	How to Navigate Using Tree Navigation	4-3
4.3	What You May Need to Know About the ADF Swing Data Context	4-4
4.4	What Happens at Runtime: How Panel Bindings Function	4-4
4.5	What You May Need to Know About the ADF Swing Bootstrap Code	4-5
4.6	How to Display Object Attributes in a Databound Text Field	4-6
4.7	How to Create a New Row in a Databound Table or Tree Control	4-6
4.8	How to Sort Columns in a Databound Table	4-7
4.9	What Happens At Runtime: How Control Bindings Function	4-8
4.9.1	Populating Controls with Data	4-8
4.9.2	Updating Data through Controls	4-8

5 Customizing ADF Bindings

5.1	About Customizing ADF Bindings	5-1
5.2	How to Customize ADF Bindings for ADF Swing Panels	5-2
5.3	How to Customize an ADF Action Binding	5-3
5.4	How to Customize an ADF Attribute Binding	5-5
5.5	How to Customize an ADF Array Combobox Binding	5-6
5.6	How to Customize an ADF Boolean Binding	5-7
5.7	How to Customize an ADF Bounded Range Binding	5-8
5.8	How to Customize an ADF Formatted Text Field Binding	5-10
5.9	How to Customize an ADF Iterator Binding	5-11
5.10	How to Customize an ADF List Binding	5-12
5.11	How to Customize an ADF List Binding in Enumeration Mode	5-13
5.12	How to Customize an ADF List Binding in LOV Mode	5-14

5.13	How to Customize an ADF LOV Button Binding	5-17
5.14	What You May Need to Know About the LOV Dialog	5-19
5.15	How to Customize an ADF Scroll Binding	5-20
5.16	How to Customize an ADF Table Binding	5-21
5.17	How to Customize an ADF Tree Binding	5-23

6 Displaying Graphs in ADF Swing Panels

6.1	About Graphs in ADF Swing Panels	6-1
6.2	How to Create a Graph for an ADF Swing Panel	6-3
6.3	What Happens When You Create a Graph Component	6-5
6.4	How to Customize the Graph Component	6-7
6.5	How to Change Graph Data	6-7

7 Working with ADF Swing Controls

7.1	About ADF Swing-Specific Controls	7-1
7.2	How to Use the JUArrayComboBox Control	7-2
7.3	How to Use the JUImage Control	7-3
7.4	What You May Need to Know About Multimedia in ADF Swing Applications	7-4
7.5	How to Use the JULabel Control	7-4
7.6	How to Use the Label For Control	7-5
7.7	How to Use the JULovEditButton Control	7-6
7.8	How to Use the JUNavigationBar Control	7-6
7.9	How to Use the JUNavigationBar Control with Find Mode	7-7
7.10	How to Disable Find Mode for ADF Swing Controls in a Panel	7-8
7.11	What You May Need to Know About Iterator Bindings in Find Mode	7-8
7.12	How to Use the JURadioButtonGroupPanel Control	7-10
7.13	How to Use the JUShuttlePanel Control	7-10
7.14	How to Use the JUStatusBar Control	7-11

8 Using Validation in the ADF Swing User Interface

8.1	About Validating Events	8-1
8.2	How to Use Validation With ADF Control Bindings	8-1
8.3	How to Use Validation With ADF Swing Panels	8-2

9 Working with an ADF Swing Login Dialog

9.1	About the ADF Swing Login Dialog	9-1
9.2	How to Create a Login Dialog	9-2
9.3	How to Run the Application Using the Login Dialog	9-2
9.4	How to Run the Application Without the Login Dialog	9-3
9.5	What You May Need to Know About Customizing the Login Dialog Code	9-3
9.6	How to Modify the Login Dialog to Work with a JDBC Connection	9-4

10 Optimizing ADF Swing Application Runtime Performance

10.1	About Optimizing ADF Swing Application Runtime Performance	10-1
10.2	How to Delay Updates to ADF Business Components from ADF Swing	10-1

10.3	What You May Need to Know About the Sync Mode Property	10-3
10.4	How to Limit Fetching of ADF Business Components Attributes in ADF Swing	10-3

11 Using Java Web Start With ADF Swing Applications

11.1	About Working with Java Web Start	11-1
11.1.1	Java Web Start Technology	11-2
11.1.2	Java Web Start and Integrated WebLogic Server	11-2
11.1.3	Java Web Start and Oracle WebLogic Server	11-3
11.2	How to Define ADF Business Components Runtime Properties	11-3
11.3	How to Set Up Runtime Configuration Information	11-4
11.4	How to Create a Java Web Start JNLP Definition	11-5
11.5	What Happens When You Create a JNLP Definition	11-6
11.6	How to Run ADF Swing Applications with Java Web Start in JDeveloper	11-7

Preface

Welcome to *Developing Swing Applications with Oracle Application Development Framework*.

Audience

This document is intended for enterprise developers who need to create and deploy database-centric desktop applications using the Oracle Application Development Framework (Oracle ADF). This guide explains how to build ADF applications that display databound Java forms using ADF Business Components and ADF Swing.

Documentation Accessibility

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

Access to Oracle Support

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

Related Documents

For more information, see the following documents:

Understanding the Oracle Application Development Framework

Developing Fusion Web Applications with Oracle Application Development Framework

Developing Applications with Oracle JDeveloper

Developing Applications with Oracle ADF Data Controls

Developing Applications with Oracle ADF Desktop Integration

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 Lifecycle

Java API Reference for Oracle ADF Share

Java API Reference for Oracle ADF Model Tester

Generic Domains Java API Reference for Oracle ADF Business Components

interMedia Domains Java API Reference for Oracle ADF Business Components

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.
<i>italic</i>	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
<code>monospace</code>	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Introduction to ADF Swing Applications

This chapter describes ADF Swing technology and ADF Swing architecture.

This chapter includes the following sections:

- [Section 1.1, "About ADF Swing"](#)
- [Section 1.2, "Creating a Desktop Application That Works with Oracle ADF"](#)
- [Section 1.3, "What Happens When You Create a Desktop Application with ADF Swing"](#)
- [Section 1.4, "Connecting to Business Components"](#)

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

1.1 About ADF Swing

ADF Swing is a technology for developing databound Java clients that simplifies coding the interaction between Swing components and business services. ADF Swing consists of the following:

- Java classes and an API for binding Swing components to business services
- XML data definition and configuration files
- Design-time tools, including wizards, for creating databound Java forms
- Several user interface components which extend certain Swing components

In Oracle Application Development Framework (Oracle ADF), you can use ADF Swing to work with a number of business services on the back end, including ADF Business Components, Enterprise Java Beans, and web services.

JDeveloper provides tools and wizards to enable your development with ADF Swing, a technology for developing databound Java clients. These databound Java clients simplify coding the interaction between Swing components and business services in an Oracle ADF application.

In addition, JDeveloper also provides several ADF Swing-specific editing tools, including the ADF Swing wizards to create and edit Swing panels and forms.

1.1.1 Advantages of Using ADF Swing

The advantages of using ADF Swing to build Java Swing clients include the following:

- Wizards create ADF Swing forms quickly
- Binding to data sources is supported for any model-based controls, including:

- Standard Swing components
- JDeveloper-provided ADF Swing components
- Third party model-based add-in components
- XML data definitions provide for easy reuse of ADF Swing frames and panel.
- Remote methods from the model layer ADF Business Components are available to the Java client through direct ADF bindings.

Desktop applications using ADF Swing and ADF Business Components do not need to implement data access and update logic. ADF Swing and ADF Business Components cleanly separate data access code from UI code resulting in thin clients without the burden on the view layer. Additionally, data access is improved with ADF Swing because its binding to ADF Business Components allows it to take advantage of the numerous performance features implemented in ADF Business Components. And because ADF Swing relies on the model-view-controller architecture, designing ADF Swing forms is no different than working with Swing components.

Your Java client code is further simplified because you'll never need to change the way you access ADF Business Components, regardless of how they are deployed. Instead, features of ADF Business Components let desktop applications connect to application modules through a simple configuration definition file. The Java client code remains unchanged for any deployment scenario, whether:

- ADF Business Components are deployed locally in the same VM as ADF Swing
- ADF Business Components are deployed remotely using EJB

1.1.2 ADF Swing Architecture

ADF Swing is the technology in Oracle ADF that facilitates building databound Java clients using Swing components. The ADF Swing API consists of a set of Java classes that take advantage of features in Oracle ADF to build a Java UI that is bound to back-end business services. Oracle ADF helper classes handle the communication between the client and the business services.

1.1.2.1 Swing MVC

ADF Swing architecture is based on a Model-View-Controller (MVC) pattern. With MVC there are three communication objects logically separated for each component:

- The Model represents the data or state of the component and is its underlying logical representation.
- The View is the component's visual representation, which describes how it looks (for example, whether it is a button or some other control, whether it uses text or icons, or what border and color it uses).
- The Controller specifies the interaction with the client (how to interpret user input). The controller notifies registered listeners when the user types text, clicks a button, tabs to the next field, and so forth.

For example, a `JCheckBox` is a Swing component which has a defined Model, View, and Controller. When the user interacts with the controller by clicking the checkbox, the controller notifies the model that it should change its state (from false to true or the reverse). The view, which is listening for changes in the state of the model, can then update itself (for example, by making the checkbox appear selected). An important point about this architecture is that the model is not aware of the view or views displaying it, nor of the controller(s) being used to update it.

The Swing API lets you set the model for every component using the component's `model` or, in some cases, `document` property. In Swing, the model for any subclass of `JTextComponent` is named *document* which is accessed using the `setDocument()` and `getDocument()` methods. The standard Swing `JLabel` component does not represent data and therefore does not follow the MVC architecture. However, ADF Swing provides a `JULabel` component to overcome this limitation when you want to assign labels using business component data.

1.1.2.2 Oracle Application Development Framework

When you develop a desktop application using ADF Swing as the client technology, you take advantage of the Java EE and Model-View-Controller architecture of Oracle ADF.

For more information about Oracle ADF, see "Introduction to Building Fusion Web Applications with Oracle ADF" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

1.2 Creating a Desktop Application That Works with Oracle ADF

You can create Java desktop applications that rely on standard Swing components and obtain the advantages of Oracle ADF in your application. In this document, the ADF Java desktop application is called the ADF Swing application. When you create ADF Swing applications in JDeveloper:

- You can work with the ADF Swing wizards in the JDeveloper New Gallery to quickly generate databound forms and panels.
- You can work with the Data Controls panel to quickly add databound Swing components to your ADF Swing forms and panels.

After you generate ADF Swing forms and panels, you can proceed to customize the appearance of your forms using the Java visual editor.

To create an ADF Swing application:

1. Choose **File > New > From Gallery** from the JDeveloper menu.
2. In the New Gallery, expand **General** category and double-click **Java Desktop Application** from the **Items** list.
3. Use the Create Java Desktop Application wizard to name the application and the project.

This creates an ADF Swing application that will include ADF Business Components for the business services in the data model project. For more information, see "Getting Started with ADF Business Components" in *Developing Fusion Web Applications with Oracle Application Development Framework*, "Getting Started with ADF Business Components".

Note: To avoid application errors, it is necessary to develop the business services and Java client application in separate project folders. The JDeveloper application template will create separate project folders in your workspace.

4. In the Applications window, select the data model project and choose **File > New > From Gallery**.

5. In the New Gallery, expand **Business Tier - ADF Business Components** and double-click **Business Components from Tables** from the **Items** list.
6. Use the Initialize Business Components Project dialog to create a connection to the database that contains the tables that you want to base your business components on.

For more information, see "How to Initialize the Data Model Project With a Database Connection" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

7. Use the Create Business Components from Tables wizard to populate the business components in your data model project.

For more information, see "How to Create Multiple Entity Objects and Associations from Existing Tables" in *Developing Fusion Web Applications with Oracle Application Development Framework*.

8. In the Applications window, select the user interface project and from the main menu, choose **File > New > From Gallery**.
9. In the New Gallery, expand **Client Tier** and expand **ADF Swing** and then double-click **Empty Form** or **Empty Panel**.
10. Use the ADF Swing wizard or dialog to add the ADF Swing form or panel to your user interface project.

The file opens in the Java visual editor. For more information, see [Chapter 2, "Creating ADF Swing Forms and Panels."](#)

Note: You must use the ADF Swing wizards to generate .java files with the necessary bootstrap code. Do not use a generic Java panel or class to design databound Java clients.

11. In the Applications window, expand the Data Controls panel and use it to insert databound UI components into the open document.
For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)
12. (Optionally) Use the Data Controls panel to insert business service actions into the open document.
For more information, see [Section 3.8, "Binding a Method with Parameters in an ADF Swing Form."](#)
13. Define the ADF bindings in their corresponding binding editors to specify the required properties of the binding.
For more information, see [Chapter 5, "Customizing ADF Bindings."](#)

1.3 What Happens When You Create a Desktop Application with ADF Swing

In an ADF Swing application, data binding between the Swing controls and the business services' data sources relies on the creation a set of ADF Swing objects that closely resemble the UI containers used to assemble the ADF Swing forms. You can see these containers and their ADF Swing-specific code when you use the ADF Swing Form wizard to generate a complete application. For example, assuming a

master-detail type form, based on a Dept and Emp view object, the wizard generates the following classes:

- FrameDeptViewEmpView1 -- extends ADF SwingFrame (a dummy implementation of the ADF SwingPanel interface)
- MDPanelDeptViewEmpView1 -- extends JPanel and implements ADF SwingPanel
- PanelDeptView -- extends JPanel and implements ADF SwingPanel
- PanelEmpView1 -- extends JPanel and implements ADF SwingPanel

where JPanel is a Swing class, and ADF SwingFrame and ADF SwingPanel are part of ADF Swing and constitute your application's data browsing panels.

The resulting project files, together with the model reference in the ADF Swing panel or form shown in [Example 1-1](#), permit the databound UI components to access the ADF model layer at runtime.

Example 1-1 Model Reference in an ADF Swing Form

```
(panel.setBindingContext(JUnitTestFrame.startTestFrame("DataBindings.cpx",
    "null", panel, panel.getPanelBinding(), new Dimension(400, 300)));)
```

For more information, see to [Section 4.1, "About Working With Data Binding."](#)

The DataBindings.cpx file maps individual pages to page definition files and declares usages of the data control defined in the DataControls.dcx file. The .cpx file defines the Oracle ADF binding context for the entire application and provides the metadata from which the Oracle ADF binding objects are created at runtime.

1.4 Connecting to Business Components

ADF Swing applications use ADF Business Components to connect to deployed business service. The ADF Swing application relies on a bc4j.xcfg configuration file to define the server connection information. The file defines all of the deployment configurations of a particular application module in the data model project and permits ADF Swing forms to access a specific view object belonging to the application module.

You can edit the configuration file to update the connection information that the ADF Swing application uses to identify the ADF Business Components application module's deployment scenario.

Note: If you edit a configuration in the bc4j.xcfg file and change the deployment platform (Middle Tier Server Type option), you will need update your data model project to add the libraries for the new platform. Choose **Deploy to *projectname.jar*** on the Common and Middle Tier archives for the deployment archive you created in your data model project.

1.4.1 How to Modify the Configuration Name

In JDeveloper you can create and edit the configurations using the Configuration Manager by right-clicking on the application module node in the Databases Window and selecting **Configurations**. In JDeveloper, the Configurations page of the overview editor for application module lets you create a new configuration and change the default configuration.

1.4.2 How to Modify the Configuration File

JDeveloper places the `bc4j.xcfg` file in a common directory in the ADF Business Components package it generates in `/myclasses`. For example, a `bc4j.xcfg` file that you generate for an ADF Business Components package named `OnlineOrders` would appear in:

```
<jdev_install>/myclasses/OnlineOrders/common/bc4j.xcfg
```

Note: If you modify the configuration file that the application uses, you must rebuild the data model project to make the configuration available to the ADF Swing client.

You do not deploy the configuration file when you deploy the ADF Swing application. The person responsible for deploying the ADF Business Components application module will automatically deploy the `bc4j.xcfg` file as a subdirectory of the application module classes directory. You need only be sure that the deployed `bc4j.xcfg` file contains a configuration that you specify for use with your ADF Swing application and that the configuration information is correct.

1.4.3 What You May Need to Know About the Client Application Libraries

When you create or update a runtime configuration (for the `bc4j.xcfg` file), JDeveloper updates several application libraries. One of the libraries contains class files common to the ADF Swing user interface project and ADF Business Components data model project (it will have a name like `Workspace1_jws_Project1_jpr_ClassesMypackage1ModuleLocal`), which are required by JDeveloper to compile and run your ADF Swing application. JDeveloper updates this library based on the most recently saved configuration definition. Consequently, you may need to reedit a data model definition to update the library with the desired classes. Also, when you move the user interface project and the data model project to a new installation, you must move all 'named user library' definitions. All the user libraries appear in the `libraries.xml` file in the `<jdev_install>/system` folder, and it is necessary to copy this file to the new JDeveloper installation.

Creating ADF Swing Forms and Panels

This chapter describes how to create ADF Swing forms and panels using design-time wizards. The wizards help you to build UI clients in Java that display ADF Swing forms using standard JFC/Swing components bound to business service data collections.

This chapter includes the following sections:

- [Section 2.1, "About Creating ADF Swing Forms and Panels"](#)
- [Section 2.2, "Process for Creating ADF Swing Panels and Forms"](#)
- [Section 2.3, "What Happens When You Create an ADF Swing Form"](#)
- [Section 2.4, "What You May Need to Know About ADF Swing Code Generation"](#)
- [Section 2.5, "What You May Need to Know About Business Components Attribute Settings"](#)
- [Section 2.6, "How to Create a Client Data Model Definition"](#)
- [Section 2.7, "How to Create a Single Table ADF Swing Form"](#)
- [Section 2.8, "How to Create a Master-Detail ADF Swing Form"](#)
- [Section 2.9, "How to Create an Empty ADF Swing Form"](#)
- [Section 2.10, "How to Create an Empty ADF Swing Panel"](#)
- [Section 2.11, "How to Create ADF Swing Edit Forms from the Data Controls Panel"](#)
- [Section 2.12, "How to Create ADF Swing Forms from the Databases Window"](#)

2.1 About Creating ADF Swing Forms and Panels

A data browsing panel displays controls through which the user can view and edit data. Therefore, it has a set of controls declared and instantiated as fields. The data browsing panel receives its panel binding from the parent frame or panel (through a `setBindingContext()` call):

```
panel.setBindingContext(panelBinding.getBindingContext());
```

After the parent container creates the data browsing panel and its panel binding, `jbInit()` is called. In the `jbInit()` method, the control is bound to attributes.

In [Example 2-1](#), `textFieldDeptName` is a `JTextField` component that is bound to the `DepartmentName` attribute of the underlying business service, where the identifier `DepartmentName` is a reference to a definition in the `PageDef.xml` file (the file defines the binding container). The binding container keeps a list of iterator bindings. Each iterator binding specifies the view object instance and (optionally) the row set iterator.

Example 2–1 JTextField Component Bound to DepartmentName Attribute

```
textFieldDeptName.setDocument((Document)panelBinding.bindUIControl  
    ("DepartmentName", textFieldDeptName));
```

At runtime, when `setDocument()` is called, ADF Swing looks for a control binding by the specified name (`DepartmentName`). If one is found in the binding context for the form, ADF Swing uses that control binding's associated iterator binding to access the value.

2.1.1 ADF Swing Design Time Wizards

Functionally, ADF Swing is divided into design time and runtime. Because the design time is fully integrated with the JDeveloper IDE through a set of wizards and dialogs, JDeveloper helps you to generate an ADF Swing application quickly. The ADF Swing design time generates code with hooks into the ADF Swing runtime.

You can use ADF Swing wizards and dialogs without a full command of the ADF Swing runtime APIs. The design time helps you to build desktop applications that display ADF Swing forms using standard JFC/Swing components bound to business service data collections. The control bindings you add to standard Swing controls, using the ADF Swing design time, allow your ADF Swing forms to get and set values on the business components.

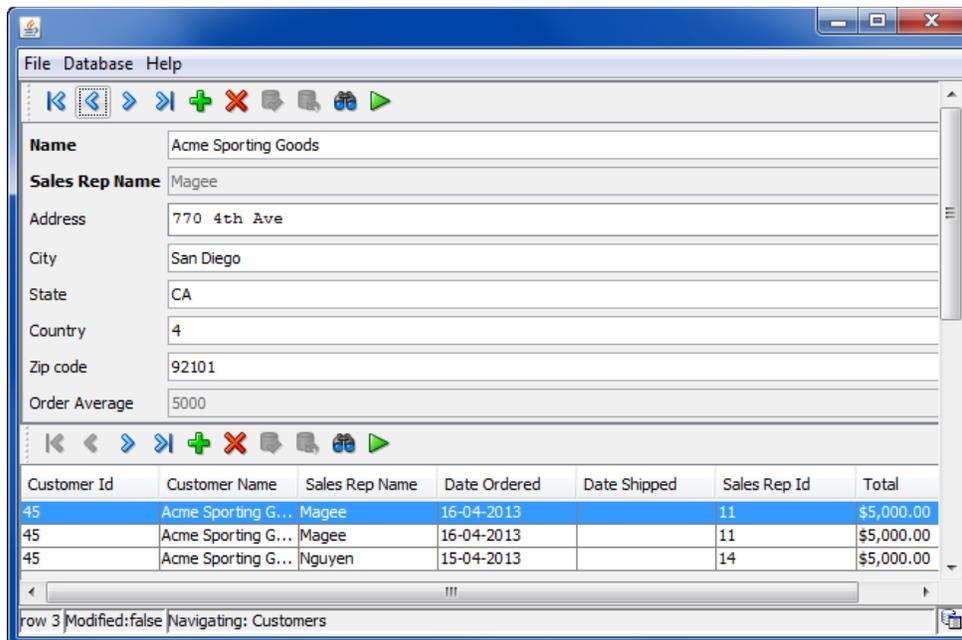
The following list of ADF Swing wizards and dialogs together with the JDeveloper IDE, help you to quickly build, run, and test an ADF Swing application or applet. You can later modify ADF Swing forms by adding more sophisticated controls and Java code to enhance your application.

- Create ADF Swing Form
- Create ADF Swing Panel
- Create ADF Swing Empty Form
- Create ADF Swing Empty Panel
- Create Java Web Start File

2.1.2 A Typical ADF Swing Form

When an ADF Swing form has been deployed to a client machine, users can use it to display and manipulate data in the form. [Figure 2–1](#) shows an example of an ADF Swing form displayed in a frame window.

Figure 2–1 Typical ADF Swing Form



At the top of the form is a menu bar. Below the menu bar is a navigation bar that controls the navigation of data in the master table. A navigation bar at the bottom of the form allows users to navigate and interact with the detail table.

In this example, the master table is the Orders table. Several databound text field controls represent columns in the Orders table and display ORDER_ID, ORDER_DATE, ORDER_SHIPPED_DATE, and ORDER_TOTAL. The form uses a databound grid control to display data from the detail OrderItems table.

