Oracle® Fusion Middleware

Mobile Developer's Guide for Oracle Application Development Framework

11*g* Release 2 (11.1.2.3.0)

E24475-01

October 2012

Documentation for Oracle Application Development Framework (Oracle ADF) developers that describes how to use Oracle JDeveloper to create mobile applications that run natively on devices.

ORACLE

Oracle Fusion Middleware Mobile Developer's Guide for Oracle Application Development Framework 11*g* Release 2 (11.1.2.3.0)

E24475-01

Primary Authors: Liza Rekadze, John Bassett

Contributing Author: Cindy Hall

Contents

6 Controlling the Display of Application Features

8 Creating ADF Mobile AMX User Interface

9 Using Bindings and Creating Data Controls

10 Using Web Services

11 Using the Local Database

Part V Advanced Topics

12 Implementing Application Features as Remote URLs

17 ADF Mobile Application Security

18 Testing and Debugging ADF Mobile Applications

Part VII Appendixes

A Troubleshooting

B Converting Preferences for Deployment

C ADF Mobile Application Usage

D Parsing XML

E ADF Mobile Sample Applications

Preface

Welcome to the *Mobile Developer's Guide for Oracle Application Development Framework*.

Audience

This document is intended for Oracle ADF developers tasked with developing applications that run on-device and are intended for users working almost exclusively in mobile environments.

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:

- *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Application Development Framework*
- *Oracle Fusion Middleware Mobile Browser Developer's Guide for Oracle Application Development Framework*

Conventions

The following text conventions are used in this document:

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

Part I

Understanding ADF Mobile

Describes ADF Mobile concepts, technology, and development approach.

Part I contains the following chapters:

- Chapter 1, "Introduction to ADF Mobile"
- Chapter 2, "Introduction to ADF Mobile Application Development"

1 Introduction to ADF Mobile

This chapter introduces ADF Mobile—an application development framework within Oracle JDeveloper that enables you to create multi-featured applications for iOS and Android-powered devices. The applications run natively on mobile devices.

This chapter includes the following sections:

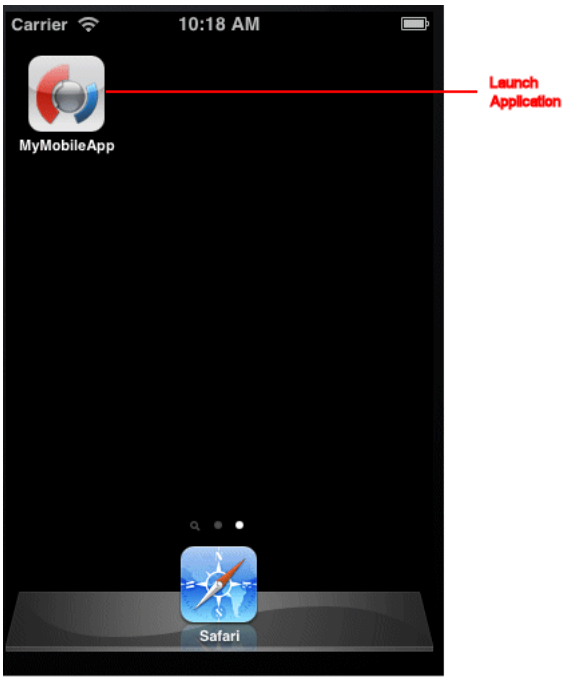
1.1 Introduction to ADF Mobile

ADF Mobile enables you to create an application that can house different types of content that run on mobile devices powered by the iOS and Android platforms. Within the framework of the application, the contained functional areas are referred to as *application features* and represent one or more specific pieces of the application's functionality.

Although each ADF Mobile application feature may have its own set of user-defined preferences, ADF Mobile enables you to apply a uniform style of user preferences to each of the application features embedded into the ADF Mobile application. Further, you can improve the end-user experience by grouping these mobile application features by functionality. For example, you can group a mobile application feature that provides customer contacts together with one for product inventory into the same ADF Mobile application. You can also control the display of the application features by such criteria as user role or device version.

From the end-user perspective, an application built using ADF Mobile is launched by clicking its application icon on the mobile device (see Figure 1–1).

Figure 1–1 Launching ADF Mobile Application



After the ADF Mobile application opens, the end user clicks an icon for an application feature. The application features can display as icons on a navigation bar (see the bottom portion of Figure 1–2) or in a page format with larger icons in a homescreen page, commonly referred to as a springboard (see Figure 1–3).