When data is entered in the Order Id field, the ADF Swing form uses the master-detail association between the Orders and OrderItems tables to locate data that is displayed in the grid control. The columns from the detail table displayed in the grid control are ORDER_ID, PRODUCT_ID, QUANTITY, and UNIT_PRICE. Finally, the form contains a status bar that provides status about the data displayed on the form.

2.1.3 Navigation in an ADF Swing Form

Table 2–1 highlights the actions you can take using the navigation bar to interact with an ADF Swing form.

Table 2–1 Navigation Actions

To perform this action:	Click
Navigate through data in a form (first, previous, next, last).	
Insert data in a row below the selected row	
Delete a selected row.	
Save changes to the database.	
Undo changes made in a form.	

Table 2–1 (Cont.) Navigation Actions

To perform this action:	Click
Toggle the behavior of the panel to support Find mode or not. In Find mode, you use the panel to enter parameters to modify the query.	
Executes the query associated with the panel. When the panel is set to use Find mode, this executes a query by example.	

2.2 Process for Creating ADF Swing Panels and Forms

The process you follow to create ADF Swing panels and forms for an ADF Swing application is similar to the processes for creating user interfaces using other ADF client technologies. The main differences are that you do not create a page flow based on a controller, and there are several ADF Swing-specific editing tools, including the ADF Swing wizards.

1. Create the application workspace and select the **ADF Java Desktop Application** application template.
For more information, see [Section 1.2, "Creating a Desktop Application That Works with Oracle ADF."](#)
2. Create the data model project for your business components.
3. Add an ADF Swing panel or form to the user interface project. There are several ways to do this:
 - Add an empty ADF Swing form to the user interface project.
For more information, see [Section 2.9, "How to Create an Empty ADF Swing Form."](#)
 - Create the ADF Business Components project.
 - If you are using ADF Business Components, you can add an ADF Swing form that is already bound to view objects and attributes you choose.
For more information, see [Section 2.1.1, "ADF Swing Design Time Wizards."](#)
 - If you are using ADF Business Components, you can add to the user interface project an ADF Swing panel that is already bound to view objects and attributes.
4. Open the form or panel in the Java visual editor.
5. Use the Components window to insert Swing controls that will not be databound, for example layout components such as `JScrollPane`.
6. Use the Data Controls panel to insert databound UI components into the ADF Swing frame or panel.
7. Use the Structure window to browse the UI components and data bindings of the page.
8. Use the Properties window to modify attributes of the page's UI components and data bindings.
9. Use the ADF binding editors to modify the characteristics of bound controls.
10. When all edits are complete, build the user interface project.

11. Run or debug the application using JDeveloper.
12. After you have debugged your user interface project, you can test deployment using Integrated WebLogic Server in JDeveloper and Java Web Start application-deployment technology. Java Web Start lets users download applications and applets using a web browser but runs the application entirely on the client without the need for a web browser.

For more information, see

<http://www.oracle.com/technetwork/java/javase/overview-137531.html>.

13. Deploy the production ADF Swing application and business services to the production web server using the generated Web Application Archive (WAR) files.
14. With Java Web Start installed on the client machines, users can easily download and launch the application. Java Web Start handles updates that you make to the application on the web server each time the user launches the application.

2.3 What Happens When You Create an ADF Swing Form

When you use the Create ADF Swing Form wizard to generate an ADF Swing application with master and detail panels based on an ADF Business Components data model, the wizard generates a container panel within an ADF Swing frame. This panel is known as the layout panel because it groups several data panels together. In addition to functioning as a UI container for one or more data browsing panels, the layout panel is able to maintain the data context for the contained data panels through its shared binding context.

Note: While the layout panel is generated by the Create ADF Swing Form wizard, it is not an essential part of the ADF Swing application. It is described here primarily to demonstrate how the ADF Swing application maintains a data context between data browsing panels through a shared binding context.

The binding context from the application frame can be passed to its contained ADF Swing panels by a call to the panel's `setBindingContext()` method, as shown in [Example 2-2](#):

Example 2-2 *Passing Binding Context to ADF Swing Panel*

```
// get the binding context from the frame
BindingContext _bctx = panelBinding.getBindingContext();
// pass the context to the first child panel
dataPanel.setBindingContext(_bctx);

//alternatively you can use
dataPanel.setBindingContext(panelBinding.getBindingContext());
```

2.4 What You May Need to Know About ADF Swing Code Generation

When you run an ADF Swing wizard in the ADF Swing section of the New Gallery, the wizard helps you to generate:

- A complete, databound Swing application, consisting of multiple ADF Swing forms

- Individual ADF Swing forms that you can use to assemble your own databound Swing application
- Empty ADF Swing forms that you can use to add databound Swing components

The wizards generate ADF Swing forms that contain standard Swing components to display the data. Before you run one of the ADF Swing wizards, you can change aspects of the way code-generation works for these components. Specifically, you can select among code-generation options in the ADF Swing pages of the Preferences dialog.

You can specify that:

- The Java visual editor should be opened whenever you create a new form or data panel.
- The user interface project should be built with the specified additional import statements.
- The user interface project should be built with additional libraries, for example when you want to use your own form components. In this case, you may have added custom components and choose to assemble an ADF Swing form starting with an empty ADF Swing form.
- The Create ADF Swing Form wizard and the Create ADF Swing Panel wizard should generate forms that use standard Swing components or your own custom implementations. Currently, you can substitute components for the navigation bar, status bar, text field, and text area that appear in ADF Swing forms.
- The Create ADF Swing Form wizard should generate a single navigation bar in forms that contain more than one data panel (for example, a master-detail form). Normally, the wizard will create each data panel with its own navigation bar. Creating a form with only one navigation bar looks visually cleaner, but requires the user to change focus between the data panels to navigate the desired row set.

2.5 What You May Need to Know About Business Components Attribute Settings

You can specify whether a form allows querying or editing of specific attributes by setting these flags for individual view object attributes in the data model project:

- **Queryable**
You can prevent the displayed attribute from participating in a query that the user initiates on the form in Find mode. Open the view object overview editor and deselect **Queryable** in the Attributes Details panel for the attribute.
- **Updatable**
You can prevent the user from editing displayed attributes. Open the view object overview editor and select **Never Updatable** in the Attributes Details panel for the attribute.

2.6 How to Create a Client Data Model Definition

ADF Swing applications require a client data model definition to connect to ADF Business Components view objects. You use the Create ADF Business Components Client Data Model Definition wizard to add one or more client data model definitions to the `DataBindings.cpx` user interface configuration file.

Note: JDeveloper updates several application libraries based on the most recently saved data model definition. If you create or edit a data model definition, but want to run your project with a different data model definition, then you must open the desired data model definition in the Create ADF Business Components Client Data Model Definition wizard as described below and save it. This action generates the appropriate classes.

Before you begin:

You will need to complete these tasks:

1. Create a data model project for your business components.

In order to define a client data model, you must first create a project with an ADF Business Components application module.

2. Compile the data model project.

To create a client data model definition on a new user interface project:

1. In the Applications window, select the user interface project and from the main menu, choose **File > New > From Gallery**.

You must launch the ADF Business Components Client Data Model Definition wizard within the ADF Swing form and panel wizards.

2. In the New Gallery, select **Client Tier** and **ADF Swing** and then double-click **Form** or **Panel**.
3. In the new form or panel wizard, on the Data Model page, click **New**.

Alternatively, you can complete the ADF Swing wizard that you launched to create the form or panel. And, then you can delete the generated form or panel if you do not want to use it in your project. The new `DataBindings.cpx` configuration file will remain.

4. In the ADF Business Components Client Data Model Definition wizard, select the desired application module and runtime configuration.
5. Click **Finish** to save the changes to the new `DataBindings.cpx` configuration file.

To edit a client data model definition in an existing user interface project:

1. In the Applications window, expand the user interface project and select the `DataBindings.cpx` configuration file.
2. From the main menu, choose **Window > Structure**.
3. In the Structure window, expand **dataControlUsages** and select the data control definition that you want to modify.
4. Optionally, right-click the data control node and choose **Delete**.
5. In the Properties window, edit data control attributes.

2.7 How to Create a Single Table ADF Swing Form

Use the Create ADF Swing Form wizard to create a single table form derived from the data model of an existing ADF Business Components project.

Before you begin:

You will need to complete these tasks:

1. Create a data model project for your business components.
In order to use business components with your ADF Swing forms, you must first create a project with an ADF Business Components application module.
2. Compile the data model project.

To create a user interface project with single table ADF Swing forms:

1. In the Applications window, select the user interface project and from the main menu, choose **File > New > From Gallery**.
2. In the New Gallery, expand **Client Tier** and **ADF Swing** and then double-click **Form**.
3. In the Create ADF Swing Form wizard, on the Form Types page, the ADF Swing form type **Form** appears preselected for use in an application.
If you want to create an applet, choose type **Applet**.
4. On the Form Types page, select **Single Table** and click **Next**.
5. Make selections to define the form appearance.
6. On the Data Model page, select an existing data model definition or click **New** to create a data model definition that specifies an application module that you wish to develop against.
7. Click **Next**.
8. On the remaining pages, make selections appropriate to specify the data your form is to display.
9. Click **Finish**.

2.8 How to Create a Master-Detail ADF Swing Form

A master-detail relationship is an association between two or more view objects defined in an ADF Business Components data model. You can generate ADF Swing forms which rely on those master-detail relationships. The values in the master form determine which detail records will be displayed.

Within an ADF Business Components data model you can define the following types of master-detail relationships:

- Master form to detail form
- Master form to multiple detail forms
- Cascading master-detail relationships (master-detail-detail forms)

The easiest way to generate this type of ADF Swing form is launch the ADF Swing Form wizard and choose **Master-Detail Tables** in the wizard. When you are finish the wizard, your project will contain:

- A main frame that contains the ADF Swing bootstrap code
- A master data panel that displays the master view object data
- A detail data panel that displays the detail view object data
- A master-detail data panel that is used to parent the individual master and detail data panels

By default, both data panels will contain their own navigation bar. The navigation bar displayed in the master data panel lets users navigate the rows of the master row set while viewing the accompanying details in the detail panel. Whereas, users navigate the detail data panel to see individual rows bound to the current master.

You can force the ADF Swing Forms wizard to generate a master-detail form with a single navigation bar by setting an ADF Swing code generation preference in the Preference dialog. Creating an ADF Swing form with only one navigation bar looks visually cleaner, but requires the user to change focus between the data panels to navigate the row set.

Use the Create ADF Swing Form wizard to create master-detail forms derived from the data model of an existing ADF Business Components project.

Before you begin:

You will need to complete these tasks:

1. Create a data model project for your business components.
 - In order to use business components with your ADF Swing forms, you must first create a project with an ADF Business Components application module.
2. Compile the data model project.

To create a user interface project with single table ADF Swing forms:

1. In the Applications window, select the user interface project and from the main menu, choose **File > New > From Gallery**.
2. In the New Gallery, expand **Client Tier** and **ADF Swing** and then double-click **Form**.
3. In the Create ADF Swing Form wizard, on the Form Type page, the ADF Swing form type **Form** appears preselected for use in an application.
 - If you want to create an applet, choose type **Applet**.
4. Select **Master-Detail Table** and click **Next**.
5. Make selections to define the form appearance.
6. On the Data Model page, select an existing data model definition or click **New** to create a data model definition that specifies an application module that you wish to develop against.
7. Click **Next**.
8. On the remaining pages, make selections appropriate to specify the data your form is to display.
9. Click **Finish**.

2.9 How to Create an Empty ADF Swing Form

Use the Create ADF Swing Empty Form dialog to create a frame that contains the ADF Swing code to share a panel binding from the business components in an existing data model project.

Before you begin:

You will need to complete these tasks:

1. Create a data model project for your business components.

In order to use business components with your ADF Swing forms, you must first create a project with business services implementation, for example an ADF Business Components application module.

2. Compile the data model project.

To create a user interface project with an empty form:

1. In the Applications window, select the user interface project and from the main menu, choose **File > New > From Gallery**.
2. In the New Gallery, expand **Client Tier** and **ADF Swing** and then double-click **Empty Form**.
3. In the Create ADF Swing Empty Form dialog, define the empty form and click **OK**.
4. You can proceed to add data panels and databound controls to your new empty form.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

2.10 How to Create an Empty ADF Swing Panel

You can use the Create ADF Swing Panel wizard or the Create ADF Swing Empty Panel dialog to quickly create a data panel. While the Create ADF Swing Panel wizard generates a complete data panel that you can add to a frame in your application, the ADF Swing Empty Panel dialog contains no control bindings. Both wizards generate the code needed to initialize an ADF Swing panel binding.

After you add the new panel class to your user interface project, you can add the panel from an existing ADF Swing form or panel.

Before you begin:

You will need to complete these tasks:

1. Create a data model project for your business components.

In order to use business components with your ADF Swing forms, you must first create a project with an ADF Business Components application module.

2. Compile the data model project.

To add single table data panel using the Create ADF Swing Panel wizard:

1. In the Applications window, select the user interface project and from the main menu, choose **File > New > From Gallery**.
2. In the New Gallery, expand **Client Tier** and **ADF Swing** and then double-click **Panel**.
3. In the Create ADF Swing Panel wizard, select a template to lay out the panel components.
4. On the Data Model page, select an existing data model definition or click **New** to create a data model definition that specifies an application module that you wish to develop against.
5. On the remaining pages, make selections appropriate to specify the data your form is to display.
6. Click **Finish**.

To add an ADF Swing data panel using the ADF Swing Empty Panel dialog:

1. In the Applications window, select the user interface project and from the main menu, choose **File > New > From Gallery**.
2. In the New Gallery, expand **Client Tier** and **ADF Swing** and then double-click **Empty Panel**.
3. In the Create ADF Swing Empty Panel dialog, enter a class name for the new empty data panel and click **OK**.
4. You can open the new panel class in the Java visual editor and add controls.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

To link the completed ADF Swing data panel with your application:

After you have created a new ADF Swing data panel, you can reuse the data panel in your application by:

- Adding it to an existing layout panel (for instance, the one created by your main ADF Swing frame).
For more information, see [Section 3.6, "How to Drop Data Panels Onto an Empty ADF Swing Form."](#)
- Creating an ADF Swing frame using the ADF Swing Empty Form wizard and add it there.

For more information, see [Section 3.6, "How to Drop Data Panels Onto an Empty ADF Swing Form."](#)

2.11 How to Create ADF Swing Edit Forms from the Data Controls Panel

You can use the Data Controls panel to create a databound form that permits editing of the displayed values using controls that you select. The form will be created for the objects of a data collection that you select.

Note: The Data Controls panel may appear empty when you first open it. Compile the data model project to populate the panel with data objects.

Before you begin:

You will need to complete these tasks:

1. Create a data model project for your business components.
In order to use business components with your ADF Swing forms, you must first create a project with an ADF Business Components application module.
2. Compile the data model project.
When your data model project uses ADF Business Components as its business service, JDeveloper registers the business service as an ADF data control for you.

To create the editable form in an existing panel in the Java visual editor:

1. Create an empty ADF Swing form or panel using the wizards.

For more information, see [Section 2.2, "Process for Creating ADF Swing Panels and Forms."](#)

2. Open the form in the Java visual editor and click the **Design** tab.
3. In the Applications window, click the expand icon in the Data Controls header.
4. In the Data Controls panel, drag the desired data collection into the open form or panel.
5. Select **Add Edit Form** from the popup list.
6. In the Create ADF Swing Edit Form dialog, select the attribute you don't want to display and click the **Delete** button.
7. Optionally, select an attribute and click the up or down arrow to change the attribute's display position in the form.
8. Optionally, select the **Control** dropdown and choose a control to display the attribute value.
9. Optionally, deselect **Create Label** for an attribute if you do not want to display a label.

By default, the attribute ID is used for the display label. Alternatively, if a control hint label exists for the business object attribute, the label will use the control hint instead of the attribute ID. Leave Create Label selected.

10. Click **OK** to save the settings.

JDeveloper creates the edit form as a new panel inside the open form or panel.

11. In the Java visual editor, resize the new edit form panel to view the controls.

The edit form uses the JGoodies `FormLayout` manager for flexible component layout.

The source for the edit form panel appears in the file

Panel<collectionname>Helper.java and the panel's control binding definitions appear in **Panel<collectionname>HelperPageDef.java**.

You can improve the performance of your ADF Swing application by defining the `fetchAttributeProperties()` method in your form. This will ensure your form performs in batch mode to fetch attribute values. For more information, see [Section 10.4, "How to Limit Fetching of ADF Business Components Attributes in ADF Swing."](#)

2.12 How to Create ADF Swing Forms from the Databases Window

Use the Databases Window to create a databound form that permits editing of the displayed values using controls that you select. The form and the necessary data bindings will be created for the database table that you select.

To create the form in an existing panel in the Java visual editor:

1. Create an empty ADF Swing form or panel using the wizards.
For more information, see [Section 2.2, "Process for Creating ADF Swing Panels and Forms."](#)
2. Open the form in the Java visual editor and click the **Design** tab.
3. In the Properties window, select **BorderLayout** from the **layout** dropdown list.
For more information, see to [Section 3.7, "How to Lay Out Data Panels in an Empty Swing Form."](#)

4. In the Components window, open the **Swing Containers** page and select the **JScrollPane** component.
5. Click inside the empty form in the Java visual editor to drop the scroll pane with its default size.
6. Resize the scroll pane.
7. From the **Window** menu, choose **Database > Databases**.
8. In the Databases window, drag the desired database table onto the open form or panel on top of the scroll pane.
9. When you want to create an editable form, select **Add Edit Forms** from the popup list.

For more information, see [Section 2.11, "How to Create ADF Swing Edit Forms from the Data Controls Panel."](#)

10. Alternatively, add a specific component to the empty form by choosing **Add Child**.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

After you lay out the data panel or form, you can improve the performance of your ADF Swing application by defining the `fetchAttributeProperties()` method in your form. This will ensure your form performs in batch mode to fetch attribute values. For more information, see [Section 10.4, "How to Limit Fetching of ADF Business Components Attributes in ADF Swing."](#)

Modifying ADF Swing Forms and Panels

This chapter describes how to customize an ADF Swing application using the Java visual editor. You use the Data Controls panel to insert databound UI components into an ADF Swing-prepared form or panel.

This chapter includes the following sections:

- [Section 3.1, "About Modifying ADF Swing Forms and Panels"](#)
- [Section 3.2, "How to Assemble ADF Swing Forms Using the Java Visual Editor"](#)
- [Section 3.3, "How to Insert UI Components into ADF Swing Panels"](#)
- [Section 3.4, "How to Change Client Data Model References"](#)
- [Section 3.5, "How to Open an ADF Swing Form with an Action Handler"](#)
- [Section 3.6, "How to Drop Data Panels Onto an Empty ADF Swing Form"](#)
- [Section 3.7, "How to Lay Out Data Panels in an Empty Swing Form"](#)
- [Section 3.8, "Binding a Method with Parameters in an ADF Swing Form"](#)

3.1 About Modifying ADF Swing Forms and Panels

After you generate ADF Swing forms and panels using the ADF Swing wizards, you may want to customize the generated application files. JDeveloper helps you customize the application using the visual tools. For example, you can use the Data Controls panel to insert already databound UI components into an ADF Swing-prepared form or panel.

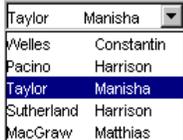
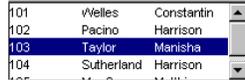
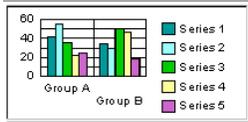
3.1.1 Value Bindings for the Entire Collection or Data Object

The Data Controls panel provides UI components that you can use to bind an entire data collection (which consists of data objects that comprise a row set), as shown in the [Table 3-1](#).

Table 3-1 UI Components That Can Be Bound to an Entire Data Collection

UI Component	Drag and Drop As	ADF Binding Type
101 Welles Constantin	Table	Table binding
102 Pacino Harrison		
103 Taylor Marisha		
104 Sutherland Harrison		
105 MacGraw Matthias		

Table 3–1 (Cont.) UI Components That Can Be Bound to an Entire Data Collection

UI Component	Drag and Drop As	ADF Binding Type
	Combo Box	List binding in Navigation mode
	List (inside a ScrollPane)	List binding in Navigation mode
	Spinner	List binding in Navigation mode
	Radio Button Group	List binding in Navigation mode
	NavigationBar	Iterator binding
	Tree	Tree binding
	Graph	Graph binding
	Slider	Scroll binding
	ScrollBar	Scroll binding

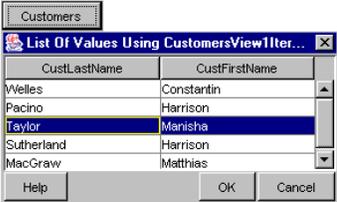
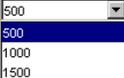
3.1.2 Value Bindings for Individual Data Object Attribute Values

The Data Controls panel provides UI components that you can use to bind a single data object attribute, as shown in [Table 3–2](#).

Table 3–2 UI Components That Can Be Bound to a Single Data Object Attribute

UI Component	Drag and Drop As	ADF Binding Type
	TextField	Attribute binding
	Edit Pane	Attribute binding
	JULabel	Attribute binding

Table 3–2 (Cont.) UI Components That Can Be Bound to a Single Data Object Attribute

UI Component	Drag and Drop As	ADF Binding Type
	Label For (for ADF Business Components to display attribute's label control hint)	Attribute binding
	Password Field	Attribute binding
	Text Area	Attribute binding
	Text Pane	Attribute binding
	Button LOV	LOV binding
	Check Box	Boolean binding
	Formatted Edit Field	Formatted Text binding
	Combo Box	List binding in Enumeration mode
	List	List binding in Enumeration mode
	Spinner	List binding in Enumeration mode
	Radio Button Group	List binding in Enumeration mode
	Progress Bar	Bounded Range binding
	Scroll Bar	Bounded Range binding
	Slider	Bounded Range binding

3.2 How to Assemble ADF Swing Forms Using the Java Visual Editor

The Create ADF Swing Empty Form dialog lets you create an empty form that you can use to assemble an ADF Swing form without the need to write additional Java code. The `main()` defined in the ADF Swing empty form contains ADF Swing code, known as bootstrap code, that:

- Establishes a connection to a business service instance, such as an ADF Business Components application module, that provides the data model for the form.
- Creates an instance of a panel binding from the data model to provide data access to the databound Swing components.

The bootstrap code generated by the wizard permits the ADF Swing empty form to share its panel binding with ADF Swing data panels that you add. You can use the Java visual editor to add the ADF Swing data panels to assemble the final databound ADF Swing form.

Note: The Create ADF Swing Form wizard helps you generate databound ADF Swing forms to browse and edit ADF Business Components view objects that you select during the process of using the wizard. If you need to create an ADF Swing form of your own design, start with an empty form that is initially databound.

To create a databound ADF Swing form entirely within the Java visual editor:

1. Create an empty form using the Create ADF Swing Empty Form dialog. This adds an ADF Swing frame to your user interface project that can share a panel binding.

For more information, see [Section 2.9, "How to Create an Empty ADF Swing Form."](#)

2. Drop an ADF Swing data panel onto the ADF Swing empty form.

For more information, see [Section 3.6, "How to Drop Data Panels Onto an Empty ADF Swing Form."](#)

3.3 How to Insert UI Components into ADF Swing Panels

Use the Data Controls panel to insert databound controls into an ADF Swing panel.

Note: The Data Controls panel may appear empty when you first open it. Compile the data model project to populate the panel with data objects.

Before you begin:

You will need to complete these tasks:

1. Create a data model project for your business components.

In order to use business components with your ADF Swing forms, you must first create a project with an ADF Business Components application module.

2. Compile the data model project.

When your data model project uses ADF Business Components as its business service, JDeveloper registers the business service as an ADF data control for you.

To insert a databound UI component into the panel in the Java visual editor:

1. Create an empty ADF Swing form or panel using the wizards.

For more information, see [Section 2.1.1, "ADF Swing Design Time Wizards."](#)

2. Open the form in the Java visual editor and click the **Design** tab.

3. In the Applications window, click the expand icon in the Data Controls header.

4. In the Data Controls panel, drag the data collection, attribute, or action that you want to bind to a UI component into the open document.
5. From the **Add Child** popup list, select the UI component that you want to add to the open document.

The new UI component appears in the document you are editing.

After you lay out the data panel or form, you may improve the performance of your ADF Swing application by defining the `fetchAttributeProperties()` method in your form. This ensures your form performs in batch mode to fetch attribute values.

3.4 How to Change Client Data Model References

You do not need to edit your application code to change the data model definition it will use to connect to your business services data source. The definition is contained entirely in the metadata for the user interface project in two files: `DataBindings.cpx` and `PageDef.xml`.

You might want to do this because you had been using a local configuration to test your application in JDeveloper and you now want to change to a data model definition that uses a remote deployment configuration. You could also decide to use an entirely different data model defined in a different business service project. Again, no code changes are required to accomplish this task.

To reference the new client data model definition in the ADF Swing metadata:

1. Add a new client data model definition to the `DataBindings.cpx` file in your user interface project and remember the name you chose (for example, `remotedatamodel`).

For more information, see [Section 2.6, "How to Create a Client Data Model Definition."](#)

Note: If you change your data model to use an ADF Business Components application module from a package in a different project and the new application module is defined as Session Bean (BMT), then you must modify the `<ejb-ref>` entry in the `web.xml` file, as well as update the `.cpx` file.

2. Optionally, you can open the `DataBindings.cpx` file in the XML editor and edit the attributes of the `BC4JDataControl` definition:
 - Choose **Window > Structure** to display the Structure window for the file.
 - In the Structure window, select the data control node you want to modify.
 - Choose **Window > Properties** to display the data control definition and edit its attributes.
3. Open the `PageDef.xml` file in the XML editor, click the **Overview** tab and select any binding that references the old data control.
4. In the Properties window, expand the **Common** section and select the **Data Source** for the desired collection from the dropdown list. You can then select the desired **Attribute** for the selected data source from the dropdown list.

Repeat for each binding in the binding definition.

3.5 How to Open an ADF Swing Form with an Action Handler

You can use the ADF Swing wizards to create ADF Swing forms with various databound controls. Later, when you want your ADF Swing forms to run from a single main window, you can create an ADF Swing frame that contains:

- The bootstrap code to create the business service client data model connection
- An action event handler to open the ADF Swing form and pass it a panel binding

When the user performs an action in the UI, such as clicking a button, an event is issued. Events are objects that describe what happened and are only reported to registered listeners. JDeveloper generates all of this code for you. The following procedure describes the code you must supply to open an ADF Swing form when the button is clicked.

To define an action to open an ADF Swing form:

1. Create an empty ADF Swing frame that creates a connection to the business service for the form.

For more information, see [Section 2.9, "How to Create an Empty ADF Swing Form."](#)

2. Open the empty ADF Swing frame in the Java visual editor and delete the Navigation Bar and Status Bar generated by the Create ADF Swing Empty Form dialog.

3. Add a JButton control from the Components window to the data panel of the empty form.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

4. Select the button in the Java visual editor.
5. In the Properties window, expand the **Events** section and in the **actionPerformed** field enter the name of the function you want executed whenever the button is clicked and press the **Enter** key.

JDeveloper takes you to a stub of your function in the source editor.

6. Add code to the event stub to create the ADF Swing form you want to display and set its visible property to true:

```
FrameMyNewView frame = new
FrameMyNewView(getPanelBinding()); frame.setVisible(true);
```

The `getPanelBinding()` method allows you to share the panel binding from main ADF Swing frame. This results in the iterator bindings to be shared between ADF Swing forms. The new frame will automatically be synchronized with the navigation bar and status bar in the first detail of the master-detail frames.

Or

Add code to create the ADF Swing form and set a new panel binding when you don't want the form to be synchronized with the frame that opened it:

```
FrameMyNewView(new
JUPanelBinding(getPanelBinding().getApplicationName(), null)); frame.setVisible(true);
```

3.6 How to Drop Data Panels Onto an Empty ADF Swing Form

You can assemble an ADF Swing form using existing data panels from your current project or you can insert a new empty data panel which you can lay out with specific controls.

To add a data panel to an ADF Swing Empty Form:

1. Open the empty form in Java visual editor and click the **Design** tab.
2. Choose **Window > Components** to display the list of Swing controls.
3. In the Components window, select **ADF Swing Regions** to view the existing data panels in your project.
4. Drag the ADF Swing panel you want to reuse onto the empty panel.
5. In the Select Controls dialog, select how you want the form to handle data panel.
6. Add controls to the data panel.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

The ADF Swing panel that you add to an ADF Swing form receives its databinding from the ADF binding container. When `setBindingContext()` is called on the form, the binding container for the form is created together with the panel's binding container.

3.7 How to Lay Out Data Panels in an Empty Swing Form

You can lay out any Swing panel using the `FormLayout` layout manager provided with the JGoodies Form framework. The `FormLayout` gives you excellent grid-based control over the placement and alignment of controls. Unlike other layouts, you can fill components across grid cells.

To set `FormLayout` on an ADF data panel:

1. Open the form in the Java visual editor and click the **Design** tab.
2. Choose **Window > Properties**.
3. In the Property window, expand the **Visual** section and select **FormLayout** from the **layout** dropdown list.

For more information, see "Adding Components" in *Developing Applications with Oracle JDeveloper*.

4. Customize the panel layout.

For more information, see "Working with Layout Managers" in *Developing Applications with Oracle JDeveloper*.

3.8 Binding a Method with Parameters in an ADF Swing Form

You can use the Data Controls panel to insert a button that will allow the user to initiate an action defined by a method of your business service. If your business service is ADF Business Components, many standard actions are predefined (such as Create, Delete, Next, Previous, Commit, and Rollback).

3.8.1 How to Populate the Data Controls Panel with JavaBean Methods

When your business service is a JavaBean class, you must define a public method and register the bean as an ADF data control. The method that you create may define arguments whose values can be supplied by the ADF Swing form user.

Alternatively, you can supply the parameter values of the method in the Properties window.

To populate the Data Controls panel with your JavaBean methods:

1. Define the JavaBean that you want your Oracle ADF application to access.
The business services appear in your data model project
2. Register the business services in your data model project with the ADF data controls.

When your data model project uses ADF Business Components as its business service, JDeveloper registers the data control for you.

3.8.2 How to Create an ADF Swing Form with Method Bindings

When you can create an ADF Swing form, you have the option of displaying a method binding as a Button or Method panel. The Method panel displays a component for each parameter and a component that display each method result.

To create an ADF Swing Form with method bindings:

1. Open the data panel in the Java visual editor.
2. In the Data Controls panel, expand the **Operations** folder for the data collection or data control that displays the desired custom method.

The Data Controls panel hierarchy represents operations (such as Create and Delete) that apply to a specific data collection in the Operations folder below the data collection. When supported by the ADF data control for your business service, you can also select operations (such as Commit and Rollback) that apply to all data collections in the current document's binding context in the Operations folder at the top branch of the hierarchy.

3. Drag the desired operation into the open document.

A method with no parameters or result is automatically added as a `JButton` component to initiate the action. If the method has parameters or a return value, JDeveloper displays a popup that lets you select how you want to add the method. Select one of these options depending on how you want the form to display the method binding in your application.

- Select **Button** when you want to add a button that will initiate the action. You are responsible for passing parameters to the method.
- Select **Method** when you want to add a method panel that will display a component for each parameter and a component to display a method result, as needed. In the Select Controls for Parameters and Results dialog that displays, select the components the panel will display.

JDeveloper adds code to the class file to bind the `JButton` or `JPanel` to the operation.

4. If you drop the method as a Button, and you want to supply parameter values yourself, you can add a text field to the form and add a `focusListener` to the text field and modify its `focusLost()` method in the data panel's `.java` file, using the

code in [Example 3-1](#) to pass the parameter to the method:

Example 3-1 Code for Passing a Parameter to Method

```
private void jTextField1_focusLost(FocusEvent e) {
    JUCtrlActionBinding action =
        (JUCtrlActionBinding)panelBinding.getCtrlBinding(jButton1);
    ArrayList arg1 = new ArrayList();
    arg1.add(jTextField1.getText());
    action.setParams(arg1);
}
```

Repeat this step to create an event handler for each text field that supplies a method parameter.

3.8.3 What You May Need to Know About Displaying a Method Result Using a JTable Component

When you drop a custom method with a return value from the Data Controls panel, JDeveloper prompts you for the component to bind to the method result. Note that if you choose to display a result that is a collection in a `JTable`, your application may fail at runtime with a `NullPointerException` when the Swing `JTable` component attempts to prepare the renderer. This exception is due to a Swing `JTable` limitation that prevents it from setting up renderers for the `Integer` type (Swing supports `Numbers`, `Doubles`, `Floats`, `Dates`, and `Booleans`). You will need to subclass the `JTable` and set a default renderer for types that Swing does not natively support.

For example, to install a custom renderer for `Integer`, you can use the code shown in [Example 3-2](#).

Example 3-2 Installing a Custom Renderer for Integer

```
private JTable jTable1 = new JTable() {
    protected void createDefaultRenderers() {
        super.createDefaultRenderers();
        setDefaultRenderer(Integer.TYPE,
            super.getDefaultRenderer(java.lang.Number.class));
    }
}
```

Working with Data Binding

This chapter describes how to create Swing containers and components that are bound to data objects from ADF Business Components. It describes the easiest way to create databound containers using the ADF Swing wizards.

This chapter contains the following sections:

- [Section 4.1, "About Working With Data Binding"](#)
- [Section 4.2, "Navigating the UI Using ADF Swing Controls"](#)
- [Section 4.3, "What You May Need to Know About the ADF Swing Data Context"](#)
- [Section 4.4, "What Happens at Runtime: How Panel Bindings Function"](#)
- [Section 4.5, "What You May Need to Know About the ADF Swing Bootstrap Code"](#)
- [Section 4.6, "How to Display Object Attributes in a Databound Text Field"](#)
- [Section 4.7, "How to Create a New Row in a Databound Table or Tree Control"](#)
- [Section 4.8, "How to Sort Columns in a Databound Table"](#)
- [Section 4.9, "What Happens At Runtime: How Control Bindings Function"](#)

4.1 About Working With Data Binding

Data binding in ADF Swing is the ability to create Swing containers and components that are bound to data in back-end business services. To enable data binding, ADF Swing provides a small API that works with the Oracle ADF model layer. The API is exposed in the application source code through a combination of ADF Swing bootstrap code:

- Call `loadCpx()` -- load the application metadata (specified in the `DataBindings.cpx` file), which specifies a connection to the business service implementation instance (for example, an ADF Business Components application module instance) using the ADF data control for the instance, as well as the ADF binding context.
- Call `setBindingContext()` -- make the ADF binding context available to the frame or panel.
- Call `createPanelBinding()` -- to create an object that will access the business service's contained data collections through Swing component models.
- Call `bindUIControl()` on the panel binding to set the ADF model for the individual components of the ADF Swing form or panel.

To work with a data binding in your Swing application, each container (a frame or panel) must either create a panel binding object or get one from the source from which

it originated. The frame that creates the first panel binding also contains the ADF Swing bootstrap code, where the connection to the business services is created. Subsequent containers that your application creates either chain from the original panel binding or they create their panel binding in order to display unrelated data. How you want to partition the data views of your application determines whether a container sets a new panel binding or whether it gets an existing one:

- If you want to create independent branches of the business services views, then your application should open a frame that sets a new panel binding.
- If you want to maintain the same view along a continuous branch of your application (say a master and detail branch for example), then secondary containers all share the panel binding object created by the initial frame.

4.1.1 ADF Swing Containers

The easiest way to create databound containers is to use the ADF Swing wizards (see the ADF Swing folder in the New Gallery). Specifically, if you use these two ADF Swing wizards, then the source code will contain the bootstrap code and constructors needed to create the panel binding:

- Use the Create ADF Swing Empty Form dialog to generate an empty frame that creates an ADF Swing panel binding with a connection to the business service used by your application, for example ADF Business Components.
- Use the Create ADF Swing Empty Panel dialog to generate an empty panel with constructors to create a new panel binding or to share one from its parent frame.

An additional benefit to using these two wizards is their support for easy drag-and-drop UI design within JDeveloper. Because they are generated with the bootstrap code for a specific data control object (which contains the business service's collections, structured objects, attributes, and methods), all of the Swing components that you insert from the Data Controls panel in JDeveloper will have access to any business service that the data control object contains.

4.1.2 Standard Java Containers

If you start with a standard frame or panel (one generated without using the ADF Swing wizards) that you want to enable an ADF Swing data binding for, you can add the appropriate ADF Swing bootstrap code to the main frame and then handle the panel binding in your secondary windows this way:

- If you want to share the panel binding with the parent frame:


```
BusinessCompViewName(getPanelBinding()); frame.setVisible(true);
```
- If you want the new frame to define its own panel binding:


```
BusinessCompViewName(new
JUPanelBinding(getPanelBinding().getApplicationName(), null)); frame.setVisible(true);
```

The first creates the frame object and set the panel binding. The second call makes the frame visible.

4.2 Navigating the UI Using ADF Swing Controls

When you create a default master-detail form using ADF Swing, it will create and place a navigation bar on both the master and the detail panel, which permits users to

scroll through the data in each panel independently. Or, you can create a single navigation bar which responds to the panel which has current focus.

4.2.1 How to Navigate Using the Navigation Bar

In the ADF Swing form, you need to move the code for the navigation bar from the individual panels to the layout panel where the navigation event will affect all the child panels. For example, you can move the code from the department and employees data panel to the layout panel.

The code that needs to be moved will be similar to [Example 4-1](#).

Example 4-1 Navigation Bar Code

```
// The declaration of the navigation bar
private JUNavigationBar navBar = new JUNavigationBar();

// The code that binds the navigation bar to the individual panel.
navBar.setModel(JUNavigationBar.getModelInstance(getPanelBinding(),
"DepartmentsView", null, "DepartmentsViewIter"));

//Add the navigation bar to the panel
add(navBar, BorderLayout.NORTH);
```

Once you have moved the navigation bar code you need to add the control binding to the layout panel which contains both the master and the detail panels. [Example 4-2](#) shows the code that you will add to the layout panel to bind the model for the navigation bar.

Example 4-2 Binding Model for Navigation Bar to Panel

```
//The declaration of the navigation bar
private JUNavigationBar navBar = new JUNavigationBar();

//Bind the model for the navigation bar to the panel
navBar.setModel(JUNavigationBar.getModelInstance(getPanelBinding(),navBar));

//Add the navigation bar to the panel
add(navBar, BorderLayout.SOUTH);
add(masterScroller, BorderLayout.NORTH);
add(detailViewPanel, BorderLayout.CENTER);
```

4.2.2 How to Navigate Using Tree Navigation

When you add a tree control to your panel you create node-populating rules using the property editor for the ADF Swing node model. The property editor does not allow you to handle node selection. If you want to handle the node-selection event in order to populate controls in a panel, you can use `JUTreeDefaultMouseListener` to synchronize master and detail panels on the selected node. [Example 4-3](#) shows how to add the listener to the tree control.

Example 4-3 Adding Listener to Tree Control

```
myTreeControl.addMouseListener(new JUTreeDefaultMouseListener
( panelBindingVar, new String [][] {
  { "NodeType1" , "DepartmentViewIter" }, "NodeType2" , "EmployeeViewIter" } }
)
);
```

4.3 What You May Need to Know About the ADF Swing Data Context

The ADF `SwingPanel` interface implemented by ADF `SwingFrame` or `JPanel` permits your ADF Swing application to:

- Maintain a consistent data context between the databound panels (also known as chaining between data panels)
- Access data through databound Swing controls

During design time, each data browsing panel that you add to the ADF Swing application gets its context for marshaling interactions between the UI controls and the business service's row set iterator from the panel binding object created in the frame or containing panel (such as the master-detail layout panel). The capability in ADF Swing to chain data browsing panels is provided without the need to write additional code. For example, the data browsing panels generated by the wizard, `PanelDeptView` and `PanelEmpView1`, share the same data context through an instance of a panel binding (`JUPanelBinding`) when each `JPanel` implements the `setPanelBinding()` and `getPanelBinding()` methods of the ADF `SwingPanel` interface.

Once you have a frame or panel that creates this panel binding, ADF Swing permits you to assemble the application by adding new data browsing panels that either share the existing panel binding object or create a new one.

Then you can use the Data Controls panel in JDeveloper to add databound controls one by one to the data panel. At the level of the Swing component, this sets the data binding by specifying an ADF Swing control model on the control's `document` or `model` property. At runtime, each control in the data panel becomes databound through the panel binding object as an argument to the control's `setModel()` or `setDocument()` method.

4.4 What Happens at Runtime: How Panel Bindings Function

To understand how the panel binding is created and used by the databound panels, consider what happens when you run the application, starting with the ADF Swing frame, and the following ADF Swing code is executed:

1. The `main()` method bootstraps the application. It starts a binding context and loads the ADF data control, based on entries in the `DataBindings.cpx` file. Then it passes the binding context with initialized ADF model objects to the panel binding to create the ADF data bindings.

For more information, see [Section 4.5, "What You May Need to Know About the ADF Swing Bootstrap Code."](#)

2. The frame is initialized (`FrameDeptViewEmpView1`, in the example above) through a constructor that takes an application object. Initialization of the frame results in a panel binding object (`JUPanelBinding`), based on an ADF model definition that may have components that are bound to data from more than one data control. The creation of the panel binding is an important part of the ADF Swing functionality, which enables data binding for Swing components and chaining of data panels.

3. The frame or applet class initializes a layout panel (`MDPanelDeptViewEmpView1`, in the example above) and sets the panel binding on the new layout panel, using the `setBindingContext()` method.

For more information, see [Section 2.3, "What Happens When You Create an ADF Swing Form."](#)

4. In the layout panel's `jbInit()` method, the data browsing (children) panels are created. For this, ADF Swing uses the shared binding context for binding the child data panels (`PanelDeptView` and `PanelEmpView1`, in the example above).
5. A control-to-attribute data binding occurs using the control's specified ADF Swing model. (This binding information is stored in the binding container XML metadata.)
6. The control binding handles events to populate and update data for the UI control
For more information, see [Section 4.9, "What Happens At Runtime: How Control Bindings Function."](#)

4.5 What You May Need to Know About the ADF Swing Bootstrap Code

When you select the frame class in the navigator and choose **Run**, the `main()` method "bootstraps" the application. It starts a binding context and loads data controls, based on entries in the `DataBindings.cpx` file. Then it passes the binding context with initialized data controls to the panel binding to create the ADF data bindings.

[Example 4-4](#) shows the bootstrap code created by the Create ADF Swing Form wizard, using selected columns from the Employees and Departments tables from the HR schema.

Example 4-4 Bootstrap Code Created by Create Form Wizard

```
// bootstrap application
JUMetaObjectManager.setBaseErrorHandler(new JUErrorHandlerDlg());

// Lookup the *.cpx file and create all data controls listed in this file.
JUMetaObjectManager mgr = JUMetaObjectManager.getJUMom();

// Use the definition classes provided by ADF Swing. Change only if you do not
// want to use custom DefClasses.
mgr.setJClientDefFactory(null);

// Create a new binding context that extends java.util.Hashtable.
BindingContext ctx = new BindingContext();

// Get user connection information if available. If not, display logon dialog.
ctx.put(DataControlFactory.APP_PARAM_ENV_INFO, new JUEnvInfoProvider());

// Set locale to the default locale of the JVM.
ctx.setLocaleContext(new DefLocaleContext(null));

// Load data binding container data binding file.
HashMap map = new HashMap(4); map.put(DataControlFactory.APP_PARAMS_BINDING_
CONTEXT, ctx);
mgr.loadCpx("mypackage.DataBindings.cpx", map);

// Get handle to the ADF Business Components application module. The code lines
// below are added only when using the ADF Swing Form wizard. Declaratively
// creating
// the frame, starting with an empty form wizard does not add the following lines.
DCDataControl app = (DCDataControl)ctx.get("model_AppModuleDataControl");
app.setClientApp(DCDataControl.JCLIENT);

// Despite the following line of code, attribute sets and fetches are normally
// performed in one batch operation. This requires only one network round
// trip. Attributes that aren't needed are not loaded to the client. The code
```

```
// line below is added only when using the ADF Swing Form wizard. Declaratively
creating
// the frame, starting with an empty form wizard does not add the following lines.
app.getApplicationModule().fetchAttributeProperties(new String[]
{"DepartmentsView1", "EmployeesView3"}, new String[][] {{"DepartmentId",
"DepartmentName" }, {"EmployeeId", "FirstName", "LastName" "DepartmentId" }},
null);

// Initialize application root class.
FormDepartmentsView1EmployeesView3 frame = new
FormDepartmentsView1EmployeesView3();

// Set binding context to the frame.
frame.setBindingContext(ctx);
frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
Dimension frameSize = frame.getSize();
```

The frame is initialized by its constructor, which does not expect any arguments by default. The binding context of the application is passed to the `setBindingContext()` method of the frame.

Initialization of the frame results in a panel binding object (`JUPanelBinding`) based on an Oracle ADF model definition that may have components that are bound to data from more than one data control. The creation of the panel binding is an important part of the ADF Swing functionality, which enables data binding for Swing components and chaining of data panels.

After you lay out the data panel or form, you can improve the performance of your ADF Swing application by defining the `fetchAttributeProperties()` method in your form. This will ensure that your form performs in batch mode to fetch attribute values. For more information, see [Section 10.4, "How to Limit Fetching of ADF Business Components Attributes in ADF Swing."](#)

4.6 How to Display Object Attributes in a Databound Text Field

When using databound text fields to display the attribute values of an object, such as those defined by an Address object, the fields will not display the attribute values (they will display instead some static text).