Figure 1–2 Application Features Displayed on Navigation Bar

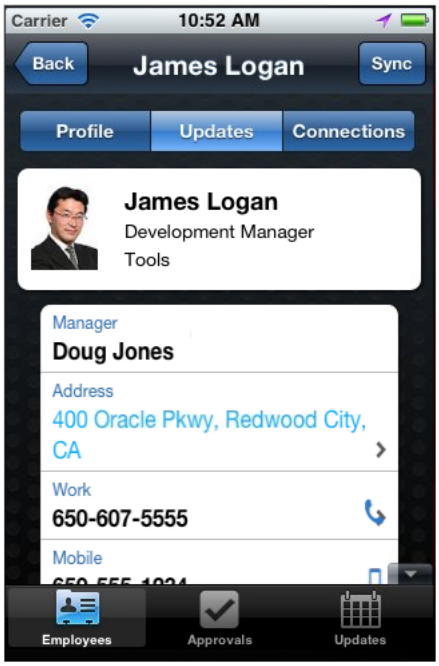
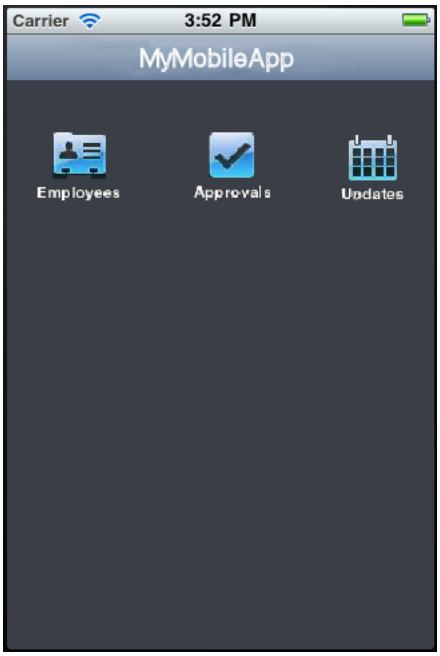


Figure 1–3 Application Features Displayed on Springboard



Because ADF Mobile supports integration of these application features with the mobile device services, the end user can, for example, place a call from the mobile device by clicking a name listed in the contacts for an application feature for a mobile field service report form or scan a receipt to update a mobile expense report. For more information on ADF Mobile user experience, see Appendix C, "ADF Mobile Application Usage."

The following types of application features can coexist within the same ADF Mobile application:

- ADF Mobile AMX—although you do not necessarily have to develop the application features that reside within the ADF Mobile application, this implementation approach allows you to define an application feature which utilizes much of the device's functionality through embedded components, with the metadata represented by an XML file. This application feature implementation, which is generated into HTML or JavaScript on a mobile device and which uses ADF data bindings, is developed declaratively in JDeveloper using a set of ADF Mobile-specific components.

  The workflows of ADF Mobile AMX pages are portable across platforms.

  For more information, see the following:

  – ADF Mobile AMX Views

  – Part IV, "Creating ADF Mobile AMX Application Features"

- Local HTML—this implementation approach allows you to employ an HTML file that might embed existing technologies, such as JavaScript and Java. You can use JDeveloper to implement an ADF Mobile application feature as a local HTML. For more information, see Section 5.9.1, "How to Define the Application Content."

- Remote URL (also known as server HTML)—this implementation approach allows you to point to a resource on the web by specifying a URL endpoint. You can use JDeveloper to implement an ADF Mobile application feature as a remote URL.

  The URL endpoints are portable across platforms.

For more information, see the following:.

- Server HTML
- Chapter 12, "Implementing Application Features as Remote URLs."

An ADF Mobile application enables the integration of its embedded application features with such native device services as phone, camera, GPS, and so on. These services can be accessed from the local HTML, ADF Mobile AMX, Java, and remote web applications.

1.2 ADF Mobile Runtime Architecture

An extension of the PhoneGap framework (see `http://www.phonegap.com/home`), the ADF Mobile architecture enables HTML5, as well as ADF-defined pages and task flows to be rendered in the same downloadable application.

ADF Mobile consists of the following parts:

- **Model** provided by Oracle ADF.

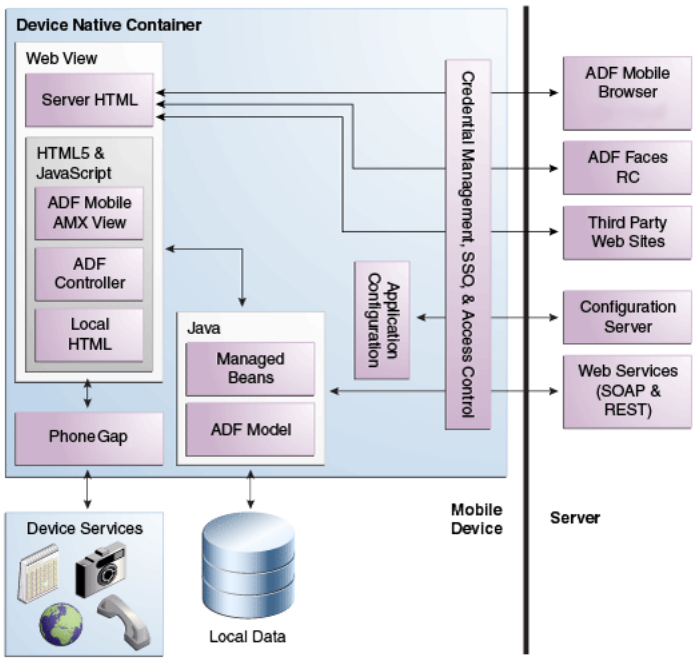
  Note: The model layer provides the facility to connect to a local database.