If you bind a text field to an object attribute, you can ensure the value is correctly displayed in the ADF Swing panel by forcing the query to reexecute on the iterator binding of the bound object.

To force `executeQuery()` on the object's iterator binding, add this method call after `jbInit()` is done in the panel, where `Street` is replaced by the name of your object attribute binding:

```
panelBinding.findControlBinding("Street").getIteratorBinding().
executeQuery();
```

It is only necessary to call `executeQuery()` on one of the object domain attributes (like `Street`) to force the iterator binding to refetch all attribute values of the same object.

4.7 How to Create a New Row in a Databound Table or Tree Control

If your business services supports create operation on the data collection, you can use the operation in an ADF Swing panel to display a new row in your databound table or

tree control. The new row will appear when the user clicks a **Create** button that has been bound to a create-and-insert action binding. Because the operation creates and inserts the row in a single step, this operation is ideal for in-place editing of the component by the user.

Note: Although the Data Controls panel displays this operation as Create, the action binding editor will be set to `CreateInsert`. This behavior differs in the case of web applications.

To create an ADF Swing bound control that uses a create operation to insert a row:

1. Open the data panel in the Java visual editor.

For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)

2. In the Data Controls panel, select the collection node that contains the attributes you want your UI component to display. This is the collection to which the control will add the row. In the dropdown list, select the desired UI component to display the new row.

Note: The data collection you select must not be a detail collection, represented in the data control hierarchy as a child of another collection node. This will ensure that the iterator binding does not perform navigation.

3. Drag the collection into the open ADF Swing panel.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

Tip: Optionally, your panel can display navigation buttons to the allow the user to browse the collection. For more information, see [Section 7.8, "How to Use the JINavigationBar Control."](#)

4. To insert the operation to create the row, select **Create** in the operations folder for the previously selected collection and select **Button** from the dropdown list.
5. Insert the button into the open ADF Swing panel.

Note: When you drag Create from the Data Controls panel, the button's action binding is set to `CreateInsert`. This behavior differs in the case of web applications.

4.8 How to Sort Columns in a Databound Table

When you use the ADF iterator binding to create a databound Swing table, you can specify the sort criteria (ascending or descending) in which the data of the table columns display. You can use one or more columns as the sort criteria, and specify the sort priority of each of those columns. For example, to sort an employee table you could choose last name as the first sort criteria and first name as the second sort criteria. In this example, the second sort criteria becomes useful when two or more

employees have the same last name thus requiring sorting, in the specified order, by first names.

When sorting the columns in a databound table, you use the iterator binding for the table.

To sort the columns in a table bound to an ADF iterator binding:

1. In the Java visual editor, right-click the form or panel that contains the table you want to sort and choose **Go to Page Definition**.
2. In the overview for the page definition editor, double-click the iterator binding in the **Executables** list.
3. In the Edit Iterator Binding dialog, click the **Sort Criteria** tab.
4. Select an attribute and choose whether the sort should be performed in ascending or descending order.

If the attribute is not a sort criteria for the table, leave the **No Sort** selection assigned.

5. Use the up and down arrow buttons to change the sort priority of an attribute.

Moving the sort criteria attribute higher in the list, yields a higher sort priority, which means the attribute will be sorted before the sort criteria attributes that appear lower in the list.

4.9 What Happens At Runtime: How Control Bindings Function

After data browsing panels are initialized, the layout panel calls `executeIfNeeded()` on the panel binding to execute the query on the ADF Business Components data source.

4.9.1 Populating Controls with Data

The `executeIfNeeded()` method determines whether the query has executed on the view object. If not, the method calls `executeQuery()` on it. This executed query brings data from the database into the cache and causes the ADF Business Components row set listener events to fire. The first among these is the `RowSetListener.rangeRefreshed` event. This event is captured by the iterator binding (because it implements `RowSetListener` and has registered itself as a listener). It retrieves the rows of the range and calls `updateValuesFromRows()` on the control binding. The control binding takes the data out from the rows and assigns them to the controls using the Swing API. As a result, the Swing API updates the panel UI with the data.

4.9.2 Updating Data through Controls

The user's interaction with an ADF Swing-bound control may cause ADF Business Components to update the data. For example, in the case of the text field (`textFieldDname`), if the user edits the text field's content and leaves the control (generating `focusLost` event), ADF Swing is notified of the event. As a result, ADF Swing will retrieve the updated data from the control and call `setAttribute()` on the row.

Customizing ADF Bindings

This chapter describes how to use ADF Model binding editors to customize control bindings in ADF Swing applications. The control is bound to the data model using ADF bindings. JDeveloper creates ADF bindings when you insert a control from the Data Controls panel.

This chapter includes the following sections:

- [Section 5.1, "About Customizing ADF Bindings"](#)
- [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels"](#)
- [Section 5.3, "How to Customize an ADF Action Binding"](#)
- [Section 5.4, "How to Customize an ADF Attribute Binding"](#)
- [Section 5.5, "How to Customize an ADF Array Combobox Binding"](#)
- [Section 5.6, "How to Customize an ADF Boolean Binding"](#)
- [Section 5.7, "How to Customize an ADF Bounded Range Binding"](#)
- [Section 5.8, "How to Customize an ADF Formatted Text Field Binding"](#)
- [Section 5.9, "How to Customize an ADF Iterator Binding"](#)
- [Section 5.10, "How to Customize an ADF List Binding"](#)
- [Section 5.11, "How to Customize an ADF List Binding in Enumeration Mode"](#)
- [Section 5.12, "How to Customize an ADF List Binding in LOV Mode"](#)
- [Section 5.13, "How to Customize an ADF LOV Button Binding"](#)
- [Section 5.14, "What You May Need to Know About the LOV Dialog"](#)
- [Section 5.15, "How to Customize an ADF Scroll Binding"](#)
- [Section 5.16, "How to Customize an ADF Table Binding"](#)
- [Section 5.17, "How to Customize an ADF Tree Binding"](#)

5.1 About Customizing ADF Bindings

When you insert a control from the Data Controls panel, the control is bound to the data model using ADF bindings. You can then edit the bindings using the binding editors.

ADF Swing provides model objects for Swing controls that are responsible for marshaling interaction between the Swing controls and the ADF Business Components view object's row set iterator. The ADF Swing implementation of Swing models are called *control bindings*.

The UI components in an ADF Swing application bind to ADF Business Components view objects. For example, an ADF Swing databound panel might have a text field (`JTextField`) for the First name and Last name that allows the user to view and modify business component attribute values.

The type of control binding used for a given Swing control depends on the actions performed by the control. In some case, controls work with multiple control bindings that define different interactions. [Table 5–1](#) shows the bindings that ADF Swing defines for the various Swing controls.

Table 5–1 Control Bindings for Controls in ADF Swing

ADF Control Binding	Swing Control
Action binding	Button
Attribute binding	Label For Password TextArea TextField TextPane EditPane
Array Combobox binding	JUArrayComboBox
Boolean binding	Checkbox
Bounded Range binding	ProgressBar ScrollBar Slider
Formatted Text Field binding	Formatted Text Field
Iterator binding	Navigation Bar
List binding	ComboBox List Radio Button Group Spinner
List binding in LOV Mode	Spinner
Button LOV binding	Button LOV
Scroll binding	ScrollBar Slider
Table binding	Table
Tree Node binding	Tree
Graph binding	JUSingleTableGraphBinding JUMasterDetailTableGraphBinding

5.2 How to Customize ADF Bindings for ADF Swing Panels

You can use the control binding editors in JDeveloper to customize the characteristics of any databound UI component that you create using the Data Controls panel.

Note: Modifications to a data binding may create discrepancies between the data binding and the UI component used to display the data. For example, if you edit the data binding of a table to display one less column of attribute values, you must use the Java visual editor to remove the column from the source.

To customize a UI component's binding from the Java visual editor:

1. Use the Data Controls panel to insert the UI component into your Java panel.

For more information, see [Section 2.11, "How to Create ADF Swing Edit Forms from the Data Controls Panel."](#)

2. With the UI component displayed in the Java visual editor, right-click the component and choose **Edit Bindings** to view the binding editor.

You can also click the **Bindings** tab in the Java visual editor and double-click the binding in the **Bindings** list.

5.3 How to Customize an ADF Action Binding

You can customize an ADF action binding on the Button UI control that you insert from the Data Controls panel.

An action binding lets users initiate actions on the attributes and collections of the specific business service. Actions are defined by the business service's class methods and will appear in the Operations folders displayed in the Data Controls panel.

When you use the Data Controls panel to insert the action as a button, the action binding editor displays the corresponding selections. You can use the action binding editor to change the data collection and action. Or, when you want the action to apply to all data collections in the current document's binding context, you can select a data control and a corresponding action.

Note: If the custom method accepts parameters, the actual value for parameters may be specified through the Properties window. The action binding editor does not support entering value for the method arguments.

For certain business services, the ADF data control for that business service may support standard actions. For example, in the case of ADF Business Components, these standard actions are available:

- Commit or rollback the changes to all bound data collections in the binding context of the data control.
- Move to the first, next, previous, or last row in the data collection's range.
- Create a row or delete the current row.
- Reset data from the data collection cache on all rows.
- Execute the data collection query to get the latest data from the database.
- Initiate a query on the data collection.
- Obtain the current row from the data collection.

When users initiate the action, the bound data collection is immediately updated. The UI reflects the change through any control bindings that use the same data collection as the action binding.

To set an action binding:

1. Open the data panel in the Java visual editor.

For more information, see [Section 2.9, "How to Create an Empty ADF Swing Form."](#)

2. In the Data Controls panel, expand the **Operations** folder for the data collection or data control.

Note: The Data Controls panel hierarchy represents operations (such as **Create** and **Delete**) that apply to a specific data collection in the Operations folder below the data collection. When supported by the ADF data control for your business service, you can also select operations (such as **Commit** and **Rollback**) that apply to all data collections in the current document's binding context in the Operations folder at the top branch of the hierarchy.

3. Drag the desired operation into the open document.

JDeveloper adds code to the class file to bind the component to the operation.

4. Display the action binding editor for the control.

For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)

5. In the **Data Collection** list, select the collection or data control on which you want to perform the action.

Expand the **Operations** folder under the root data control node when you want to select operations (such as **Commit** and **Rollback**) that apply to all data collections in the current document's binding context.

Note: Add other UI controls to your data panel to display the results of the action on the data collection. Those controls need only set a control binding on the same data collection as the action binding to reflect the action in the UI.

6. Select the **Operation** for the action to perform on the selected data collection.

7. You should leave **Iterator** empty if you have selected a custom method, such as **Commit** or **Rollback**, which are actions on the ADF data control and do not require an iterator.

If the **Iterator** list already displays a named iterator to access the selected data collection, you may leave the selection unchanged, or if you have selected one of the predefined actions provided by the business service data collection (for example, Next, Next Set, Previous, and Previous Set), click **New** and create the iterator so it appears in the dropdown list.

8. Click **OK** to save the binding settings.

9. Open the Properties window to define any method parameter values.

You can also create an ADF Swing form that allows the user to supply the parameters of the method. For more information, see [Section 3.8, "Binding a Method with Parameters in an ADF Swing Form."](#)

JDeveloper adds the `setModel()` method in the `jbInit()` method to create the action binding. [Example 5-1](#) shows the method which references `DataControlId` to specify the metadata after inserting a button from the Data Controls panel.

Example 5-1 Method Referencing DataControlId After Inserting Button

```
myButton.setModel((ButtonModel)panelBinding.bindUIControl("DataControlId",
myButton));
```

Metadata for the new binding appears in the page definition file (`PageDef.xml`).

Notes: The custom method argument definitions, if any, appear undefined until you use the Properties window to specify the values.

By default when you create an action binding for the same method more than once, the return location of the method is the same. This means if you want to create unique action bindings for the same method, you must edit the `ReturnName` attribute to supply a unique name for each binding. Normally, you will leave the location the same for duplicate usages of the same action binding (where each usage specifies different parameter values). This permits all bound controls in the binding context to find the result under the same return name.

5.4 How to Customize an ADF Attribute Binding

You can set an ADF attribute binding on these basic UI components that you insert from the Data Controls panel:

- **Password Field** - mask the attribute value entered by the user
- **TextArea** - display plain text with multiple lines
- **TextField** - display plain text in a single line
- **JUIImage** - display an attribute of type BLOB or OrdMedia
- **JULabel** - display the attribute value as a label
- **Label For** - display the control hint label defined for ADF Business Components attributes

You can set an ADF attribute binding on these UI components to display various kinds of content (besides text):

- **Edit Pane** - display various kinds of components that can be edited
- **TextPane** - display various kinds of components, that should not be editable

The behavior of the attribute binding depends on the type of control used. Users may view and, in some cases, edit the value of a single attribute defined by a data collection. You use the attribute binding editor to select the data collection and attribute.

Note: In an ADF Business Components project, you can make attribute values updatable by setting a control hint on the attribute. In that case, users will be able to edit the updatable attribute's values directly.

To set an attribute binding:

1. Open the data panel in the Java visual editor.
For more information, see to [Generating an Empty ADF Swing Panel](#).
2. In the Data Controls panel, drag the desired attribute to display into the open form or panel.
Be sure to select an attribute and not a data collection.
3. From the **Add Child** popup list, select the UI component that you want to add to the open document.
The new UI component appears in the document you are editing.
4. Display the attribute binding editor for the component.
For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)
5. In the attribute binding editor, select the **Data Source** that contains the attribute you want to display.
6. In the **Attribute** list, select a single attribute to display as the value of the control.
7. If the iterator dropdown list already displays a named iterator to access the selected data collection, leave the selection unchanged. If the dropdown appears empty, click **New** and create the iterator so it appears in the dropdown list.

Note: Changing the iterator selection in the dropdown list will remove the previously made attribute selection. Before you change the iterator selection, take note of the original attribute selection. If you need to, you can press **Cancel** to exit the binding editor without updating the original attribute selection.

8. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-2](#) shows the method created after inserting a text field from the Data Controls panel.

Example 5-2 Set Model Method for Attribute Binding

```
myTextField.setDocument((Document)panelBinding.bindUIControl("DataControlId", myTextField));
```

Metadata for the new binding appears in the binding definition (`PageDef.xml`).

5.5 How to Customize an ADF Array Combobox Binding

You can set an ADF array combobox binding on the `JUArrayComboBox` UI control that you insert from the Data Controls panel.

The array combobox binding lets the user view and, in some cases, edit the values displayed from a list defined by an attribute that specifies an array. You use the array binding editor to select the data collection and attribute.

Note: In an ADF Business Components project, you can make attribute values updatable by setting a control hint on the attribute. In that case, users will be able to edit the updatable attribute's values directly.

To set an array combobox binding:

1. Open the data panel in the Java visual editor.

For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)

2. In the Data Controls panel, drag the attribute to display into the open form or panel.

The attribute must be an attribute of a business service data collection that has been mapped as type `oracle.jbo.domain.array` in the data model project.

3. From the **Add Child** popup list, select **JUArrayComboBox**.

The new UI component appears in the document you are editing.

4. Display the array combobox binding editor for the component.

For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)

5. In the array combobox binding editor, select the **Base Data Source** that contains the attribute you want to display.
6. In the **Display Attribute** list, select a single attribute of type array to display the values in the combobox control.
7. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-3](#) shows the method created after inserting the `JUArrayComboBox` from the Data Controls panel.

Example 5-3 Set Model Method for Array Binding

```
juArrayComboBox.setArrayBinding((JUDefaultControlBinding)panelBinding.  
bindUIControl("DataControlId", juArrayComboBox));
```

Metadata for the new binding appears in the binding definition (`PageDef.xml`).

5.6 How to Customize an ADF Boolean Binding

You can set an ADF boolean binding on the checkbox UI component that you insert from the Data Controls panel.

The boolean binding lets users select the component and update an attribute in a data collection based on the component's selection state. You use the boolean binding editor to select the data collection and attribute on which you want the component to operate, then specify values corresponding to the component's selection state (for example, "true" for selected and "false" for unselected). You must know what values the bound attribute takes in order to supply meaningful values.

To set a boolean binding:

1. Open the data panel in the Java visual editor.
For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)
2. In the Data Controls panel, drag the attribute to display into the open form or panel.
Be sure to select an attribute and not a data collection.
3. From the **Add Child** popup list, select **Check Box**.
The new UI component appears in the document you are editing.
4. Display the boolean binding editor for the component.
For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)
5. In the **Base Data Source** list, select the data collection which contains the attribute you want to update.
6. In the **Attribute** list, select the attribute to display with the values list.
7. When you want to supply the values from a list binding, select the **Server List Binding Name** from the dropdown list.
8. In the **Selected State Value** field, enter the value the model will use to update the attribute when the user makes the control appear selected.
9. In the **Unselected State Value** field, enter the value the model will use to update the attribute when the user makes the control appear unselected.
10. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-4](#) shows the method created after inserting a checkbox from the Data Controls panel.

Example 5-4 Set Model Method for Boolean Binding

```
myCheckbox.setModel((ButtonModel)panelBinding.bindUIControl("DataControlId",  
myCheckbox));
```

Metadata for the new binding appears in the binding definition file (`PageDef.xml`).

5.7 How to Customize an ADF Bounded Range Binding

You can set an ADF bounded range binding on these UI controls that you insert from the Data Controls panel:

- `ProgressBar`
- `ScrollBar`
- `Slider`

The behavior of the bounded range control binding depends on the type of control used. Users may view and, in some cases, edit the value of a single attribute defined by a data collection. You use the bounded range binding editor to select the data collection and attribute, then define the range of permissible values. You must know what values the bound attribute takes in order to supply a meaningful range.

Note: In an ADF Business Components project, you can make attribute values updatable by setting a control hint on the attribute. In that case, users will be able to edit the updatable attribute's values directly.

To set a range binding:

1. Open the data panel in the Java visual editor.
For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)
2. In Data Controls panel, drag the desired attribute to display into the open form or panel.
Be sure to select an attribute and not a data collection.
3. Add the control to the data panel.
For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)
4. Display the bounded range binding editor for the control.
For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)
5. In the bounded range binding editor, select the **Base Data Source** that contains the attributes you want to display in the table.
6. In the **Attributes** list, select a single attribute to display as the value of the control.
7. If the iterator dropdown list already displays a named iterator to access the selected data collection, leave the selection unchanged. If the dropdown appears empty, click **New** and create the iterator so it appears in the dropdown list.

Note: Changing the iterator selection in the dropdown list will remove the previously made attribute selection. Before you change the iterator selection, take note of the original attribute selection. If you need to, you can press **Cancel** to exit the binding editor without updating the original attribute selection.

8. In the case of a ProgressBar, Slide, and ScrollBar control, you can specify a **Minimum Display Value** and a **Maximum Display Value**. These correspond to the start and end values supported by the control.
9. In the case of the Slider and ScrollBar control, you can specify the **Extent Value** or interval between tick marks. Each tick mark corresponds to a possible value that the user could apply to the bound attribute.

Note: In the case of the slider and scrollbar, attribute values are calculated by adding the control's minimum value to the product of the increment value and index of the selected tick mark. For example, if the minimum value is 20, the increment value is 10, and the user has selected the 3rd tick mark in the control, then the attribute value would be $20 + (10 \times 3) = 50$.

10. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-5](#) shows the method created after inserting a slider from the Data Controls panel.

Example 5-5 Set Model Method for Bounded Range Binding

```
mySlider.setModel((BoundedRangeModel)panelBinding.bindUIControl("DataControlId",  
mySlider));
```

Metadata for the new binding appears in the binding definition file (`PageDef.xml`).

5.8 How to Customize an ADF Formatted Text Field Binding

You can set an ADF formatted text field binding on the Formatted Edit Field UI component that you insert from the Data Controls panel.

The formatted text field binding lets the user view and edit the value of a single attribute defined by a data collection. You use the formatted text field binding editor to select the data collection and attribute. The editor also lets you select a formatter to display the bound attribute's value with a specific format mask applied.

Note: In an ADF Business Components project, you can make attribute values updatable by setting a control hint on the attribute. In that case, users will be able to edit the updatable attribute's values directly.

To set a formatted text field binding:

1. Open the data panel in the Java visual editor.
For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)
2. In the Data Controls panel, drag the attribute to display into the open form or panel.
Be sure to select an attribute and not a data collection.
3. Add the control to the data panel.
For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)
4. Display the formatted text field binding editor for the control.
For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)
5. In the formatted text field binding editor, select **Attribute** in the topmost dropdown list, and select the **Data Source** that contains the attribute you want to display.
6. In the **Attribute** list, select a single attribute to display as the value of the control.
7. In the topmost dropdown list, select **Format** and then select the **Formatter** to use.
8. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-6](#) shows the method created after inserting a formatted edit field from the Data Controls panel.

Example 5-6 Set Model Method for Formatted Text Field Binding

```
jFormattedTextField.setDocument((Document)panelBinding.bindUIControl("DataControlId", jFormattedTextField));
```

Metadata for the new binding appears in the binding definition (`PageDef.xml`).

5.9 How to Customize an ADF Iterator Binding

You can set an ADF iterator binding on the `NavigationBar` UI control that you insert from the Data Controls panel.

You use an iterator binding on a `NavigationBar` control to manage the position of the current data object on the data collection. When the user clicks on the navigation bar buttons, the data object position changes and any other indicator control bound to the same data collection gets updated.

To set an iterator binding:

1. Open the data panel in the Java visual editor.
For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)
2. In the Data Controls panel, drag the data collection to navigate into the open form or panel.
3. Add the `NavigationBar` control to the data panel.
For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-7](#) shows the method created after inserting a navigation bar from the Data Controls panel.

Example 5-7 Set Model Method for Iterator Binding

```
jUNavigationBar.setModel(JUNavigationBar.createViewBinding(panelBinding, jUNavigationBar, "SelectedDataCollection", null, "IteratorId"));
```

Metadata for the new binding appears in the binding definition file (`PageDef.xml`).

To modify the number of rows the iterator can display:

1. In the Applications window, right-click the data panel that contains the component for which you want to limit the range size and select **Open**.
2. In the Java visual editor, click the **Bindings** tab and then from the **Executables** list select iterator corresponding to the navigation bar and click **Edit**.
In most cases, you can select the control binding in the **Bindings** list to identify the corresponding iterator in the **Executables** list. Navigation bars are an exception since they do not require a control binding.
3. In the Properties window, expand the **Common** section and edit the **RangeSize** value and press **Enter**.

You can increase the value from the default value of 10 rows.

Note: The value -1 and 0 have specific meaning: the value -1 returns all available objects from the collection, while the value 0 will return the same number of objects as the collection uses to retrieve from its data source.

To sort the columns in a table bound to an ADF iterator binding:

1. In the Applications window, right-click the data panel that contains the table you want to sort and select **Open**.
2. In the Java visual editor, click the **Bindings** tab and then from the **Executables** list select the iterator for the data collection to which the table is bound and click **Edit**.