- **View** expressed as HTML or XML.
- **Controller** defined with XML.
- **Java runtime** powered by an embedded Java VM bundled with each application.

  Note: ADF Mobile's model-view-controller stack resides on a mobile device and represents reimplementation of ADF's model-view-controller layers. UI metadata is rendered to native components on device and is bound to the model through the ADF Model.

Figure 1–4 shows the overall runtime architecture of ADF Mobile.

Figure 1–4 ADF Mobile RunTime Architecture



As shown in Figure 1–4, the following elements comprise the ADF Mobile runtime architecture:

- **Device-Native Container** represents an application container, or template, compiled as a device-native application binary. This container provides the runtime environment for an ADF Mobile application to run as an on-device, native application in the mobile device's operating system (for example, iOS). Besides hosting the client-side components for an ADF Mobile application, it provides navigation utilities, such as a springboard and navigation bar, which enable access to particular application features.
- **Web View** is a part of the device native container that uses a mobile device's web engine to display and process web-based content. In an ADF Mobile application, the web view is the primary mechanism to render and deliver the application user interface.
- **Server HTML** represents a web-based user interface that is generated on the server and delivered as a web page to the ADF Mobile application. The application HTML code, business logic, and page flow logic are generated on a remote server. Server HTML can access device native services, such as camera, through the JavaScript API supported by PhoneGap, as long as it is running inside an ADF Mobile application. Common options for server HTML-based pages are ADF Mobile browser and Oracle ADF Faces rich client-based pages. For more information, see Chapter 12, "Implementing Application Features as Remote URLs."
- **Local HTML** represents web pages developed using JDeveloper or third-party tools that are directly embedded within an ADF Mobile application. These pages are delivered as a part of the ADF Mobile application. Local HTML files can access device native features through the JavaScript APIs supported by PhoneGap.
- **ADF Mobile AMX Views** are based on the ADF Mobile AMX technology that delivers a JSF-like development experience to working with an HTML5-based user interface. ADF Mobile AMX views are defined using UI and code editors provided

by JDeveloper. These views are embedded into an ADF Mobile application and deployed to a mobile device. At run time, the JavaScript engine in the web view renders ADF Mobile AMX view definitions into HTML5 components. Of the implementation approaches provided by ADF Mobile, application features developed using the ADF Mobile AMX components provide the most authentic device-native user experience through their extensive support of animation and gestures. For more information, see Part IV, "Creating ADF Mobile AMX Application Features".

- **ADF Controller** is represented by a mobile version of Oracle ADF controller that supports a subset of Oracle ADF task flow components available to a server-based Oracle ADF application. Both bounded and unbounded Oracle ADF task flows are supported, as well as a subset of events and scopes that are supported by the server-based ADF. For more information, see Section 7.2, "Creating Task Flows."
- **Java** provides a Java runtime environment for an ADF Mobile application. This Java Virtual Machine (JVM) is implemented in device-native code, and is embedded (or compiled) into each instance of the ADF Mobile application as part of the native application binary. The JVM is based on the JavaME Connected Device Configuration (CDC) specification.
- **Managed Beans** are Java classes that can be created to extend the capabilities of ADF Mobile, such as provide additional business logic for processing data returned from the server. Managed beans are executed by the embedded Java support, and therefore must conform to the JavaME CDC specifications. For more information, see Chapter 9, "Using Bindings and Creating Data Controls."
- **ADF Model** in an ADF Mobile application supports a subset of business logic components available to a server-based Oracle ADF application. ADF model contains the binding layer that provides an interface between the business logic components and user interface, as well as the execution logic to invoke REST or SOAP-based web services. For more information, see Chapter 9, "Using Bindings and Creating Data Controls."
- **Application Configuration** refers to services that allow key application configurations to be downloaded and refreshed. For example, URL endpoints for a web service or remote URL connection. Application configuration services download the configuration information from a WebDav-based server-side service. For more information, see Section 10.7, "Administering Web Services."
- **Credential Management and Access Control** refers to client-side services that provide security-related services for an ADF Mobile application. For example, a local credential store that securely caches user credentials to support an offline authentication or access control services that display or hide application features based on user access privileges. For more information, see Chapter 17, "ADF Mobile Application Security."
- **PhoneGap** is an open-source code library that provides a common JavaScript API to access various mobile device services, such as the camera. PhoneGap provides a majority of the device services integration for an ADF Mobile application. PhoneGap JavaScript APIs are further abstracted as device data controls in the JDeveloper design time for ADF Mobile AMX-based views, allowing for integration of device services by simply dragging and dropping data controls to their ADF Mobile AMX views.
- **Local Data** refers to data stores that reside on the device. In ADF Mobile, these are implemented as encrypted SQLite databases. Create Retrieve Update Delete (CRUD) operations are supported to this local data store through the Java layer,

using JDBC-based APIs. For more information, see Chapter 11, "Using the Local Database."