In most cases, you can select the control binding in the **Bindings** list to identify the corresponding iterator in the **Executables** list. Navigation bars are an exception since they do not require a control binding.
3. In the Edit Iterator Binding dialog, click the **Sort Criteria** tab and use the arrow buttons to change the sort order for the columns in the table.

5.10 How to Customize an ADF List Binding

You can set an ADF list binding on the list UI control that you insert from the Data Controls panel.

A list binding lets users view a list consisting of attribute values from a data collection. You use the list binding editor to select the data collection and attributes to display.

To set a list binding:

1. Open the data panel in the Java visual editor.

For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)
2. In the Data Controls panel, drag the data collection to display into the open form or panel.
3. Add the list control to the data panel.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

Note: When you use the Data Controls panel to drop a JList control into an ADF Swing panel or form, the list will initially display all attributes of the selected collection. To modify the list of display attributes, you can use the List Binding Editor.

4. Display the list binding editor for the control.

For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)
5. In list binding editor, select the **Base Data Collection** that contains the attributes you want to display in the list.
6. In the **Display Attributes** list, select the attributes to display.

You can add as many attributes as you like to the **Display Attributes** list.

7. If the iterator dropdown list already displays a named iterator to access the selected data collection, leave the selection unchanged. If the dropdown appears empty, click **New** and create the iterator so it appears in the dropdown list.

Note: Changing the iterator selection in the dropdown list will remove previously made attribute selections. Before you change the iterator selection, take note of the original attribute selections. If you need to, you can press **Cancel** to exit the binding editor without updating the original attribute selections.

8. To rearrange the attributes of the list, select an attribute in the **Display Attributes** list and click a **Move Selection** arrow button to reposition the attribute in the list.

The position of the attribute from top to bottom in the list determines the position of the attributes in the list from top to bottom.

9. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-8](#) shows the method created after inserting a list from the Data Control panel.

Example 5-8 Set Model Method for List Binding

```
myList.setModel((ListModel)panelBinding.bindUIControl("DataControlId", myList));
```

Metadata for the new binding appears in the binding definition file (`PageDef.xml`).

5.11 How to Customize an ADF List Binding in Enumeration Mode

You can set an ADF list binding in enumeration mode on these UI controls that you insert from the Data Controls panel:

- ComboBox
- List
- Radio Button Group
- Spinner

An enumeration mode binding lets users select a value from a display list to update an attribute in a data collection. You use the model list binding editor in enumeration mode to select the data collection and attribute on which you want the control to operate, then specify a set of values from which the user may select. You must know what values the bound attribute takes in order to supply meaningful choices.

To set a list binding in enumeration mode:

1. Open the data panel in the Java visual editor.

For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)
2. In the Data Controls panel, drag the desired attribute to display into the open form or panel.

Be sure to select an attribute and not a data collection.
3. Add the control to the data panel.

For more information, see to [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

4. Display the list binding editor for the control.

For more information, see to [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)

5. In the **Data Collection** list, select the data collection that contains the attribute you want to update.
6. In the **Attribute** list, select the attribute to display with the values list.
7. If the iterator dropdown list already displays a named iterator to access the selected data collection, leave the selection unchanged. If the dropdown appears empty, click **New** and create the iterator so it appears in the dropdown list.

Note: Changing the iterator selection in the dropdown list will remove the previously made attribute selection. Before you change the iterator selection, take note of the original attribute selection. If you need to, you can press **Cancel** to exit the binding editor without updating the original attribute selection.

8. If the control is a combobox, list, or radio button group, enter the values in **Set of Values** in the order you want the control to display them. Press **Enter** to set the value and begin typing a new value. Values you supply must be valid for the attribute.
9. If the control is a spinner, select **Range of Values** when you want the user to be able to choose from a range that you define. If you prefer to define a specific list, select **Static Values** and enter the values as described for list and combobox above.
10. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-9](#) shows the method created after inserting a spinner from the Data Controls panel.

Example 5-9 Set Model Method for Spinner Binding

```
mySpinner.setModel((SpinnerModel)panelBinding.bindUIControl("DataControlId",
mySpinner));
```

Metadata for the new binding appears in the binding definition file (`PageDef.xml`).

5.12 How to Customize an ADF List Binding in LOV Mode

You can set an ADF list binding in LOV (list of value) mode on the Button LOV UI control that you insert from the Data Controls panel.

The Button LOV provides both the UI to display the selection and the UI (in the form of a separate window) to make the selection. All other controls require that you add a separate control to display the target of the user's selection.

A LOV mode binding lets users choose a value from a list that displays the data collection rows of one or more attributes. When the user makes the selection, the LOV updates one or more attributes of another data collection based on their selection. You use the list binding editor in LOV mode to define the source and target data

collections, the binding between their attributes, and the attributes to display in the LOV.

To set a list binding in LOV mode:

1. Open the data panel in the Java visual editor.
For more information, see [Generating an Empty ADF Swing Panel](#).
2. In the Data Controls panel, drag the attribute to display into the open form or panel.
Be sure to select an attribute and not a data collection.
3. Add the control to the data panel.
For more information see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

Note: Move the Button LOV on the panel in a position near the form field that you want to receive the LOV selection.

4. Display the list binding editor for the control.
For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)

To define the LOV to display:

1. In the list binding editor, select the **Binding Definition** tab and select the data collections you want your LOV to use:
 - **Base Data Source** defines the collection that you want to use to display your list of values for selection. This should be a collection that is not constrained by a master data collection in the data model project.
 - **List Data Collection** is the collection that contains the attribute you want to receive the selected value in the ADF Swing panel. This should be the same data collection that your panel displays.
2. If the iterator dropdown list already displays a named iterator to access the selected data collection, leave the selection unchanged. If the dropdown appears empty, click **New** and create the iterator so it appears in the dropdown list.

Note: Changing the iterator selection in the dropdown list will remove previously made attribute selections. Before you change the iterator selection, take note of the original attribute selections. If you need to, you can press **Cancel** to exit the binding editor without updating the original attribute selections.

3. Click **Add** to bind at least one attribute between the two data collections.
The bottom area of the list binding editor displays a table with a list of possible LOV binding attributes between the collection used to display the list of values and the collection used to receive the attribute selection.
4. Choose the attribute from the **LOV Attributes** dropdown that you want to use to supply the value to the field displayed in the ADF Swing form.

5. Choose the attribute from the **Target Attributes** dropdown that you want to receive the value from the LOV.
6. Click **Add** again to bind multiple attributes through the same LOV.
7. Select the **LOV Display Attributes** tab and select the attributes that you want the LOV window to display.

You can add as many attributes as you like to the **Selected Attributes** list, although you are not required to include the LOV binding attribute (the attribute that you want to display the selected value).

8. Optionally, select the **NULL Value Selection** tab and supply a NULL value when the attribute supports it.

This is the item the user may select when no other value applies

9. Optionally, select the **Runtime LOV Dialog Details** tab and customize the display location and size of the LOV dialog.
10. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-10](#) shows the method created after inserting a list from the Data Controls panel.

Example 5-10 Set Model Method for List Binding

```
myList.setModel((ListModel)panelBinding.bindUIControl("DataControlId", myList));
```

Metadata for the new binding is added to the binding definition file (`PageDef.xml`).

To insert a control to display the updated LOV target attribute value:

1. In the Data Controls panel, drag the attribute which you previously selected as the LOV binding's target attribute into the open panel.
2. From the **Add Child** popup list, select the UI component that you want to insert to display the updated attribute value.

To modify the number of rows to display in the LOV:

1. In the Applications window, right-click the data panel that contains the LOV component for which you want to limit the range size and select **Open**.
2. In the Java visual editor, click the **Bindings** tab and then from the **Executables** list select iterator corresponding to the LOV component binding and click **Edit**.

In most cases, you can select the control binding in the **Bindings** list to identify the corresponding iterator in the **Executables** list.

3. In the Properties window, expand the **Common** section and edit the **RangeSize** value and press **Enter**.

You can increase the value from the default value of 10 rows.

Note: The value -1 and 0 have specific meaning: the value -1 returns all available objects from the collection, while the value 0 will return the same number of objects as the collection uses to retrieve from its data source.

For additional information about working with range size, see [Section 10.4, "How to Limit Fetching of ADF Business Components Attributes in ADF Swing."](#)

5.13 How to Customize an ADF LOV Button Binding

You can set an ADF list binding in LOV (list of value) mode on the Button LOV UI control that you insert from the Data Controls panel.

The Button LOV provides both the UI to display the selection and the UI (in the form of a separate window) to make the selection. All other controls require that you add a separate control to display the target of the user's selection.

A LOV mode binding lets users choose a value from a list that displays the data collection rows of one or more attributes. When the user makes the selection, the LOV updates one or more attributes of another data collection based on their selection. You use the list binding editor in LOV mode to define the source and target data collections, the binding between their attributes, and the attributes to display in the LOV.

To set a list binding in LOV mode:

1. Open the data panel in the Java visual editor.

For more information, see [Generating an Empty ADF Swing Panel](#).

2. In the Data Controls panel, drag the attribute to display into the open form or panel.

Be sure to select an attribute and not a data collection.

3. Add the control to the data panel.

For more information see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

Note: Move the Button LOV on the panel in a position near the form field that you want to receive the LOV selection.

4. Display the list binding editor for the control.

For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)

To define the LOV to display:

1. In the list binding editor, select the **Binding Definition** tab and select the data collections you want your LOV to use:
 - **Base Data Source** defines the collection that you want to use to display your list of values for selection. This should be a collection that is not constrained by a master data collection in the data model project.
 - **List Data Collection** is the collection that contains the attribute you want to receive the selected value in the ADF Swing panel. This should be the same data collection that your panel displays.
2. If the iterator dropdown list already displays a named iterator to access the selected data collection, leave the selection unchanged. If the dropdown appears empty, click **New** and create the iterator so it appears in the dropdown list.

Note: Changing the iterator selection in the dropdown list will remove previously made attribute selections. Before you change the iterator selection, take note of the original attribute selections. If you need to, you can press **Cancel** to exit the binding editor without updating the original attribute selections.

3. Click **Add** to bind at least one attribute between the two data collections.
The bottom area of the list binding editor displays a table with a list of possible LOV binding attributes between the collection used to display the list of values and the collection used to receive the attribute selection.
4. Choose the attribute from the **LOV Attributes** dropdown that you want to use to supply the value to the field displayed in the ADF Swing form.
5. Choose the attribute from the **Target Attributes** dropdown that you want to receive the value from the LOV.
6. Click **Add** again to bind multiple attributes through the same LOV.
7. Select the **LOV Display Attributes** tab and select the attributes that you want the LOV window to display.
You can add as many attributes as you like to the **Selected Attributes** list, although you are not required to include the LOV binding attribute (the attribute that you want to display the selected value).
8. Optionally, select the **NULL Value Selection** tab and supply a NULL value when the attribute supports it.
This is the item the user may select when no other value applies
9. Optionally, select the **Runtime LOV Dialog Details** tab and customize the display location and size of the LOV dialog.
10. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-11](#) shows the method created after inserting a button LOV from the Data Controls panel.

Example 5-11 Set Model Method for LOV Button Binding

```
myList.setModel((ButtonModel)panelBinding.bindUIControl("DataControlId",
jButton1));
```

Metadata for the new binding is added to the binding definition file (`PageDef.xml`).

To insert a control to display the updated LOV target attribute value:

1. In the Data Controls panel, drag the attribute which you previously selected as the LOV binding's target attribute into the open panel.
2. From the **Add Child** popup list, select the UI component that you want to insert to display the updated attribute value.

To modify the number of rows to display in the LOV:

1. In the Applications window, right-click the data panel that contains the LOV component for which you want to limit the range size and select **Open**.

2. In the Java visual editor, click the **Bindings** tab and then from the **Executables** list select iterator corresponding to the LOV component and click **Edit**.

In most cases, you can select the control binding in the **Bindings** list to identify the corresponding iterator in the **Executables** list.

3. In the Properties window, expand the **Common** section and edit the **RangeSize** value and press **Enter**.

You can increase the value from the default value of 10 rows.

Note: The value -1 and 0 have specific meaning: the value -1 returns all available objects from the collection, while the value 0 will return the same number of objects as the collection uses to retrieve from its data source.

For additional information about working with range size, see [Section 10.4, "How to Limit Fetching of ADF Business Components Attributes in ADF Swing."](#)

5.14 What You May Need to Know About the LOV Dialog

The `JButton` control is unique when you specify an ADF Swing LOV (list of values) control binding because unlike other Swing controls that support the LOV control binding, a `JButton` control will display an LOV dialog. The dialog the user sees displays the attributes you specified for display in the LOV binding.

The LOV dialog supports several custom features:

- Find-mode operations that may be initiated by the user when this feature is enabled.
- Custom title and display location for dialog.
- Ability to subclass `JULovPanelInterface` to display your own LOV dialog when the user clicks the `JButton` control with the LOV binding.

When Find-mode operation is enabled for the LOV binding on the `JButton` control, the user can set the LOV dialog into Find mode by clicking the Find button in the LOV dialog navigation bar. The LOV dialog in Find mode lets the user enter view criteria to execute a query on the bound view object rows. The LOV binding executes the query and displays only the rows that match the specified view criteria.

The user executes a Find-mode query through a LOV dialog by:

1. Clicking **Set to Find** mode in the LOV dialog. navigation bar to display a single view criteria field for each bound attribute.
2. Entering values in the view criteria field for the attributes they want to match.
The entered value must include the appropriate comparison symbol (>, <, =). All values in the same view criteria are AND'ed together.
3. Clicking **Execute Query** Button in the LOV dialog navigation bar to return the results to the LOV dialog.

For details about the usage of view criteria in ADF Business Components, see the JavaDoc for the `oracle.jbo.ViewCriteria` class.

5.15 How to Customize an ADF Scroll Binding

You can set an ADF scroll binding on these UI controls that you insert from the Data Controls panel:

- ScrollBar
- Slider

The scroll binding lets users view the relative position of the current data object in the bound data collection. The control thumb or indicator will be proportional to the number of data objects displayed out of the full range of the data collection. You use the scroll binding editor to select the data collection on which you want the control to operate.

To set a scroll binding:

1. Open the data panel in the Java visual editor.
For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)
2. In the Data Controls panel, drag the data collection to scroll into the open form or panel.
3. Add the control to the data panel.
For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)
4. Display the scroll binding editor for the control.
For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)
5. In the scroll binding editor, select the **Base Data Source** that contains the data object to scroll.
6. If the iterator dropdown list already displays a named iterator to access the selected data collection, leave the selection unchanged. If the dropdown appears empty, click **New** and create the iterator so it appears in the dropdown list.

Note: Changing the iterator selection in the dropdown list will remove the previously made attribute selection. Before you change the iterator selection, take note of the original attribute selection. If you need to, you can press **Cancel** to exit the binding editor without updating the original attribute selection.

7. You can determine the data collection object (row) count to use. Normally, leave **Estimate Scrolling Range** selected. You can select it if you want to count any rows from the database which have not yet been cached by the data collection.

Note: This option is visible only when the data model project contains ADF business components; when other business services are created in the data model project, this option is not supported. In the case of an ADF Business Components data model project, this option appears selected by default because, when unselected (thereby forcing the actual row count), it may trigger an additional query.

8. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5–12](#) shows the method created after inserting a slider from the Data Controls panel.

Example 5–12 Set Model Method for Scroll Binding

```
mySlider.setModel((BoundedRangeModel)panelBinding.bindUIControl("DataControlId",
mySlider));
```

Metadata for the new binding appears in the binding definition file (`PageDef.xml`).

5.16 How to Customize an ADF Table Binding

You can set an ADF table binding on the Table UI control that you insert from the Data Controls panel.

A table binding lets users view a table consisting of attribute names (column headers) and values from a data collection. You use the table binding editor to select the data collection and attributes to display. You can also specify display properties for column attributes, including column width and a table cell editor to display.

Note: In an ADF Business Components data model project, you can make attribute values updatable by setting a control hint on the attribute. In that case, users will be able to edit the updatable attribute's values directly in the table. Another control hint lets you change the label displayed by the column header for each attribute.

To set a table binding:

1. Open the data panel in the Java visual editor.

For more information, see [Section 2.9, "How to Create an Empty ADF Swing Form."](#)

2. In the Data Controls panel, drag the desired data collection to display into the open form or panel.
3. Add the table control to the data panel.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

Note: When you use the Data Controls panel to drop a `JTable` control into an ADF Swing panel or form, the table will initially display all attributes (columns) of the selected collection. To modify the table's list of display attributes, you can use the Table Binding Editor, however, the changes will not be reflected immediately in the Java visual editor. In order to synchronize the visual editor with the table's binding information (saved in the form or panel's `PageDef.xml` file), you can either recompile the class file, or you can merely resize the table in the visual editor.

4. Display the table binding editor for the control.

For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)

5. In table binding editor, display the **Attribute Binding** page and select the **Base Data Source** that contains the attributes you want to display in the table.

Tables that you create must be based on scalar attributes of a collection or structured object. For example, if you attempt to drop the `return()` method of a complex collection (contains arrays) as a table, the collection's attributes will not be displayed because they are defined by accessors. You can drop individual accessors of the collection as a table because they define attributes.
6. In the **Available Attributes** list, select the attribute to display and add it to the **Selected Attributes** list. You may add as many attributes as you like to the Display Attributes list.
7. To rearrange the columns of the table, select an attribute in the **Selected Attributes** list and click a **Move Selection** arrow button to reposition the attribute in the list.

The position of the attribute from top to bottom in the list determines the position of the columns in the table from left to right.
8. Select **Attribute Properties** and optionally change the column width and cell editor to use.
9. Click **OK** to save the binding settings.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-13](#) shows the method created after inserting a table from the Data Controls panel.

Example 5-13 Set Model Method for Table Binding

```
myTable.setModel((TableModel)panelBinding.bindUIControl("DataControlId",  
myTable));
```

Metadata for the new binding appears in the binding definition file (`PageDef.xml`).

To customize table columns:

1. In table binding editor, display the **Attribute Properties** page and view the list of attribute name (columns) that your table will display.
2. Optionally, double-click in the **Width** field corresponding to the attribute whose column width you want to modify. The default width is determined by the size of the table displayed in the Java visual editor.

You can also specify column widths by setting control hints at the level of the business component attribute. When the control hint width is specified, that value will appear in the Attribute Properties page of the table binding editor.

3. Press **Return** to set the new width on the field.
4. Optionally, click in the **Editor** field corresponding to the attribute whose editor you want to set. The default editor is a simple text field.

Note: You can additionally specify format masks by setting the formatter and format control hints at the level of the business component attribute.

5. Select an editor from the list and press the ellipses button to open the binding editor.

In the case of the ComboBox and Spinner editors, you can select an existing binding from the page definition. The binding you select must be bound to the same attribute displayed in the table binding editor.

6. Click **OK** to save the binding settings.

5.17 How to Customize an ADF Tree Binding

You can set an ADF hierarchical tree binding on the tree UI control that you insert from the Data Controls panel.

A tree binding lets users view a hierarchical list of attributes derived from master-detail relationships as specified by the business services in your data model project. You use the tree binding editor to define a set of rules that determine how the tree binding should traverse the relationships of the business service data collections you select.

In order for the tree binding to construct a tree with multiple branches, the business services should meet these requirements:

- A starting or root data collection (for example, an ADF Business Components view object) that you will use as the control binding target.
- One or more accessors (for example, an ADF Business Components viewlink) from that root collection and other data collections. The tree binding editor will display an allowable set of accessors for that node to drill down in to the next level.

Once you create the ADF binding for the tree control, you can use the tree control to let users navigate through the rows of the bound data collection. Currently, there is no navigation binding for the tree control, so you will need to add a `JUTreeDefaultMouseListener` on the control.

To set a tree binding:

1. Open the data panel in the Java visual editor and add a `JScrollPane` container to the panel.

For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)

2. In the Data Controls panel, drag the data collection to display into the open form or panel.

3. Add a Tree control to the scrollable panel.

For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)

4. Display the tree binding editor for the `JTree` control.

For more information, see [Section 5.2, "How to Customize ADF Bindings for ADF Swing Panels."](#)

To define the root node of the tree:

1. In the tree binding editor, click the **Edit Rule** tab to define the parent node rule to display from the master data collection.

For example, a master collection of "customers" has a detail collection of "orders".

2. In the **Root Data Source** list, select the data collection that you want to use to populate the tree root node.
This is the master collection that defines the root node. For example, a customer's master collection that has an orders detail collection.
3. Click the **Add Rule** icon button to complete the rule definition for the root node.
4. In the **Attributes** list, select a single attribute to display as the parent nodes for the tree.
For example, the last name of each customer. Currently, the `JTree` control is limited to displaying a single attribute for each branch.
5. In the **Accessor** list, select the accessor that specifies the link between the master collection and the first branch in the tree.
For example, an ADF Business Components accessor for the customers and orders collections, might appear as `OrdersView` in the list.

Note: If your data model does not contain master-detail accessors for the data collections you select, the **Accessor** list will appear empty. You must define an accessor for your business service in your data model project for all branches of the tree except the leaf (terminal) nodes.

To define the branches of the tree:

1. Select the parent tree-level rule and click the **Add Rule** icon button and select the child-branch collection to display from the master collection.
This is the detail collection that defines the nodes for the tree. For example, an orders detail collection from which you will display order id information in the tree.
2. In the **Attributes** list, select a single attribute to display as the branch nodes in the tree, for example, the last name of each customer.
3. In the **Accessor** list, select the accessor that specifies the link between the first branch data collection and the next branch in the tree.
For example, an ADF Business Components accessor for the orders and orderitems collections might appear as `OrderItemsView` in the list.

Note: If the branch data collection you select has no further accessor defined for it in the data model project, the word `<none>` will appear in the list. This means no accessor is required because the selected collection's attribute will appear as leaf nodes in the tree.

4. You may repeat these steps to define new branches in the tree as long as your business services support accessors to traverse the branches.
5. Click **OK** to save the binding settings.
The binding editor performs error checking for the rules you defined. It displays the following error messages:
 - "Invalid definition, either no root or no rules defined." Either you did not define a rule or you did not define a rule for the root node.

- "Defined root not found in this Application Module." The view object usage name defining the root node cannot be found in the application module defined in the form binding.
- "No rule found for accessor... ." There is no rule definition for every accessor in the rules table. Error checking does not verify that the number of polymorphic rules for a particular accessor; only that each accessor in the rules table has a rule definition.

Developer adds the `setModel()` method in the `jbInit()` method to create the control binding and reference `DataControlId` to specify the metadata. [Example 5-14](#) shows the method created after inserting a tree from the Data Controls panel, the method which references `DataControlId` to specify the metadata.

Example 5-14 Set Model Method for Tree Binding

```
myTree.setModel((TreeModel)panelBinding.bindUIControl("DataControlId", myTree));
```

Metadata for the new binding appears in the binding definition file (`PageDef.xml`).