- On the server side, the **Configuration Server** refers to a WebDav-based server that hosts configuration files used by the application configuration services. The configuration server is delivered as a reference implementation. Any common WebDav services hosted on a common J2EE server can be used for this purpose. For more information, see Section 10.7, "Administering Web Services."
- On the server side, the **ADF Mobile Browser** refers to a framework for developing server-side applications that can be used for implementation of the remote URL ADF Mobile application feature. For more information, see Chapter 12, "Implementing Application Features as Remote URLs."

2

Introduction to ADF Mobile Application Development

This chapter describes the development approach for ADF Mobile applications.

This chapter includes the following sections:

2.1 Introduction to ADF Mobile Application Development

To ensure the best design for your ADF Mobile application, Oracle recommends that you follow an iterative, step-by-step development process.

Although you can develop the application features in addition to the ADF Mobile application itself, this workflow is not necessarily the standard case; application development involves creating the ADF Mobile application and embedding the application features that can be created by other developers. In other words, you create an ADF Mobile application by combining content that you create yourself with the content that was developed separately by someone else and then integrated into the ADF Mobile application. Knowing Xcode or Android application development is not a prerequisite for creating either the ADF Mobile application itself or the specific content for an application feature.

For more information, see Chapter 4, "Getting Started with ADF Mobile Application Development."

2.2 Infrastructure Requirements

Although development needs vary depending on the target platform, the minimum requirements for creating, building, and testing an ADF Mobile application are as follows:

- Oracle JDeveloper
- Oracle JDeveloper extension for ADF Mobile
- Platform-specific tools (such as Xcode, Android SDK, and so on)
- A mobile device or its simulator

For more information on prerequisites, see Section 3.2, "Prerequisites for Developing ADF Mobile Applications."

2.3 Supported Platforms, Devices, and Databases

ADF Mobile supports the following platforms:

- iOS 5 or later
- Android 2.2 or later

The following mobile devices are supported:

- Apple iPhone
- Apple iPad
- Android-powered devices

The supported database is SQLite (see Chapter 11, "Using the Local Database").

2.4 ADF Mobile Application Architecture

The following are potential architectures for your ADF Mobile application:

- A basic connected application: this type of application includes a user interface that is backed directly by a web service data control that, in turn, invokes a web service hosted on a server. Note that only REST-XML and SOAP-based data services can be accessed exclusively through data controls.
- A connected application that uses moderate or complex data services: this type of application includes a user interface backed by Java bean data controls. In addition, Java classes (POJOs) are used to perform the following:
  - To retrieve and persist data from more complex data sources.
  - To work with data retrieved from a server before the data is passed to the user interface: if the data source is based on REST-XML or SOAP formats, then the application is backed by Java bean data controls, which, in turn, are backed by POJOs that persist data and retrieve data from a web services data control. A typical implementation case would include a Java class that backs the user interface, and another class that hosts the data retrieval logic.

  If the application is to consume JavaScript Object Notation (JSON) data, then neither web service data controls nor SOAP and REST-XML web services are involved. Instead, JSON services must be invoked directly, and then JSON data parsed in the application code.
- A disconnected application: this type of application requires a local database populated with data. Typically, two modules of code are needed:
  - The first code module allows the user interface to retrieve data from the local database. This module is responsible for creating Java beans and Java bean data controls to serve data to the user interface; CRUD operations on the local database are performed using the JDBC code.

  - The second code module contains implementation of the Java classes that retrieve data from the server and populates the local database through JDBC code. This module can even run a background thread if you choose to implement a background data synchronization. These Java classes are responsible for data retrieval either through data controls (if REST-XML or SOAP data is consumed), or directly from JSON data sources.

For more information, see the following:

- Chapter 10, "Using Web Services"
- Chapter 11, "Using the Local Database"

2.5 Typical Development Stages

Typically, you perform the following activities when building an ADF Mobile application:

- Gather requirements
- Design
- Develop
- Deploy
- Test
- Debug
- Secure
- Enable access to the server-side data (optional)
- Redeploy
- Retest and debug
- Publish

The steps you take to build your ADF Mobile application will generally occur as follows:

1. During the design stage, consider the tasks a mobile user will be performing, keeping in mind that hand-held usage is different from that of a laptop or desktop computer. How will your ADF Mobile application help users get their jobs done? How will the users interact with the device? The more streamlined the application, the more they will use it.

   The next steps are to determine whether the application is required to work in a connected or disconnected mode; understand the device services integration requirements; determine the server-side data source and protocol.