To add images to display in the tree binding editor:

The tree binding editor lets you choose icons to display for each node in a branch and the open and close icon for each branch. To view the icon choices in the tree binding editor, you must copy your icon image files to the `src` folder of your user interface project. Use the icon dropdown menus in the tree binding editor to make your choice; each menu displays all the image files in the current `project/src` folder.

Displaying Graphs in ADF Swing Panels

This chapter describes how to create databound graphs in ADF Swing panels. The graph component allows you to display data through a wide variety of graphs.

This chapter contains the following sections:

- [Section 6.1, "About Graphs in ADF Swing Panels"](#)
- [Section 6.2, "How to Create a Graph for an ADF Swing Panel"](#)
- [Section 6.3, "What Happens When You Create a Graph Component"](#)
- [Section 6.4, "How to Customize the Graph Component"](#)
- [Section 6.5, "How to Change Graph Data"](#)

6.1 About Graphs in ADF Swing Panels

JDeveloper lets you create a graph component that binds with data sources from any of the business services supported in Oracle ADF. Supported business services include ADF Business Components, Enterprise JavaBeans (EJB), Toplink, Web Services, and custom JavaBeans.

Note: In order to work with graphs, your business service must be able to provide data in the format expected by the graph component, as described in [Section 6.2, "How to Create a Graph for an ADF Swing Panel."](#)

You can use the JDeveloper Data Controls panel to drag data model components such as ADF Business Components view objects onto panels. At any time after creating an ADF databound graph component, you can change the graph properties using the design-time facilities of JDeveloper.

JDeveloper also provides support for building databound graphs through its seamless integration of Oracle Business Intelligence Beans (BI Beans) technology. Although you do not need to install the BI BeansJDeveloper addin component to use the graph technology, you may want to install the addin to view the available documentation.

The graph component capabilities and features include:

- Easy to integrate with custom applications
Since a graph component is reusable, you can add it to custom applications, including ADF web applications (currently JSP pages only) and ADF Swing applications.
- Easy to build and maintain

You can use the Data Controls panel to drag data model components such as ADF Business Components view objects onto the form. The graph properties and databinding are exposed in the Properties window to enable the developer to quickly change these settings.

- Programmatic access through Java

You can modify graph components by calling the BI Beans Graph component's public methods.

- Support for different data sources

ADF Business Components, Enterprise JavaBeans, Toplink, Web Services, custom Java Beans

- Generate simple and master-detail graphs
- Small downloads
- Excellent performance when deployed

The ADF Swing `oracle.jbo.uicli.jui` package includes the following classes which are related to the databound graph component:

- `JUGraphBinding`
- `JUSingleTableGraphBinding`
- `JUMasterDetailGraphBinding`

Table 6–1 suggests graph types for different aspects of data that you want to emphasize in a graph.

Table 6–1 Graph Types

Data Aspect	Example	Graph Type
The difference between items over time or at the same point in time	Which products are selling best in which stores?	Clustered bar or 3-D bar
Trends over time	Is our market share increasing or decreasing?	Absolute line or Stacked area
Cyclical trends over time	Have the sales of this product peaked in March consistently over the past three years?	Radar graph
Rate of data change	How fast is our market share increasing?	Absolute line or Stacked area
Percentage or changes in percentage	How much of our revenue comes from each product line? Is our big seller bringing in the same percentage of revenue as it did last year at this time?	Pie or Percentage bar, line, or are
Relationship of parts to the whole	Which products' sales are most closely related to our total sales? Which products' sales follow the trend of the total sales? Which ones do not?	Pie, Percentage bar or line, or Stacked bar, line, or area graph.
Relationship of parts to the whole, with more detail for one of the parts	Which product line accounts for the highest percentage of sales? Within that product line, which districts make up the most sales?	Pie-bar

Table 6–1 (Cont.) Graph Types

Data Aspect	Example	Graph Type
Changes in all parts of a whole	Do the products that bring the most revenue to the entire company bring in the most revenue for each district?	Multiple pie or Stacked bar, line, or area
Relationships between two variables	Do sales increase when we spend more on marketing?	Scatter or Polar
Relationships between three variables	Do sales and profits increase proportionally when we spend more on marketing?	Bubble
Location of defects in a system	Where are the highest percentage of defects occurring?	Pareto

6.2 How to Create a Graph for an ADF Swing Panel

You can set an ADF graph binding for a graph component that you insert from the Data Controls panel into an ADF Swing panel.

To understand the selections you make when building the graph component, you must understand how the graph component obtains sufficient data from your ADF Business Components data sources to:

- Plot each graph marker (as determined by up to three data points)
- Label each graph group, for example, the months of the year
- Draw either a single series or a multi-series type graph
- Label each graph

The data binding is determined in the Graph Editor by the data model components that you select (for example, the attributes of an ADF Business Components view object). The Graph Editor helps you choose objects and attributes for a particular graph type, but you must be familiar with the data model in order to make the specific choices. In general, when you make a graph type selection, the Graph Editor displays one attribute for each data point value needed to draw the markers for your graph type. If you select a group type graph, the Graph Editor also prompts you to choose an accessor that links the master data collection with the correct detail data collection.

For instance, if you want to plot a Hi-Lo-Close (HLC) Stock graph type for multiple stocks in the Graph Editor, you must choose:

- The master data collection that contains one row for stock type
- The detail data collection that contains the rows corresponding to the data of each stock
- The attribute from the master data collection that specifies the label for each stock graph
- One attribute from the detail data collection for each marker data value (in this case, Hi, Low, and Close)

Whereas, if you decide to plot a single stock as a simple bar graph that shows monthly high values in the Graph Editor, you must choose:

- An unconstrained data collection (one that does not belong to a master-detail relationship)
- One attribute that provides the label for each month

- One attribute for the marker data value (in this case, Hi value only)

It is helpful to understand how the graph component interprets the data from the business services data model. In general, the requirements of the data model for your data model project depend on:

- Whether the graph type you select requires one, two, or three data point values to determine each graph marker
- Whether you want to plot a series (single table) or a group type (master-detail) graph

For example, assume you want a simple graph, such as a bar graph with the data shown in [Example 6-1](#).

Example 6-1 Bar Graph Data

ENAME	SAL	Comm
KING	1000	200
CLARK	2000	100
MILLER	1500	50

Each row in the table corresponds to a series in the graph and each column corresponds to a group.

Some types of graphs, like the bar graph, require one value per marker. This is in contrast to other graph types, like the stock HLC graph, which require three values per marker (high, low and close). When your graph requires multiple data values, it is convenient to store them in separate rows in your database table, as shown in [Example 6-2](#).

Example 6-2 Storing Multiple Data Values

Date	High	Low	Close
10 Jun 03	11	10	11
11 Jun 03	11	7	9
12 Jun 03	10	9	9.5

The graph component supports the type of graph whose data is stored in a single table. When your graph requires multiple values, then the data for the chart should be modeled as a master/detail relationship. Each detail provides one series of data, corresponding to the master value. In the stock graph example above, the data model could look like [Example 6-3](#).

Example 6-3 Data Model

Master table stock_ticker_table

```

-----
ticker                symbol
Oracle Corporation    ORCL
XYZ Corporation       XYZ
    
```

Detail table stock_price_table

```

-----
ticker Date          High Low Close
ORCL   10 Jun 03        23  22  23
ORCL   11 Jun 03        24  23  23
ORCL   12 Jun 03        25  24  24
XYZ?   10 Jun 03         10   9   9
XYZ    11 Jun 03         10   9   9
XYZ    12 Jun 03         10   8   9
    
```

In a single-table graph, the data model is simple and need only contain:

- A standalone data collection, where each row plots one graph marker
- One attribute that supplies the label for the groups that the graph displays
- One attribute per graph data point, with as many attributes as required to plot each graph marker

In a more complex group type graph, the data model is based on a master-detail relationship and must contain:

- A master data collection, where each row specifies one series but contains no data
- One master data collection attribute that supplies the label for each series
- One detail data collection for each series, where each row plots one graph marker for that particular series
- One detail data collection attribute that supplies the label for the groups displayed by the graph
- One detail data collection attribute per graph data point, with as many attributes as required to draw each graph marker

To build a databound graph:

1. In the user interface project, open the Java visual editor on the data panel or form.
For more information, see [Section 3.3, "How to Insert UI Components into ADF Swing Panels."](#)
2. In the Data Controls panel, drag the collection you want to bind to the graph component into the open form or panel.
3. From the **Add Child** popup list, select **Graph**.
4. In the Java visual editor, click the **Bindings** tab, select the graph component binding from the list and click **Edit**.
5. In the Graph Editor, choose what specific data will appear in the graph such as the view selection, series attributes, and data attributes.

6.3 What Happens When You Create a Graph Component

When you create a graph component, JDeveloper generates files which describe the graph properties and associated data binding information. JDeveloper generates and updates the files shown in [Table 6-2](#).

Table 6-2 Files Generated for Databound Graph Components

File	Description
<code><object>.java</code>	The Java implementation of the graph component which lets you programmatically set and get graph properties.
<code>BIGraphDef.xml</code>	The graph properties definition file.
<code><webpageName>PageDef.xml</code>	The ADF bindings definition file containing the ADF data binding information for the graph.

The files are added to the project and are accessible from the Applications window. They reside in `/src/client` directory relative to the application.

[Example 6-4](#) shows a sample `PageDef.xml` file for a graph component in an application consisting of an ADF Swing panel and ADF Business Components for the business service.

Example 6-4 Sample PageDef.xml File for a Graph Component

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<DCContainer
  id="untitled1PageDef"
  xmlns="http://xmlns.oracle.com/adfm"
  Package="view" >
  <Contents >
    <DCIterator
      id="model_AppModuleDataControl_EmployeesView1Iter"
      Binds="model_AppModuleDataControl.EmployeesView1" >
    </DCIterator>
    <DCCControl
      id="EmployeesView1"
      DefClass="oracle.jbo.uicli.graph.JUSingleTableGraphDef"
      SubType="DCGraph"
      ControlClass="oracle.dss.graph.Graph"
      IterBinding="model_AppModuleDataControl_EmployeesView1Iter"
      SeriesType="SINGLE_SERIES"
      SeriesLabel="EmployeeId"
      GraphPropertiesFileName="view.BIGraphDef1" >
      <Contents >
        <AttrNames>
          <Item Value="EmployeeId" />
          <Item Value="Salary" />
          <Item Value="CommissionPct" />
          <Item Value="ManagerId" />
          <Item Value="DepartmentId" />
        </AttrNames>
        <ColumnLabels>
          <Item Value="EmployeeId" />
          <Item Value="Salary" />
          <Item Value="CommissionPct" />
          <Item Value="ManagerId" />
          <Item Value="DepartmentId" />
        </ColumnLabels>
      </Contents>
    </DCCControl>
  </Contents>
</DCContainer>
```

[Example 6-5](#) shows a sample `DataBindings.xml` file for a graph component in an application consisting of an ADF Swing panel and ADF Business Components for the business service.

Example 6-5 Sample DataBindings.xml file for a Graph Component

```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<JboProject
  id="DataBindings"
  xmlns="http://xmlns.oracle.com/adfm"
  SeparateXMLFiles="false"
  Package=""
  ClientType="Generic" >
  <Contents >
    <DataControl
```

```

        id="model_AppModuleDataControl"
        SupportsFindMode="true"
        SupportsTransactions="true"
        Package="model"
        Configuration="AppModuleLocal"
        FactoryClass="oracle.adf.model.bc4j.DataControlFactoryImpl" >
    </DataControl>
    <Containeer
        id="untitled1PageDef"
        ObjectType="BindingContainerReference"
        FullName="view.untitled1PageDef" >
    </Containeer>
</Contents>
</JboProject>

```

6.4 How to Customize the Graph Component

At any time after creating a databound graph component, you can change graph properties using the design-time facilities of the Graph Editor. The Graph Editor displays a visual representation of the databound graph using pseudo-data not actual data. For example, you may want the graph to display using a different graph type or graph style.

To edit an existing graph component:

1. In the Applications window, expand the user interface project and right-click **BIGraphDefx.xml** in the view package and choose **Open**.
2. In the Graph Editor, from the editor's toolbar, select either **Graph Type** or **Format Graph**.
3. When you select **Graph Type**, you can select another graph type to display the data or change the graph's appearance or style.
4. Click the appropriate tab to display its related settings.
5. When you select **Format Graph**, you can change the graph's titles, legend, plot area, and X- and Y-Axis settings.
6. Click the appropriate tab to display its related settings.
7. You can also edit the graph's properties by selecting an element in the Structure window and double-clicking to display its property settings in the Properties window.

Note: If an error message displays about insufficient data, select another graph type as the data model you specified does not contain enough data columns to render the graph. For example, line graphs require more than one data column. For a description of the available graph types, see [Section 6-1, "Graph Types."](#)

6.5 How to Change Graph Data

At any time after creating a graph for ADF Swing panels, you may decide to display other data in the graph. The Customize Graph Binding editor lets you select another data source for the graph. However, if the data source belongs in another data model you will need to create a new databound graph, as described in [Section 6.2, "How to Create a Graph for an ADF Swing Panel."](#)

To change the graph data:

1. In the Applications window, with the form containing the graph component open in the Java visual editor, select the **Bindings** tab.
2. In the Bindings and Executables page, select the graph binding and click **Edit**.
3. In Graph Editor, select another iterator from the dropdown list.

Working with ADF Swing Controls

This chapter describes how to use ADF Swing controls on ADF Swing forms and panels. In addition to the standard Swing controls that you can use to design ADF Swing applications, ADF Swing provides its own set of controls, called ADF Swing controls.

This chapter contains the following sections:

- [Section 7.1, "About ADF Swing-Specific Controls"](#)
- [Section 7.2, "How to Use the JUArrayComboBox Control"](#)
- [Section 7.3, "How to Use the JUImage Control"](#)
- [Section 7.4, "What You May Need to Know About Multimedia in ADF Swing Applications"](#)
- [Section 7.5, "How to Use the JULabel Control"](#)
- [Section 7.6, "How to Use the Label For Control"](#)
- [Section 7.7, "How to Use the JULovEditButton Control"](#)
- [Section 7.8, "How to Use the JUNavigationBar Control"](#)
- [Section 7.9, "How to Use the JUNavigationBar Control with Find Mode"](#)
- [Section 7.10, "How to Disable Find Mode for ADF Swing Controls in a Panel"](#)
- [Section 7.11, "What You May Need to Know About Iterator Bindings in Find Mode"](#)
- [Section 7.12, "How to Use the JURadioButtonGroupPanel Control"](#)
- [Section 7.13, "How to Use the JUShuttlePanel Control"](#)
- [Section 7.14, "How to Use the JUStatusBar Control"](#)

7.1 About ADF Swing-Specific Controls

In addition to the standard Swing controls that you can use to design ADF Swing applications, ADF Swing provides its own set of controls. These additional controls are available on the Data Controls panel:

- JUArrayComboBox
- JUImageControl
- JULabel
- JULabel - Label For

- JLOVEditButton
- JNavigationBar
- JURadioButtonGroupPanel
- JShuttlePanel
- JStatusBar

The type of control binding used for a given ADF Swing control depends on the actions performed by the control. [Table 7-1](#) identifies the bindings and the various ADF Swing controls that they support.

Table 7-1 Control Bindings and ADF Swing Controls

Binding	ADF Swing Control
Attribute binding	Label For JLOVEditButton (button for LOV dialog) JUImageControl JULabel
Array Combobox binding	JUArrayComboBox
Iterator binding	JNavigationBar JStatusBar
List Binding in Enumeration Mode	JShuttlePanel
List Binding in List of Values (LOV) Mode	JURadioButtonGroupPanel

Like the standard Swing controls that they supplement, these controls also rely on the MVC or Model-View-Controller architecture. The ADF Swing controls are unique because the controllers they rely on to manage end user interactions do not exist among the Swing controls. Together with the ADF model bindings, the ADF Swing controls allow you to design Java desktop applications with more diverse databound functionality.

7.2 How to Use the JUArrayComboBox Control

The array combobox is an ADF Swing `JUArrayComboBox` control that can be bound to an attribute of a business service data collection that has been mapped as type `oracle.jbo.domain.array`. Because the array combobox displays its list of values from a single attribute, it works with the ADF attribute binding, which accesses the data stored in the database through an attribute of a data collection. Currently, without design-time support in JDeveloper for editing Object arrays, the elements of the array attribute are limited to Scalar values, such as numbers, strings, or date.

Note: Although the `JUArrayComboBox` does not provide design-time support for using object arrays, you can create an extension in JDeveloper that implements the ADF Swing interface.

Users can use the array combobox to view the existing attribute values. They can also add, update, and delete displayed values.

To bind to an array attribute of type scalar:

1. In the user interface project, open the Java visual editor on the data panel or form.

2. In the Data Controls panel, drag the attribute you want to bind to the image control into the open form or panel.
3. From the **Add Child** popup list, select **JUArrayComboBox**.

JDeveloper adds code to the class file to bind the array combobox to the attribute:

```
myJUArrayComboBox1.setDocument((Document)panelBinding.  
bindUIControl("MyAttribute", myJuArrayComboBox1));
```

To insert a value to the array attribute:

1. In the Java visual editor, select the dropdown list and press **Ctrl + Click** to create a new empty list item.
2. In the empty list item, type the value and press **Enter**.

To update an existing value of the array attribute:

- Select the item, type the new value and press **Enter**.

To remove an existing value from the array attribute:

- Select the item and press **Delete**.

7.3 How to Use the JUImage Control

The image control is an ADF Swing `JUImage` control that can be bound to an attribute of a business service collection through the ADF Swing attribute binding. The attribute binding accesses images stored in the database through a particular data collection. Datatypes that the `JUImage` control supports includes images stored as:

- ADF Business Components domain `BLOB` datatype
- ADF Business Components domain `Raw` datatype
- Oracle *interMedia* domain `IMAGE` type

To insert a databound image control into a form or panel:

1. In the user interface project, open the Java visual editor on the desired data panel or form.
2. In the Data Controls panel, drag the attribute you want to bind to the image control into the open form or panel.
3. From the **Add Child** popup list, select **JUImageControl**.

JDeveloper adds code to the class file to reference the `MyImageAttribute` binding definition in the page definition file associated with the form:

```
myJUImageControl1.setModel((JUDefaultControlBinding)  
panelBinding.bindUIControl("MyImageAttribute", myJUImageControl1));
```

To add a scrolling region to your image:

1. In the Properties window, set the `autoscrolls` field to **True**.

Within an ADF Swing form, users can delete and update the image file stored in the database, but they cannot edit the image itself.

To use the JUImage control:

The Image control consists of a panel with a display area and two buttons:

- **Change** button to invoke an Open dialog box in which users can select the image file to load.
- **Clear** button to delete the image.

To change the data binding of an image control in a form or panel:

1. Open the form or panel containing the `JUIImage` control in the Java visual editor.
2. Right-click the `JUIImage` control you want to change and select **Edit Bindings** from the menu.

The `JUIImage` Control Binding dialog displays.

7.4 What You May Need to Know About Multimedia in ADF Swing Applications

The `JUIImage` ADF Swing control is available to access multimedia in the database. The `JUIImage` control is limited to images only.

Note: JDeveloper currently does not support audio and video usages of the `OrdMediaControl` component. This means that developers are not be able to create new applications using ADF Swing *interMedia* controls. However, the required *interMedia* libraries are still shipped with JDeveloper for backward compatibility. ADF Swing applications of previous JDeveloper releases that use *interMedia* will continue working when upgraded to ADF Swing.

The `JUIImage` control is an earlier implementation of a control in ADF Swing used to display images from the database. Your application's performance is always improved when you can stream the image from the database's supported *interMedia* object types. In the case of the `JUIImage` control, the image file must be entirely downloaded before the application can display it. Additionally, `JUIImage` is limited to RAW, LONG RAW, BLOB, and *interMedia* IMAGE types.

7.5 How to Use the `JULabel` Control

The `JULabel` is an ADF Swing control that can be bound to an attribute of a business service collection through the ADF Swing attribute binding. The attribute binding accesses the data stored in the database through a particular data collection. The `JULabel` control renders the attribute value as a label.

Or, you can display the label from a control hint that has been defined for ADF Business Components attributes. For more information, see [Section 7.6, "How to Use the Label For Control."](#)

To insert a databound label into a form or panel:

1. Open the data panel in the Java visual editor.
For more information, see [Section 2.10, "How to Create an Empty ADF Swing Panel."](#)
2. From the **Window** menu, choose **Applications**.
3. In the Applications window, expand the Data Controls panel.

4. In the Data Controls panel, drag the attribute you want to bind to the label into the open form or panel.
5. From the **Add Child** popup list, select **JULabel**.

JDeveloper adds code to the class file to bind the `JULabel` to the attribute:

```
myJuLabel.setModel((JULabelBinding)panelBinding.bindUIControl("MyAttribute", myJuLabel));
```

Note: The text property will be overwritten when you run the ADF Swing form.

To change the data binding of a label in a form or panel:

1. Open the form or panel containing the `JULabel` control in the Java visual editor.
2. Right-click the `JULabel` control you want to change and select **Edit Bindings** from the menu.

The Edit Attribute Binding dialog displays.

7.6 How to Use the Label For Control

The Label For control is an ADF Swing `JULabel` control that can be bound to an attribute of a business service collection through the ADF Swing attribute binding. The binding accesses the previously defined control hint defined for an attribute of an ADF Business Components view object. The `JULabel` control renders the label from the defined hint.

Or, you can display the label from the attribute value. For more information, see to [Section 7.5, "How to Use the JULabel Control."](#)

To insert a databound label into a form or panel:

1. In the user interface project, open the Java visual editor on the data panel or form.
2. In the Data Controls panel, drag the attribute you want to bind to the Label For control into the open form or panel.
3. From the **Add Child** popup list, select **Label For**.

JDeveloper adds code to the class file to bind the `JULabel` to the attribute:

```
myJULabel1.setText(panelBinding.findCtrlValueBinding("MyAttribute").getLabel());
```

Note: The **text** property will be overwritten when you run the ADF Swing form.

To change the data binding of a label in a form or panel:

1. Open the form or panel containing the label control in the Java visual editor.
2. Right-click the `JULabel` control you want to change and select **Edit Bindings** from the menu.

The Edit Attribute Binding dialog displays.

7.7 How to Use the JULovEditButton Control

The LOV edit button control is an ADF Swing `JULovEditButton` control that can be bound to an attribute of a business service collection through the ADF Swing attribute binding. The `JULovEditButton` control allows the user to make multiple selections for the displayed shuttle list.

To insert a databound LOV edit button into a form or panel:

1. In the user interface project, open the form or data panel in the Java visual editor.
2. In the Data Controls panel, drag the collection you want to bind to the `JULovEditButton` control into the open form or panel.
3. From the **Add Child** popup list, select **Button LOV**.

JDeveloper adds code to the class file to reference the `DCShuttle` binding definition in the page definition file associated with the form:

```
myjButton.setModel((ButtonModel)panelBinding.bindUIControl  
("Address1",myjButton));
```

To change the data binding of an LOV edit button in a form or panel:

1. Open the form or panel containing the `JULovEditButton` control in the Java visual editor.
2. Right-click the `JULovEditButton` control you want to change and select **Edit Bindings** from the menu.

The Create Button LOV Binding dialog displays.

7.8 How to Use the JUNavigationBar Control

The navigation bar is an ADF Swing `JUNavigationBar` control that can be bound to:

- A business service data collection through the model object
- A `JUPanel` that has a panel binding passed to it through its constructor

The effect of these bindings determines the scope of the navigation bar actions. If you bind to a data collection, the `JUNavigationBar` lets you navigate on those controls that share the same panel and that are bound to the same data collection through its attributes. Whereas, you can control the navigation of controls in child panels, by adding the `JUNavigationBar` control to the containing panel and setting the panel binding.

To bind to a data collection:

1. In the user interface project, open the Java visual editor on the data panel or form.
2. In the Data Controls panel, drag the data collection you want to bind to the navigation bar control into the open form or panel.
3. From the **Add Child** popup list, select **JUNavigationBar**.

JDeveloper adds code to the class file to bind the `JUNavigationBar` to the collection:

```
jUNavigationBar1.setModel(JUNavigationBar.createViewBinding  
(panelBinding, jUNavigationBar1, "MyDataCollection", null,  
"MyCollectionIterator"));
```

To bind to a JUPanel:

1. In the user interface project, open the Java visual editor on the layout panel that contains the panels you want to navigate on.
2. In the **ADF Swing Controls** page of the Components window, drag the **JUNavigationBar** control into the open form.
3. In the Properties window for the **JUNavigationBar** control, select the **model** field and select **ADF Swing panel binding**.

JDeveloper adds code to the class file to bind the `JUNavigationBar` to the panel binding passed into the current panel:

```
myNavBar.setModel(JUNavigationBar.createPanelBinding (panelBinding,
myNavBar));
```

7.9 How to Use the JUNavigationBar Control with Find Mode

In ADF Swing, you can support parameterized queries by using the `JUNavigationBar` control in an ADF Swing data panel with the Find mode enabled. Find mode lets users use the data panel to enter search criteria through supported ADF Swing control bindings.

When you run the panel, in Find mode, the navigation-mode component no longer forces the search criteria components to update. They will remain empty in order to permit the user to entry search criteria.

A typical usage for a data panel with Find mode is:

1. The user places the panel in Find mode.
For instance, a user may click the Find button, which is provided by default with a `JUNavigationBar` (that is bound to an ADF Business Components view object).
2. The user enters find criteria to restrict the results of the data already returned to the form.
3. The Find mode performs an anchored, wild card search.
It uses the first character of the search column as an anchor, where all the strings that begin with the entered string are matched.
4. Another control that you bind to the same view object in ADF Swing, such as the `JTable` control, displays the results of the parameterized query.

A parameterized query is a query that contains a placeholder that must be supplied at runtime. For example, in the following PL/SQL statement, `min_salary` is a placeholder for a parameter value that will be supplied at runtime.