   When designing server-side services, it is critical to provide for optimization for the mobile access: if server-side web services are very complex, it would be difficult for the mobile application to consume them. This is not only due to the amount of data that needs to be passed, but also the amount of the client-side logic that must be written to process the results. It is preferable to expose a set of server-side interfaces provided specifically for mobile. You also need to understand the client business services that must be developed, such as all Java modules and data controls that need to be created. In addition, you should create wireframes for the views and task flow in the application, which can help you to visualize the application functionality and assist in the development process.

As a final design step, you should consider how to partition the application functionality into separate application features that represent a group of functionality and associated views. Then you can start designing the client user interface and task flows by creating wireframes.

2. When setting up your work environment, download and install the ADF Mobile extension, and then install the necessary components and complete the required setup for development and deployment. For more information, see Chapter 3, "Setting Up the ADF Mobile Environment."

3. When creating your ADF Mobile application using JDeveloper, you use the ADF Mobile application creation wizard. The artifacts that result from creating the application include descriptor files for the ADF Mobile application and for the application features, default images for icons and tabs for all supported platforms, and a set of data controls used for accessing the services of a mobile device (such as camera, GPS, or email).

   For more information, see the following:

   - Chapter 4, "Getting Started with ADF Mobile Application Development"
   - Chapter 5, "Defining an ADF Mobile Application"

4. When implementing the application features, you perform a thorough evaluation of the business need to determine which application features should be included within the ADF Mobile application. Using the overview editors provided by ADF Mobile, your tasks for implementing an application feature include identifying its type (HTML, remote URL, or ADF Mobile AMX, or native UI), its display properties (display name, navigation bar and springboard icon), and its display behavior as dictated by both the mobile device capabilities and the user role.

   For more information, see the following:

   - Chapter 5, "Defining an ADF Mobile Application"
   - Part IV, "Creating ADF Mobile AMX Application Features"
   - Chapter 12, "Implementing Application Features as Remote URLs"

5. During the application deployment stage, you start with creating a deployment profile that will support devices and simulators for its respective platform. Creating a deployment profile may include selecting the display icon used for the ADF Mobile application itself in various orientations (such as landscape or portrait) and setting the application's signing options (such as debug or release).

   You then proceed to deploying your application to the mobile device or simulator.

   Note: With ADF Mobile applications, it is required that you deploy to the device or simulator before attempting any testing and debugging (see Chapter 18, "Testing and Debugging ADF Mobile Applications"). The application cannot be run until you deploy it.

   For more information, see Chapter 16, "Deploying ADF Mobile Applications."

6. During the testing and debugging stage, you test, debug, and optimize your application. For more information, see Chapter 18, "Testing and Debugging ADF Mobile Applications."

7. Enabling and configuring security for the application typically requires configuring the login server, such as the Oracle Identity Connect server, or it can

be any web page protected by the basic HTTP authentication mechanism. In addition, you may have to configure the access control server.

For more information, see Chapter 17, "ADF Mobile Application Security."

8. After ensuring that your application functions as expected at a basic level, you can implement the Java code to access the server-side data:
   - For connected applications, these Java classes should invoke web services directly. If your application uses SOAP or REST XML-based data sources, you invoke web services through data controls, with the assistance of a set of helper classes that you can invoke from your code to invoke the data controls and return data. If your application uses JSON-based data sources, your code should directly invoke the JSON service and return data, after which you need to parse the JSON data from the server and populate the objects holding data collections accordingly.
   - For disconnected applications, your code should populate the local SQLite database. Then, the code that backs the user interface can retrieve data from the SQLite database instead of directly invoking web services.

   For more information, see the following:
   - Section 2.4, "ADF Mobile Application Architecture"
   - Chapter 10, "Using Web Services"
   - Chapter 11, "Using the Local Database"
9. During the second round of deployment, you ensure that after adding security to your application and enabling access to the server-side data the application deployment runs as expected and the application is ready for the final testing and debugging.
10. During the final round of testing and debugging, you focus on the security and the server-side data access functionality ensuring that their integration into the application did not result in errors and unexpected behavior.
11. Deploying the application to the production environment typically involves publishing to an enterprise server, the Apple App Store, or an application marketplace, such as Google Play. After you publish the ADF Mobile application, end users can download it to their mobile devices and access it by clicking the designated icon (see Appendix C, "ADF Mobile Application Usage"). The application features bear the designated display icons and display as appropriate to the end user and the user's device.

2.6 The Application Lifecycle

The lifecycle of an ADF Mobile application is driven by events that occur at the levels of the mobile device operating system, the JVM, and ADF Mobile. The application's reaction to these events is enabled through the use of the `LifeCycleListener`'s methods. For more information, see Section 5.6, "About Lifecycle Event Listeners."

2.7 Sample Applications

After setting up your development environment (see Chapter 3, "Setting Up the ADF Mobile Environment"), you can examine ADF Mobile sample applications located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer.

The sample applications demonstrate the following:

- How to create a basic HelloWorld ADF Mobile application.
- How to enable the application to react to life cycle events.
- How to use skinning.
- How to develop the ADF Mobile AMX application feature, including the user interface, navigation from page to page, managed beans, data change events, and so on.

For more information, see Appendix E, "ADF Mobile Sample Applications."

2.8 ADF Mobile AMX Application Feature

ADF Mobile AMX is basically a subframework within ADF Mobile and provides a set of layout, field, and data components that enable you to create an application feature that behaves appropriately for both the iOS and Android user experience. ADF Mobile AMX supports some of Oracle ADF components, data controls, bindings, and the Expression Language that belong to the following layers:

- Model layer. See Chapter 9, "Using Bindings and Creating Data Controls."
- User interface (UI) layer. See the following:
  - Chapter 7, "Creating ADF Mobile AMX Pages"
  - Chapter 8, "Creating ADF Mobile AMX User Interface"
- Expression Language. See Section 9.2, "Understanding EL Support."

Differences in every layer exist due to the inherent differences between a mobile application and a web application. For more information, see Section 2.9, "Comparison of ADF Mobile and Server-Based Oracle ADF."

2.9 Comparison of ADF Mobile and Server-Based Oracle ADF

Table 2–1 summarizes the functionality, components, and technologies supported by ADF Mobile and compares them to those supported by a server-based Oracle ADF application.

Table 2–1 Differences Between ADF Mobile and Server-Based Oracle ADF Application

Layer	Supported by ADF Mobile	Supported by Server-Based Oracle ADF
ADF Model	■ SOAP, REST XML, and REST JSON-based data sources and data controls ■ Subset of the model layer Java (for example, there is no support for Java web service proxy) ■ JDBC	Full range of data sources and data controls
View	■ Locally rendered ADF Mobile AMX or custom HTML5 pages. ■ HTML pages rendered on the server.	■ Server-rendered only ■ ADF Faces rich client and Trinidad JSF components

Table 2–1 (Cont.) Differences Between ADF Mobile and Server-Based Oracle ADF

Layer	Supported by ADF Mobile	Supported by Server-Based Oracle ADF
Controller	For ADF Mobile AMX application feature (see Section 2.8, "ADF Mobile AMX Application Feature"): ▪ Subset of Oracle ADF task flow components ▪ Logic resides on the mobile device	Full ADF task flow
Java Support	JavaME CDC and Java 1.4	Java EE with the latest Java

As described in Table 2–1, in the view layer, ADF Mobile provides you with an option of using locally rendered or server-rendered views to present the user interface. ADF Mobile AMX enables the most seamless user experience. For the server-based Oracle ADF, the user interface is rendered on the server and is primarily provided through ADF Faces rich client components or Trinidad components, both of which are based on the JSF technology.

In the controller layer, an abbreviated version of the ADF task flow that supports a subset of components is provided when ADF Mobile AMX contents are used in a mobile application. ADF Mobile task flow supports views, control flow case, wild card control flow case, method calls, and routers, but does not support regions. In ADF Mobile, the page flow logic resides entirely on a mobile device, enabling the page navigation without a round trip to the server.

In the ADF Model layer, ADF Mobile supports SOAP, REST XML, and REST JSON as the server-side data sources. In addition, ADF Mobile supports JDBC connection and APIs to the local database, but it does not support ADF business components. Furthermore, ADF Mobile does not implement all of the Java methods supported by the server-based ADF. For example, you cannot use the programmatic access binding context. Instead, you can access the binding data by invoking the Expression Language (EL) expressions. In addition, since ADF Mobile does not support the Java proxy for web services, to access web services programmatically, you must use data controls in conjunction with the web services invocation helper classes.

For Java support, ADF Mobile's embedded Java virtual machine follows the JavaME CDC specification, which is based on Java 1.4, meaning that you cannot use any Java 1.5 or later features in your Java code.

Part II

Getting Started

Describes how to set up the development environment for ADF Mobile application development and provides instructions on creating an ADF Mobile application.

Part II contains the following chapters:

- Chapter 3, "Setting Up the ADF Mobile Environment"
- Chapter 4, "Getting Started with ADF Mobile Application Development"

3

Setting Up the ADF Mobile Environment

This chapter provides information on setting up the ADF Mobile environment for application development and deployment.

This chapter includes the following sections:

3.1 Introduction to the ADF Mobile Environment

Before developing an ADF Mobile application, you must set up your development environment by downloading, installing, and configuring various software components.

3.2 Prerequisites for Developing ADF Mobile Applications

Prerequisites for developing an ADF Mobile application vary depending on the type of work you are planning to do, as well as your target platform:

- What You Need to Create an Application
- What You Need to Deploy an ADF Mobile Application to a Development Environment

3.2.1 What You Need to Create an Application

Before you start creating an ADF Mobile application (see Chapter 4, "Getting Started with ADF Mobile Application Development") for iOS, ensure that you have the following components available:

- A computer running Mac OS 10.6.7 or later
- Xcode 4.2 or later (see `http://developer.apple.com/xcode/` and Section 3.4.2, "How to Prepare Xcode")
- Version 5 or later of iOS SDK (see `http://developer.apple.com/devcenter/ios/`)
- An Apple developer ID

- Oracle JDeveloper (see Section 3.3, "Setting Up JDeveloper" and Section 3.4.1, "How to Configure JDeveloper for iOS Development")
- Oracle JDeveloper extension for ADF Mobile (see Section 3.3, "Setting Up JDeveloper")

For more information, see Section 3.4, "Setting Up Development Tools for iOS Platform."