```
SELECT ename, job, mgr FROM emp WHERE sal < :min_salary
```

The data panel in Find mode uses the ADF Swing control bindings to display fields for each attribute in the bound ADF Business Components view object whose `queriable` property is set to `true`. The view object defines the initial query executed by the business components.

The easiest way to create the data panel for Find mode operations is to bind a `JUNavigationBar` to the ADF Business Components view object that you want the data panel to search on. The navigation bar provides a Find button that the user selects to toggle the behavior of the ADF Swing control bindings for the ADF Swing form:

- When Find mode is enabled on the data panel, the controls provide search criteria for a parameterized query on the queryable attributes of the ADF Business Components view object.
- When Find mode is disabled, the controls provide data access to display and edit attributes of the view object.

The state of the Find-mode button is controlled by the `hasFindButton` property on the `JUNavigationBar` (true by default).

7.10 How to Disable Find Mode for ADF Swing Controls in a Panel

Note that all of the ADF Swing control bindings support the Find mode. If you want to create a data panel specifically for searches, limit your form to use these controls bound to ADF Business Components view object attributes to provide search criteria:

- `JTextField` -- the user must enter a value
- `JCheckBox` -- the user selects one of the predefined values
- `JSlider` -- the user selects one of the predefined values

It is recommended that your data panel disable controls not in the above list when the user selects the Find mode. Disabling controls that do not support Find mode prevents the user from attempting to submit invalid search criteria.

Additionally, UI components that utilize an ADF list binding in navigation mode do not participate in the Find mode. However, if you drop a navigation-mode component, such as the databound `ComboBox` or `JList`, into a panel and activate Find mode, the component will not be disabled and will still permit the user to unintentionally display data in components that should appear empty in Find mode. To disable the interaction of the navigation component and the other components in Find mode, you can create a secondary iterator and edit the navigation-mode component's data binding to use that iterator.

To disable the interaction of the navigation component and the other components in Find mode:

1. In the user interface project, open the Java visual editor on the layout panel that contains the panels you want to navigate on.
2. In Java visual editor, click the **Bindings** tab and in the **Executables** list, click **Create executable binding**.
3. In the Insert Item dialog, select **Iterator** and click **OK**.
4. In the Create Iterator Binding dialog, select the data collection that your other components are bound to and enter a new name for the **Iterator Id** and click **OK**.
5. In the **Bindings** list, double-click the binding for the component that operates in navigation mode and select the new iterator in the control binding editor.

7.11 What You May Need to Know About Iterator Bindings in Find Mode

In addition to its basic data iteration functionality, the iterator binding also cooperates with the bound data collection to simplify implementing query-by-example capability for your application data by providing:

- A collection of query-by-example criteria rows, and
- An easy-to-use Find mode to work with these query criteria

In Oracle ADF, each data collection has an associated view criteria (`ViewCriteria`) collection of zero or more view criteria rows. Each view criteria row (`ViewCriteriaRow`) has the same attribute structure as a row in its related data collection, except that the attribute values are all treated as a `String` data type. This data type allows the user to enter query criteria containing comparison operators and wildcard characters.

For example, to indicate you want to find all departments whose department number is greater than 5 and whose department name matches the string 'ACC%', you would fill in the attributes of a view criteria row related to a `DeptView` collection (based on a query over the familiar DEPT table in the SCOTT schema) like this:

- Deptno > 5
- Dname ACC%

The iterator binding's Find mode makes it easy to create search pages that populate the attributes of the view criteria collection for query-by-example functionality. When an iterator binding is set to work in Find mode, it switches to use a different row set iterator over the related view criteria collection instead. This means that when Find mode is enabled, control bindings that reference the iterator binding will display and update attributes in the current view criteria row. Likewise, a range binding that references an iterator binding in Find mode allows you to render a table of current query-by-example view criteria rows.

When Find mode is disabled, the iterator binding switches back to work with its row set iterator over the data collection. This can be done explicitly by calling the iterator binding's `setFindMode()` method, or implicitly by calling its `executeQuery()` method.

Note: By calling the `createRow()` method on the iterator binding's row set iterator while in Find mode, it is possible to create additional view criteria rows and then proceed to populate their attributes with additional criteria. The default semantics are that query-by-example criteria in the same view criteria rows are logically AND-ed together, while criteria resulting from separate view criteria rows are logically OR-ed together.

In actual practice, using multiple view criteria rows is not a common use case, but knowing it is possible helps to explain functionality. However, after the user enters Find mode, it is no longer possible to create the row on the original collection. Only when the user exits Find mode will it be possible to create a row that does not participate in the view criteria.

Although the ADF iterator binding provides its Find mode functionality independent of the kind of backend data control you choose, currently only the ADF Business Components data control makes automatic use of the view criteria collection of view criteria rows at runtime. It delegates the iterator binding's view criteria functionality to the underlying ADF view object, which implements the query-by-example criteria by automatically building appropriate SQL WHERE clause predicates based on the view criteria rows.

Other data control types would currently require a subclassed data control implementation containing some custom coding to read the query-by-example criteria from the view criteria collection and translate them into an appropriate runtime search implementation.

For details about the usage of view criteria in ADF Business Components, see the JavaDoc for the `oracle.jbo.ViewCriteria` class.

7.12 How to Use the JURadioButtonGroupPanel Control

The radio button group is an ADF Swing `JURadioButtonGroupPanel` control that can be bound to an attribute group of a business service data collection through the ADF navigate binding. When the user changes the radio button selection, any controls in the containing panel that are also bound to the same data collection through its attributes will display from the selected data object.

To bind a radio button group to navigate data collections:

1. In the user interface project, open the Java visual editor on the desired data panel or form.
2. In the Data Controls panel, drag the data collection you want to bind to the radio button group control into the open form or panel.
3. From the **Add Child** popup list, select **Radio Button Group**.

JDeveloper adds code to the class file to bind the `JURadioButtonGroupPanel` to the collection:

```
jURadioButtonGroupPanel1.setModel((JUIButtonGroupBinding)
panelBinding.bindUIControl("MyDataCollection",
jURadioButtonGroupPanel1));
```

4. In the List binding editor, choose the display attributes you want to use with the radio button panel.

To change the data binding of an radio button group:

1. Open the form or panel containing the `JURadioButtonGroupPanel` control in the Java visual editor.
2. Right-click the `JURadioButtonGroupPanel` control you want to change and select **Edit Bindings** from the menu.

The Edit List Binding dialog displays.

To customize the layout of the radio button panel:

The default layout for the radio buttons is by row, but you can change the layout through the `rowCount` and `columnCount` properties of the panel.

For example, to display the radio button selections in columns:

- Set `rowCount` to 0, and `columnCount` to the desired number of columns.

7.13 How to Use the JUShuttlePanel Control

The Shuttle control is an ADF Swing `JUShuttlePanel` control that can be bound to an attribute of a business service collection through the ADF Swing list binding. The `JUShuttlePanel` control allows the user to make multiple selections for the displayed shuttle list.

To insert a databound shuttle panel into a form or panel:

1. In the user interface project, open the form or data panel in the Java visual editor.

2. In the Data Controls panel, drag the collection you want to bind to the `JUShuttlePanel` control into the open form or panel.
3. From the **Add Child** popup list, select **Shuttle**.

JDeveloper adds code to the class file to reference the `DCShuttle` binding definition in the page definition file associated with the form:

```
myJUShuttlePanel1.setModel((JUShuttleModel)
(panelBinding.findNestedPanelBinding("DCShuttle")));
```

To change the data binding of a shuttle panel in a form or panel:

1. Open the form or panel containing the `JUShuttlePanel` control in the Java visual editor.
2. Right-click the `JUShuttlePanel` control you want to change and select **Edit Bindings** from the menu.

The Edit Shuttle Binding dialog displays.

7.14 How to Use the JUStatusBar Control

The status bar is an ADF Swing `JUStatusBar` control that relies on a panel binding to display status information for the UI control that shares its panel binding and has the current focus in the displayed form or panel.

To insert a status bar into a form or panel:

1. In the user interface project, open the form or data panel in the Java visual editor.
2. In the Data Controls panel, drag the collection you want to bind to the `JUStatusBar` control into the open form or panel.
3. From the **Add Child** popup list, select **Status Bar**.

JDeveloper adds code to the class file to define a panel binding for the status bar:

```
myJUStatusBar1.setModel(JUStatusBar.createPanelBinding(panelBinding,
JUStatusBar1));
```

Using Validation in the ADF Swing User Interface

This chapter describes how to use client-side validation in the ADF Swing application.

This chapter contains the following sections:

- [Section 8.1, "About Validating Events"](#)
- [Section 8.2, "How to Use Validation With ADF Control Bindings"](#)
- [Section 8.3, "How to Use Validation With ADF Swing Panels"](#)

8.1 About Validating Events

JDeveloper performs validation using:

- Rules defined in the ADF Business Components data model project
- Upon activity in the user interface

Through the ADF Swing application you perform validation on the ADF Swing form to prevent users from entering erroneous data. For example, you may want to prevent alpha characters from being input to a field which requires a telephone number. Similarly you may want to validate the data entered and inform the user.

8.2 How to Use Validation With ADF Control Bindings

Validation in the user interface is handled through source code at the level of the control binding. To do this, you define your validation code as a `PlainDocument` event and then use the `setDocument()` method to register the code with the specific control. Then the call to `setDocument()` on the control you bound to the ADF Business Components attribute item will work with the document which contains the validation code. [Example 8-1](#) illustrate validation that occurs at each key press.

Example 8-1 Validation at Every KeyStroke Example

```
//Add this code to create a document with the validation code and set it for the control
```

```
mDeptno.setDocument(new javax.swing.text.PlainDocument()  
{  
    public void insertString(int offs, String str, javax.swing.text.AttributeSet a)  
        throws javax.swing.text.BadLocationException  
    {  
        StringBuffer buf = new StringBuffer(str);  
        int size = buf.length();
```

```

char c;
for (int i = 0; i < size; i++)
{
    c = buf.charAt(i);
    if (!Character.isDigit(c))
    {
        Toolkit.getDefaultToolkit().beep();
        buf.deleteCharAt(i);
    }
}

super.insertString(off, buf.toString(), a);
}

});

/*Here is the code that is generated for you which will set the control binding
for the document. This will work with the document you defined above. */

mDepartmentId.setDocument((Document)panelBinding.bindUIControl("DepartmentId",
mDepartmentId));

```

8.3 How to Use Validation With ADF Swing Panels

The `panelBinding` object, which is responsible for managing the link between the UI and the ADF Business Components data model layer, can trigger an event on an action. This type of validation allows you to perform validation when the user inputs data into a field and then navigates to a different field. You may want to perform this type of validation at the panel which should notify the user, without breaking a business rule as defined in the business components.

For example, consider a salary which has a range of 100 up to 1000 defined in the business components. If the salary field is set to anything less than 500, you want to alert the user and highlight the field. In this case, you want to validate when the value is being set. The `panelBinding` object will trigger an event when `beforeSetAttribute()` is called.

To set validation on the ADF Swing panel:

1. Open the panel in the Java visual editor.
2. In the Structure window, expand **Other** and then select **panelBinding**.
3. In the Properties window, expand the **Events** section and enter an event name in the **beforeSetAttribute** field.
4. In the Java source editor, add the validation code to the generated event handler, similar to the sample shown in [Example 8-2](#).

Example 8-2 Code Added to Generated Event Handler

```

if (e.getAttributeName().equals("Salary"))
{
    Object val = e.getNewValue();
    if (val != null)
    {
        Integer n = new Integer(val.toString());
        if (n.intValue() < 500)
        {
            mSalary.setBackground(Color.red);
            throw new oracle.jbo.JboException("That's a bit low!");
        }
    }
}

```

```
}  
}  
}
```

Working with an ADF Swing Login Dialog

This chapter describes how to create a login dialog and run the ADF Swing application using the login dialog. The ADF Swing login dialog is optimized to work with the ADF Business Components project.

This chapter includes the following sections:

- [Section 9.1, "About the ADF Swing Login Dialog"](#)
- [Section 9.2, "How to Create a Login Dialog"](#)
- [Section 9.3, "How to Run the Application Using the Login Dialog"](#)
- [Section 9.4, "How to Run the Application Without the Login Dialog"](#)
- [Section 9.5, "What You May Need to Know About Customizing the Login Dialog Code"](#)
- [Section 9.6, "How to Modify the Login Dialog to Work with a JDBC Connection"](#)

9.1 About the ADF Swing Login Dialog

You can add a customizable login dialog to your user interface project that will require a user name and password to run your ADF Swing application that you have created for an ADF Business Components data model. Currently, the login dialog is not supported for other business services.

By default the generated login dialog works with Oracle WebLogic Server container-enforced authentication and uses `SECURITY_PRINCIPAL` and `SECURITY_CREDENTIALS` properties. However, you can customize the generated file when you prefer to bypass these security provider properties and work instead with a JDBC connection that requires `DB_USERNAME_PROPERTY` and `DB_PASSWORD_PROPERTY` properties.

You add the login dialog class to your user interface project when you generate a frame to run your ADF Swing application. You can use two wizards in ADF Swing to generate a `JFrame` with the appropriate ADF Swing bootstrap code:

- Create ADF Swing Form wizard - click **Generate a Login Dialog** in the File Names page
- Create ADF Swing Empty Form dialog - click **Generate a Login Dialog**

When you run either of these with the generate login dialog option selected, JDeveloper:

- Modifies the application object constructor in the application bootstrap code to create an instance of `JLoginDialog`:

```
BindingContext ctx = new BindingContext();  
ctx.put(DataControlFactory.APP_PARAM_ENV_INFO, new JCLoginDialog());
```

- Adds the `JCLoginDialog.java` file to your user interface project, which implements the `EnvInfoProvider` interface to provide the runtime login dialog. The wizard will not overwrite an existing `JCLoginDialog.java` file of the same name.

Because the generated login dialog (`JCLoginDialog.java`) implements the methods of the interface, it gives you the starting code that you can modify. Using the `JCLoginDialog.java` file, you can customize any aspect of the login dialog:

- Its visual appearance by adding images and changing the layout
- Configuration parameters to connect to the ADF Business Components application module
- Connection parameters to connect to the database
- Specification of the number of times to retry connecting after failing to connect

9.2 How to Create a Login Dialog

You can use the Create ADF Swing Form wizard or the Create ADF Swing Empty Form dialog to add a generated login dialog to your ADF Swing application. The generated login dialog relies on the Oracle ADF security framework to authenticate principal and credentials and therefore does not work with a JDBC connection.

To add the ADF Swing login dialog to your project:

1. In the Applications window, select the user interface project and launch the Create ADF Swing Form wizard or the Create ADF Swing Empty Form dialog.
2. On the last page of the form wizard, select **Generate a Login Dialog**.
3. Click **Finish** to complete the wizard.

9.3 How to Run the Application Using the Login Dialog

To use the login dialog with Oracle WebLogic Server and ADF Security authentication, you must run the Configure ADF Security wizard to configure the ADF Swing application to use ADF security.

To configure the ADF Swing client to use ADF authentication:

1. In the Applications window, select the user interface project for which ADF authentication is needed and choose **Application > Secure > Configure ADF Security**.
2. In the ADF Security wizard, select **ADF Authentication and Authorization**.
3. Click **Finish**.

To enable the login provider for ADF Swing forms:

1. In the Applications window, right-click the user interface project and choose **Project Properties**.
2. In the Project Properties dialog, click **Libraries and Classpath** to display the current list of libraries associated with your user interface project.
3. Click **Add Library**.

4. In the list of libraries, locate **BC4J Security** and add it to the list.
Otherwise an error message will display when using the logon dialog. The BC4J Security library contains the JAR files required to use ADF authentication within the project.
5. Click **OK** to save the project settings.
6. If your user interface project requires access to ADF Business Components deployed as an EJB session bean, you must grant read/write access to the users in your group.

User accounts are stored in the `jazn-data.xml` file located in `./src/META-INF` directory relative to the ADF Swing application. One of the default names is `admin/welcome`. To test your logon dialog, use one of the default names. Alternatively, you can add your own user to `jazn-data.xml`.

The `jazn-data.xml` file encrypts the password the first time the server instance gets started after an account is added. To make sure a password is encrypted, add an `!` in front of the password. For example, to encrypt the password `WELCOME`, define it as `!WELCOME`.

To work with the username/credential pair for your users, use the code in [Example 9-1](#) in your application.

Example 9-1 Code for Username/Credential Pairs

```
Hashtable h = panelBinding.getBindingContext().getDefaultDataControl().
getApplicationModule().getSession().getEnvironment();

String username = h.get(JboContext.SECURITY_PRINCIPAL);
String credential = h.get(JboContext.SECURITY_CREDENTIALS);
...
```

9.4 How to Run the Application Without the Login Dialog

When you run one of the ADF Swing wizards to generate an ADF Swing frame, but do not generate a login dialog in your ADF Swing user interface project (by leaving **Generate a Login Dialog** in the wizard unselected), you can still run the application by selecting **Save Password** in the Edit Database Connection dialog. In this case, when you run the application, JDeveloper displays the frame without prompting you for login.

9.5 What You May Need to Know About Customizing the Login Dialog Code

You generate the `JLoginDialog.java` file in your user interface project when you want to modify the starter code that implements the required methods of the `EnvInfoProvider` interface. The interface is used in the ADF Business Components connection strategy to provide the hooks to change login parameters at runtime.

The `EnvInfoProvider` interface expects the following methods to be implemented:

```
public void getInfo(String info, Object connEnvironment)
```

This method is called before connecting to the database. It allows you to update (and return) the hashtable with all the connection parameters.