If Android is your target platform, the following is required:

- Android SDK with Platform 2.3 or later and its tools (see `http://developer.android.com/sdk/`)
- Oracle JDeveloper (see Section 3.3, "Setting Up JDeveloper" and Section 3.5.1, "How to Configure JDeveloper for Android Development")
- Oracle JDeveloper extension for ADF Mobile (see Section 3.3, "Setting Up JDeveloper")

For more information, see Section 3.5, "Setting Up Development Tools for Android Platform."

You do not need to install any additional tools for creating specific types of ADF Mobile application content (HTML, remote URL, or ADF Mobile AMX). For more information, see Section 5.9, "Defining the Content Types for an Application Feature."

3.2.2 What You Need to Deploy an ADF Mobile Application to a Development Environment

Before you deploy your ADF Mobile application (see Chapter 4, "Getting Started with ADF Mobile Application Development"), ensure that you have the following components available:

- All components listed in Section 3.2.1, "What You Need to Create an Application."
- The ADF Mobile application.
- Various login credentials. For more information, see Chapter 16, "Deploying ADF Mobile Applications."

3.3 Setting Up JDeveloper

Setting up your development environment starts with installing Oracle JDeveloper and its ADF Mobile extension.

Before you begin:

Download and install Oracle JDeveloper. Select the Studio Developer (All Features) role when prompted.

For more information, see *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*. If your target platform is iOS, see the section about using Oracle JDeveloper on the Mac OS X platform in *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

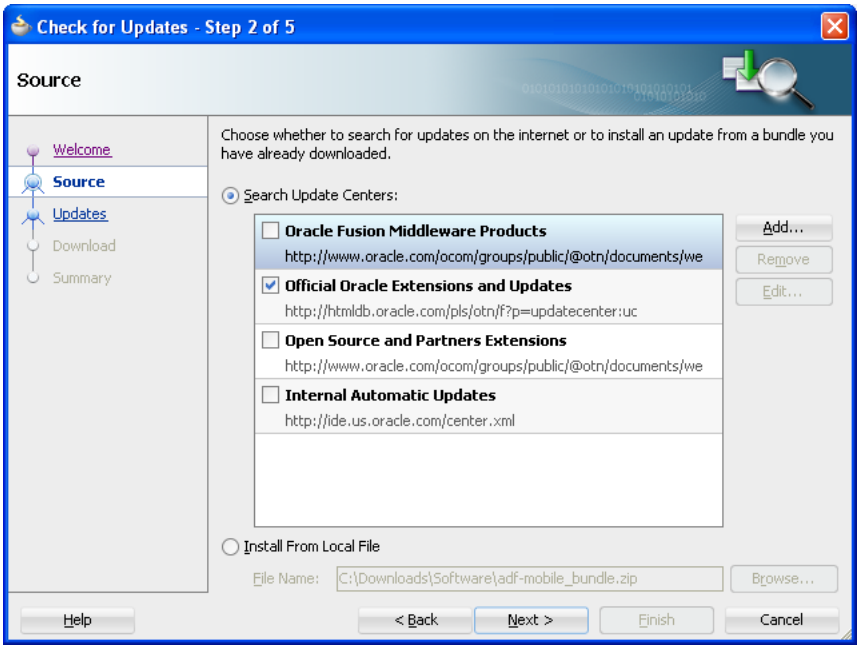
To download and install the ADF Mobile extension:

1. In JDeveloper, choose **Help**, then **Check for Updates**, and then click **Next**.

Note: You might need to configure proxy settings by selecting **Tools > Preferences** from the main menu, and then **Web Browser and Proxy** from the tree on the left of the **Preferences** dialog.

2. In the **Source** page that Figure 3–1 shows, select **Official Oracle Extensions and Updates**, and then click **Next**.