```
Public int getNumOfRetries()
```

This method determines how many times the business components will attempt to connect after failing. Each time it will obtain connection information from the `EnvInfoProvider`.

Note: The method `public void modifyInitialContext(Object initialContext)` has been used in previous releases, but is now deprecated. You may implement it as an empty method.

9.6 How to Modify the Login Dialog to Work with a JDBC Connection

You can modify the generated ADF Swing login dialog to work with a JDBC connection instead of Oracle ADF authentication.

To modify the generated ADF Swing login dialog for JDBC connections:

1. Add the login dialog to your user interface project using the Create ADF Swing Form wizard or the Create ADF Swing Empty Form dialog.

For more information, see [Section 9.2, "How to Create a Login Dialog."](#)

2. In the user interface project, double-click the `JCLoginDialog.java` file.
3. In the Java source editor, comment out the code as shown in [Example 9-2](#).

Example 9-2 Commented Code in `JCLoginDialog.java`

```

/*
    String securityEnforceStr =
        ((String) ((Hashtable) connEnvironment).get(PropertyMetadata.ENV_SECURITY_
ENFORCE .pName));
        if (securityEnforceStr == null
            (securityEnforceStr != null && PropertyConstants.SECURITY_ENFORCE_
NONE.equals(securityEnforceStr)))
        {
            return null;
        }
*/

```

4. Press **Ctrl+F** to display the Find dialog in the source editor.
5. Perform this search and replace operation:

Change `JboContext.SECURITY_PRINCIPAL` to `Configuration.DB_USERNAME_PROPERTY`
6. Display the Find dialog again and perform another search and replace operation:

Change `JboContext.SECURITY_CREDENTIALS` to `Configuration.DB_PASSWORD_PROPERTY`

Note: These changes appear twice within the file: once when reading from the hash table and once when setting user name and password from the fields in the login dialog.

7. Save the changes to `JCLoginDialog.java`.
8. If you saved the password when you created your data model project (the default), you need to change these settings. In the Databases window, right-click the connection and choose **Properties**.

9. In the Edit Database Connection dialog, delete the password and deselect **Save Password**.

You are now ready to run your ADF Swing application. If you have a JDBC connection that the `bc4j.xcfg` file names, the application will invoke the dialog automatically.

Optimizing ADF Swing Application Runtime Performance

This chapter describes how to improve ADF Swing application runtime performance. It describes data synchronization techniques for optimum performance as well as a technique to limit fetching of ADF Business Components attributes in ADF Swing applications.

This chapter contains the following sections:

- [Section 10.1, "About Optimizing ADF Swing Application Runtime Performance"](#)
- [Section 10.2, "How to Delay Updates to ADF Business Components from ADF Swing"](#)
- [Section 10.3, "What You May Need to Know About the Sync Mode Property"](#)
- [Section 10.4, "How to Limit Fetching of ADF Business Components Attributes in ADF Swing"](#)

10.1 About Optimizing ADF Swing Application Runtime Performance

You can make method calls in your ADF Swing code to optimize the network traffic between the client and the ADF Business Components business services in remote deployment mode.

10.2 How to Delay Updates to ADF Business Components from ADF Swing

Users who work with the design time tools in JDeveloper will by default obtain the optimization described in this section.

However, you may need to perform these optimizations if you develop applications outside of JDeveloper using the provided APIs for ADF Business Components and do not use the JDeveloper design time to create the ADF Swing client.

You can improve network traffic at the expense of less frequent validation submitted by an ADF Business Components client application to the remotely deployed business components by setting the data synchronization mode of the ADF Business Components application module to batch. Batch mode means attribute value changes made by the user will be held on the client tier until the user performs an action that forces the changes to be submitted. Also known as lazy sync mode, batch mode will reduce the number of trips over the network, consequently validation of the attributes by ADF Business Components will be delayed.

The data synchronization is "immediate" mode by default which applies changes as soon as they are made and results in immediate validation of attribute values.

Note: In local mode deployment (the client and ADF Business Components reside in the same VM), sync mode is always "immediate".

To change a client's synchronization setting within JDeveloper:

1. In the Applications window, change the display mode to **Directory View**, then expand the user interface project and select the `DataBindings.cpx` file.

The `DataBindings.cpx` file is added to the `adfmsrc` folder of your user interface project.

2. In the Structure window, expand **dataControlUsages** and select the desired data control.
3. In the Properties window, expand the **Other** section and select the desired setting from the **syncMode** field dropdown list.

To set the sync mode of an application module (outside of the JDeveloper design time):

- Call `setSyncMode()` on the `ApplicationModuleImpl` class:

```
yourAm.setSyncMode(ApplicationModule.SYNC_IMMEDIATE);
```

You can set it to `SYNC_IMMEDIATE` when you want to maximize control of attribute validation by ADF Business Components at the expense of network optimization. For example, in this mode, if the client sets ten attributes of an employee, your code would call ten `setAttribute()` methods in order to provide validation for each new value.

Or

```
yourAm.setSyncMode(ApplicationModule.SYNC_LAZY);
```

You can set it to `SYNC_LAZY` to optimize your application by reducing the number of trips over the network. In this mode, the client-side attribute set requests are buffered until the next time the client sends an event to ADF Business Components, instead of when the data is changed. No further changes to your application code are required. For example, if the client sets ten attributes of an employee, your code would call the `setAttribute()` methods only after the client:

- Changes the current row
- Calls the `postChanges()` or `commit()` methods
- Calls the `sync` method explicitly
- Calls the `validate()` method on the row

If you are inside a class that extends `ApplicationModuleImpl`, because it implements the `ApplicationModule` interface, you can just add the code shown in [Example 10-1](#).

Example 10-1 Code When Class Extends `ApplicationModuleImpl`

```
ApplicationModule yourAm = panelBinding.getApplication().getApplicationModule();
yourAm.setSyncMode(SYNC_LAZY);
```

However, if you are in another class (for example, in the client code), then you need to qualify the constant with the interface on which it appears, as shown in [Example 10–2](#).

Example 10–2 Code When In Another Class

```
ApplicationModule yourAm = panelBinding.getApplication().getApplicationModule();
yourAm.setSyncMode(ApplicationModule.SYNC_LAZY);
```

You should call `setSyncMode()` in the ADF Swing bootstrap code, immediately after the data control object is created. Or, you can call `getApplication()` and set the sync mode in the constructor for your ADF Swing panel. The location you choose depends upon whether you want the sync mode to persist for the entire application or to be modified at the level of the panel.

10.3 What You May Need to Know About the Sync Mode Property

If you decide to set data synchronization programmatically on the `ApplicationModule`, you should be aware of an interaction that will occur with the `syncMode` property setting on the ADF Business Components data control. Only the following three valid combinations exist.

- `am.setSyncMode(SYNC_IMMEDIATE)` + `syncMode` property "Batch" -- yields Batch mode. All updates will be delayed until next `am.sync()`, which needs to be called by the application at an appropriate time by calling `bindingContainer.refresh()`. Alternatively, updates will occur automatically when Rollback and Commit actions are initiated.

The `SYNC_IMMEDIATE` + "Batch" combination is the default in JDeveloper 10.1.2 and later when you create an ADF BC project. This is the combination to use for 3-tier ADF Swing applications in JDeveloper 10.1.2 and later.

- `am.setSyncMode(SYNC_LAZY)` + `syncMode` property "Immediate" -- yields Immediate mode and set attribute method calls will be delayed until the next row navigation.

The `SYNC_LAZY` + "Immediate" combination is the setting for ADF Swing 9.0.5 and earlier applications that are deployed in 3-tier mode and that you upgrade to 10.1.2 and later.

- `am.setSyncMode(SYNC_IMMEDIATE)` + `syncMode` property "Immediate" -- yields Immediate mode.

The `SYNC_IMMEDIATE` + "Immediate" combination is appropriate for applications that have a very high level of interactivity with the database and/or where immediate validation is required. In this case the application should be run in immediate mode so that data is synchronized immediately. It is not recommended for ADF Swing applications.

10.4 How to Limit Fetching of ADF Business Components Attributes in ADF Swing

Users who work with the design time tools in JDeveloper will by default obtain the optimization described in this section.

However, you may need to perform these optimizations if you develop applications outside of JDeveloper using the provided APIs for ADF Business Components and do not create view object metadata (XML files).

You can optimize startup time for an ADF Business Components client application and the remotely deployed business components by specifying the list of view object attributes that your client uses. If you create a project without the metadata, by coding to the API, you will want to add `fetchAttributeProperties()` to the bootstrap code of the client forms with a list of only the attributes used by the form. Without this method call, your client form would fetch all control hint properties (including the attributes format and label for example) for all the attributes of the named view objects in the application module, in a single network roundtrip.

For example, when you do not intend to use all the attributes of the ADF Swing form's bound view object, with the `fetchAttributeProperties()` method, your ADF Swing form fetches only the information required to layout your forms, while ignoring the attributes you do not require.

Note: In local mode deployment (the client and ADF Business Components reside in the same VM), the fetching of attribute properties is not supported.

To minimize retrieving of attribute properties (outside of the JDeveloper design time):

- Call `fetchAttributeProperties()` on the `ApplicationModule`:

The method takes as arguments a list of view object names and a list of attribute names for each view object. The lists may include all or some of the attributes. If your forms require all the attributes of a view object, you may specify `null` as the attribute argument. In this case, the startup time improvement may not be significant since your form uses all the attributes of the view object anyway.

Note: When you use the ADF Swing Form or Panel wizards to generate complete forms, the `fetchAttributeProperties()` method is added for you.

You should call `fetchAttributeProperties()` in the ADF Swing bootstrap code, as shown in [Example 10-3](#), immediately after the data control object is created. In the following example, the custom properties for three attributes of the first view object and five attributes of the second view object are downloaded.

Example 10-3 Calling `fetchAttributeProperties` in ADF Swing Bootstrap Code

```
// bootstrap application
JUMetaObjectManager.setBaseErrorHandler(new JUErrHandlerDlg());
JUMetaObjectManager mgr = JUMetaObjectManager.getJUMom();
mgr.setADF SwingDefFactory(null);
BindingContext ctx = new BindingContext();
ctx.put(DataControlFactory.APP_PARAM_ENV_INFO, new JUEnvInfoProvider());
ctx.setLocaleContext(new DefLocaleContext(null));
HashMap map = new HashMap(4);
map.put(DataControlFactory.APP_PARAMS_BINDING_CONTEXT, ctx);
mgr.loadCpx("DataBindings.cpx", map);
DCDataControl app = (DCDataControl)ctx.get("model_AppModuleDataControl");
app.setClientApp(DCDataControl.JCLIENT);
app.getApplicationModule().fetchAttributeProperties(new String[]
    {"V01", "V02"}, new String[][]
    {
        {"V01Attr1", "V01Attr2", "V01Attr3"},
```

```
    {"VO2Attr1", "VO2Attr2", "VO2Attr3", "VO2Attr4", "VO2Attr5"}  
  }, null);
```

...

Calling `fetchAttributeProperties()` prevents property methods such as `getFormat()` or `getLabel()` from being called on the ADF Business Components attribute definition whenever the form is created.

Using Java Web Start With ADF Swing Applications

This chapter describes how to run ADF Swing applications with Java Web Start in JDeveloper. Java Web Start is a tool for deploying the ADF Swing application to a web server from which users can download and run the application on their client machines.

This chapter contains the following sections:

- Section 11.1, "About Working with Java Web Start"
- Section 11.2, "How to Define ADF Business Components Runtime Properties"
- Section 11.3, "How to Set Up Runtime Configuration Information"
- Section 11.4, "How to Create a Java Web Start JNLP Definition"
- Section 11.5, "What Happens When You Create a JNLP Definition"
- Section 11.6, "How to Run ADF Swing Applications with Java Web Start in JDeveloper"

11.1 About Working with Java Web Start

Java Web Start is an application deployment software that lets you maintain ADF Swing applications on the web server. Once deployed, users can use Java Web Start to download the application to run on their client machines. Before you deploy your ADF Swing application or applet to the web server, you can simulate the user's experience of running the application with Java Web Start within JDeveloper and Integrated WebLogic Server. Then you can use JDeveloper's Java EE web deployment process to move the entire production application to the web server.

Java Web Start is an application deployment technology. JDeveloper supports the creation of the XML-based JNLP (Java Network Launching Protocol) definition upon which the Java Web Start technology is based. With Java Web Start and the Create Java Web Start File dialog in JDeveloper, you can set up ADF Swing applications and applets to be maintained on the web server, but to be downloaded and run on client machines.

To launch ADF Swing applications with Java Web Start in JDeveloper, you must download and install the Java Web Start software at this web site <http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136112.html>. Users of your ADF Swing application will also be required to install the software on their machines.

11.1.1 Java Web Start Technology

Unlike the applet approach to deploying web-centric Java applications, Java Web Start does not rely on the web browser to perform the downloading of the application JAR files. Instead, Java Web Start downloads the application resources after the Java Web Start JNLP descriptor is downloaded through the web browser. The JNLP descriptor causes Java Web Start to launch and perform the actual downloading.

The technologies involved are:

- Web server -- stores the JNLP-enabled JSP files, JAR files for client, middle tier, and runtime libraries.
- Application server (optional) -- needed only if the middle tier runs remotely with respect to client. For example, in the case of EJB session beans.
- Web browser (optional) -- downloads JSP page, which launches Java Web Start. Java Web Start then downloads the dependent resources and launches the application or applet. Java Web Start can also run independently without the browser, using a command like:

```
javaws http://localhost/myapp/test.jnlp
```

- Web browser --- optional, but convenient to launch Java Web Start and initiate the download. Used when an HTML file contains a link to the JNLP-enabled JSP page. The user follows the link, which behind the scenes causes Java Web Start to run.

With the Java Web Start software installed once on the user's machine, individual users can run ADF Swing applications and applets simply by clicking on a web page link (the appropriate files are generated by the ADF Swing dialog). Once the application is running, the web browser can be closed, and the application continues to function.

If the application is not present on their computer, Java Web Start automatically downloads all necessary files from the web server where the application libraries reside. It then caches the files on the client computer so the application is always ready to be relaunched anytime either from an icon on your desktop or from the browser link. The most current version of the application is always presented to the user since Java Web Start performs updates as needed.

11.1.2 Java Web Start and Integrated WebLogic Server

JDeveloper provides Integrated WebLogic Server. You can use it to simulate the process of deploying the Web Application Archive and downloading for use with Java Web Start. JDeveloper follows the Java EE deployment profile conventions for archiving components that run on the client machine (simple archive) and components that are deployed to the web server (Web Application Archive).

After you complete the Create Java Web Start File dialog, you need to run the Ant build file `ctbuild.xml` to complete the JDeveloper setup and sign the JAR files.

Note: You will not be required to deploy the generated `client_war.deploy` profile to use Integrated WebLogic Server. JDeveloper provides a default `web.xml` definition to locate the contents of the `public_html` directory in your JDeveloper `mywork` folder.

Once you have signed your JAR files, you can launch the Java Web Start software in JDeveloper using the generated `.jsp` file. Java Web Start downloads the components identified by the `.jnlp` file. Another definition in the `.jnlp` file determines whether the ADF Swing is to run as an application or a secure applet. Once you have launched

Java Web Start and the downloading is complete, you can close your web browser and continue to run the application or applet.

11.1.3 Java Web Start and Oracle WebLogic Server

When you are ready to deploy to the Oracle WebLogic Server, you must set up the context roots for the various runtime libraries required by the ADF Swing application using the previously generated `ctbuild.xml` archive file. These libraries are required to run the application and they have to be accessible through HTTP. (One context root, for example, is `bc4j` and this enables the `bc4jmt.jar` file to be downloaded as `http://myachine:8888/bc4j/lib/bc4jmt.jar`).

To deploy an ADF Swing application to the Oracle WebLogic Server you use the generated `client_war.deploy` file to set up classes.

11.2 How to Define ADF Business Components Runtime Properties

When you run ADF Swing forms to access the model layer, your data model project for ADF Business Components must contain a runtime configuration (`bc4j.xcfg`) file that specifies the application module connection for your deployment scenario.

When you want to run your ADF Swing application within JDeveloper, the application will use the default local configuration.

Later, you can edit the runtime configuration to update the connection information when you change or create a new ADF Business Components deployment scenario.

Note: If you do not create an application module runtime configuration, the `bc4j.xcfg` file in your data model project defines a default configuration that lets you run ADF Swing forms within JDeveloper. The default configuration `local` is also called *local mode deployment* because it assumes the business components and the ADF Swing application run in the same VM.

To create a configuration:

1. In the Applications window, expand the **model** package in the **Application Sources** folder for the data model project and double-click the application module node.
2. In the overview editor, click the **Configurations** navigation tab and click **Create new configuration object**.
3. In the Edit Configuration dialog, click the **Application Module** tab and choose the middle-tier server type and a previously defined connection.

Click **Help** for further details.

To edit a configuration:

1. In the Applications window, expand the **model** package in the **Application Sources** folder for the data model project and double-click the application module node.
2. In the overview editor, click the **Configurations** navigation tab and then select the configuration to edit from the list and click **Edit**.
3. In the Edit Configuration dialog, click the **Application Module** tab and choose the middle-tier server type and a previously defined connection.

Click **Help** for further details.

4. If you changed the deployment platform specified by the configuration file by choosing a different **Middle Tier Server Type** option in the Edit Configuration dialog, then you must update the data model project to add the libraries for the new platform:
 - Create a deployment archive for your data model project.
 - In the Applications window, expand the deployment archive folder.
 - To update the libraries, choose **Deploy to *projectname.jar*** on the **Common** and **Middle Tier** archives.

Note: Configurations are accessed through the ADF data control for the application module, but the configuration information is recorded in the `bc4j.xcfg` file. This file is hidden in the Applications window and visible by right-clicking the application module and choosing **Configurations**.

When you want to refer to the new configuration in your user interface project, you must edit the `DataBindings.cpx` file.

To reference a data model configuration in the ADF Swing client:

1. Locate the `DataBindings.cpx` node in the user interface project folder.
2. Select the **DataBindings.cpx** node and display the Structure window.
3. In the Structure window, expand the **dataControlUsages** node and select the **AppModuleDataControl** node.
4. In the Properties window, expand the **Other** section and select the desired configuration from Configuration field dropdown list.

11.3 How to Set Up Runtime Configuration Information

Before you use can run an ADF Swing application, you must set up runtime configuration information for your application. Your application needs these files to determine a runtime configuration:

- An ADF Business Components configuration file (`bc4j.xcfg`) in your data model project that specifies the application module connection and deployment scenario for the business components.
- A client data model definition file (`DataBindings.cpx`) in your user interface project that specifies the application module and ADF Business Components runtime configuration your ADF Swing forms will use.

There are two ways to use the runtime information:

- When you want to test your ADF Swing application in JDeveloper
- When you want to deploy your ADF Swing application for production

You will want to create a separate ADF Business Components configuration and client data model definition for each case.

To set up runtime configuration information for ADF Swing applications:

1. Create the ADF Business Components runtime configuration (in the `bc4j.xcfg` file) that specifies the application module connection and deployment scenario for the business components.

For more information, see [Section 11.2, "How to Define ADF Business Components Runtime Properties."](#)

2. Create an ADF Swing data model definition that specifies the application module and business component runtime configuration your ADF Swing forms will use.

For more information, see [Section 2.6, "How to Create a Client Data Model Definition."](#)

3. Run a ADF Swing wizard to generate ADF Swing forms or data panels.

4. In the Data Model page of the wizard, select the previously created client data model definition.

For more information, see [Section 2.1.1, "ADF Swing Design Time Wizards."](#)

Note: You can also click **New** to create a data model definition based on another application module and runtime configuration that you select.

11.4 How to Create a Java Web Start JNLP Definition

You use the Create Java Web Start File dialog to create a JSP page that the user conveniently runs from their web browser to dynamically generate the JNLP definition. Users launch the JNLP file to initiate downloading of the application to their client machine.

Note: Before you can use the Create Java Web Start File dialog, you must set up runtime configuration information for your application. During development, use the default `local` mode deployment configuration to avoid potential security conflicts that occur in the remote deployment scenario.

For more information, see [Section 11.3, "How to Set Up Runtime Configuration Information."](#)

To create the JNLP definition for your application or applet:

1. In the Applications window, right-click the ADF Swing user interface project for which you want to generate a JNLP definition and choose **New > From Gallery**.
2. In the New Gallery, expand **Client Tier**, select **ADF Swing**, and then **Java Web Start (JNLP) Files for ADF Swing**, and click **OK**.
3. In the Create Java Web Start File dialog, specify the properties of the JNLP file and click **OK**.

Note: If you are planning to run the ADF Swing application using EJB as the business service, then you must replace the `weblogic.jar` reference in the JNLP definition with `wlclient.jar` instead. Edit the JNLP definition in the files described below to make this substitution. If you attempt to run the ADF Swing application with the downloaded `weblogic.jar` file, you may see the `oracle.jbo.JboException: JBO-29000: STRINGMANAGER: Message file: 'oracle.jbo.CSMMessageBundle' not found.` exception.

Your ADF Swing user interface project contains:

- `webstart.jsp` -- dynamically generate the JNLP file that describes the client and ADF Business Components archive files and whether the ADF Swing is an applet or an application.
- `webstartmt.jsp` -- an extension to the first `.jsp` file. Dynamically generates the JNLP file that describes the ADF Business Components runtime libraries required for a specific deployment scenario.
- `ctbuild.xml` file -- an Ant-based makefile that helps you to create signed archive files for the client side and the ADF Business Components middle-tier classes.
- `client_war.deploy` file -- use to generate the WAR file deployment profile.
- `web.xml` file -- defines the deployment descriptors for the project files.
- `webstart.html` file -- use to launch your application using Java Web Start.

If you want to require authentication of JAR files for added security when deploying your application to Oracle WebLogic Server, open a command prompt window and create the certificate:

```
keytool -export -alias yourkeyname -file mykey.cert
```

where *yourkeyname* is the name of the key that you will use to sign the JAR.

For more information about the Key and Certificate Management Tool, see <http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html>.

You are now ready to create the Web Archive before running the ADF Swing application. For more information, see "How to Create an ADF Swing Web Archive for Java Web Start" in *Developing Applications with Oracle JDeveloper*.

11.5 What Happens When You Create a JNLP Definition

To support downloading of files from the web server through Java Web Start, the Create Java Web Start File dialog generates:

- `ctbuild.xml` Ant build file -- use the build file to generate these application resources:
 - `client.jar` -- contains the ADF Swing application source files
 - `mymt.jar` -- contains the ADF Business Components source files
- `client_war.deploy` profile -- use to deploy the contents of the `public_html` directory in your JDeveloper `mywork` folder. It therefore contains the `client.jar`, `mymt.jar`, and either a static JNLP file or JSP files.

Additionally, the JNLP definition is generated dynamically through a `.jsp` file:

- `webstart.jsp` -- lets you dynamically generate the JNLP file that describes the client and ADF Business Components archive files and whether the ADF Swing is an applet or an application.
- `webstartmt.jsp` -- an extension to the first `.jsp` file. Dynamically generates the JNLP file that describes the ADF Business Components runtime libraries required for a specific deployment scenario.

11.6 How to Run ADF Swing Applications with Java Web Start in JDeveloper

Before you deploy your ADF Swing application or applet to a production web server, you can use Integrated WebLogic Server in JDeveloper to simulate the user's experience of running the application with the Java Web Start software.

Note: You cannot debug a ADF Swing application in JDeveloper while running with Java Web Start.

You must create a deployment profile to set up Integrated WebLogic Server with your application libraries and JAR files. Java Web Start relies on the JNLP file that you generate to identify which files to download and how to launch the application. For more information, see [Section 11.4, "How to Create a Java Web Start JNLP Definition."](#)

Before you begin:

1. Set up runtime configuration information for the business components deployment scenario you choose.

For more information, see [Section 11.3, "How to Set Up Runtime Configuration Information."](#)

Note: Choose the default `local` runtime configuration when you just want to test the business components using local-mode deployment. This option uses the same VM to run the ADF Business Components and ADF Swing libraries.

2. Archive the application components using the `ctbuild.xml` Ant makefile generated by the ADF Swing Java Web Start wizard.

For more information, see "How to Create an ADF Swing Web Archive for Java Web Start" in *Developing Applications with Oracle JDeveloper*.

Note: The build file generates two signed archive files in your project's `public_html` directory: `client.jar` and `mymt.jar`.

To run the ADF Swing application using Java Web Start:

1. Install the Java Web Start software on your machine.
2. In your user interface project, right-click the `local.html` file and choose **Run local.html** to launch your web browser with this page.
3. Click the **Launch ADF Swing Project** link to run your application using the Java Web Start software.

4. You can close your browser once Java Web Start completes the download and launches the application. Java Web Start lets you run both applications and secure applets.

The next time you run the application, Java Web Start will download only those source files or libraries that have changed from the previous download.

Note: If you attempt to run the ADF Swing application and have created a JNLP definition for Oracle WebLogic Server deployment, using `weblogic.jar`, you may see the `oracle.jbo.JboException: JBO-29000: STRINGMANAGER: Message file: 'oracle.jbo.CSMessageBundle' not found.` exception. This exception occurs when running Java Web Start on Oracle WebLogic Server with EJB as the business service. You must replace the `weblogic.jar` reference in the JNLP definition with `wlclient.jar` instead. You can edit the JNLP definition to make this substitution.