Figure 3–1 Checking for Updates in JDeveloper

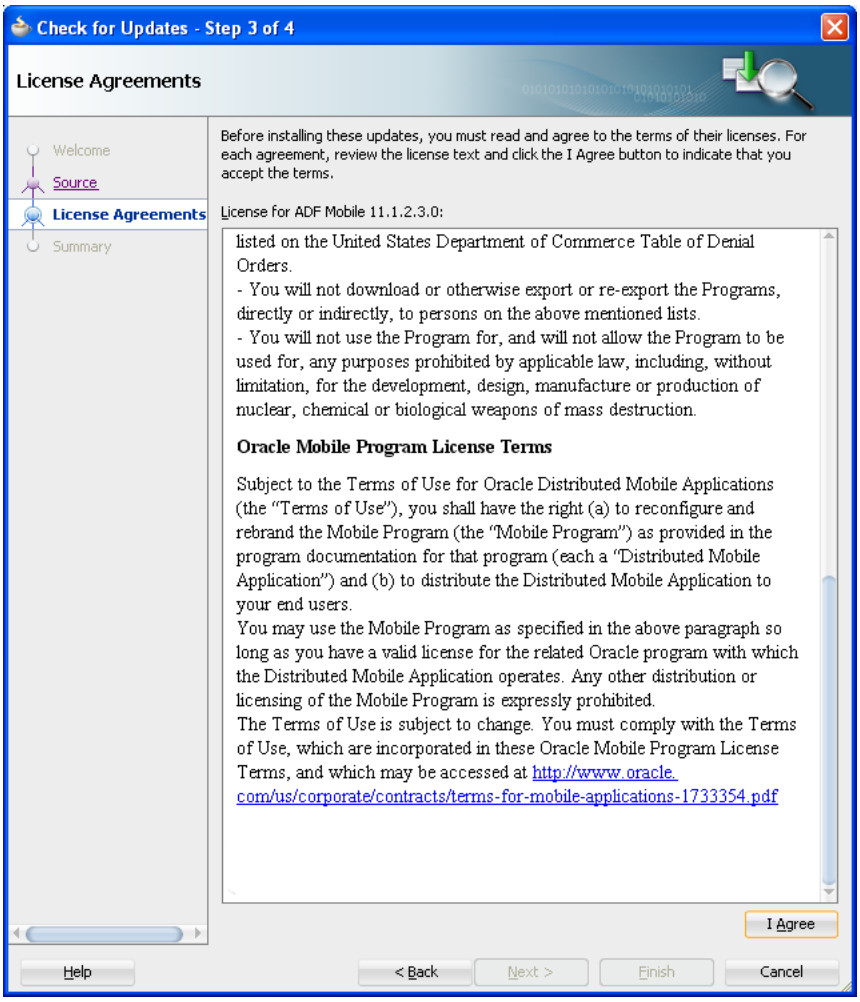


3. In the **Updates** dialog, select the **ADF Mobile** update.
4. In the License Agreements page, shown in Figure 3–2, review *The Oracle Technology Network License Terms for Oracle Mobile*.

Note: You must comply with all of the license terms and conditions with respect to the Oracle ADF Mobile Program. The Oracle ADF Mobile Program may be accessed at `http://www.oracle.com/technetwork/indexes/downloads/index.html`.

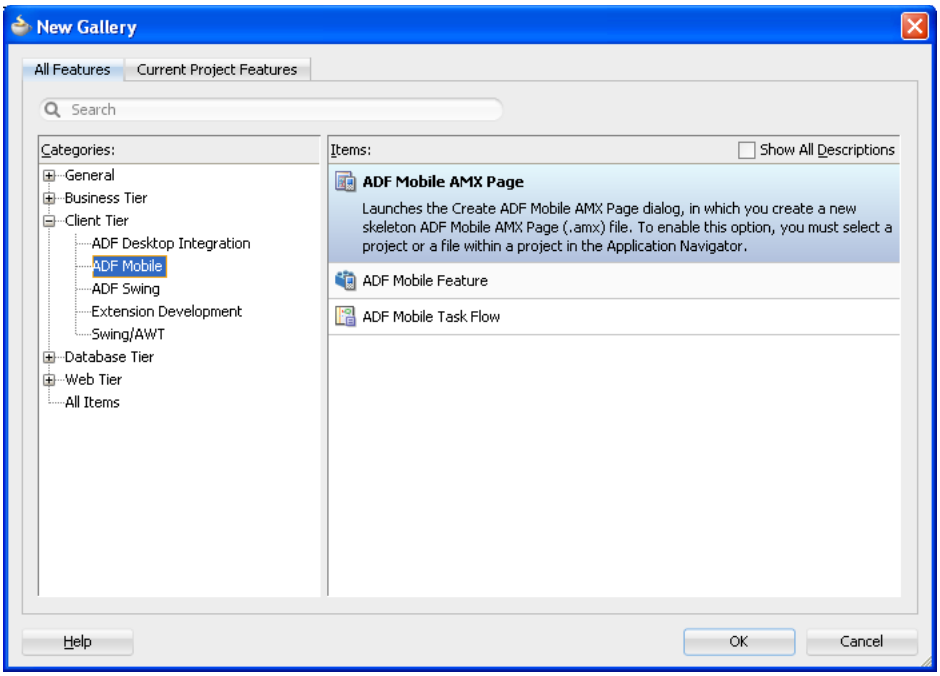
5. Click **I Agree**.

Figure 3–2 Licensing Agreements for the Oracle ADF Mobile Program



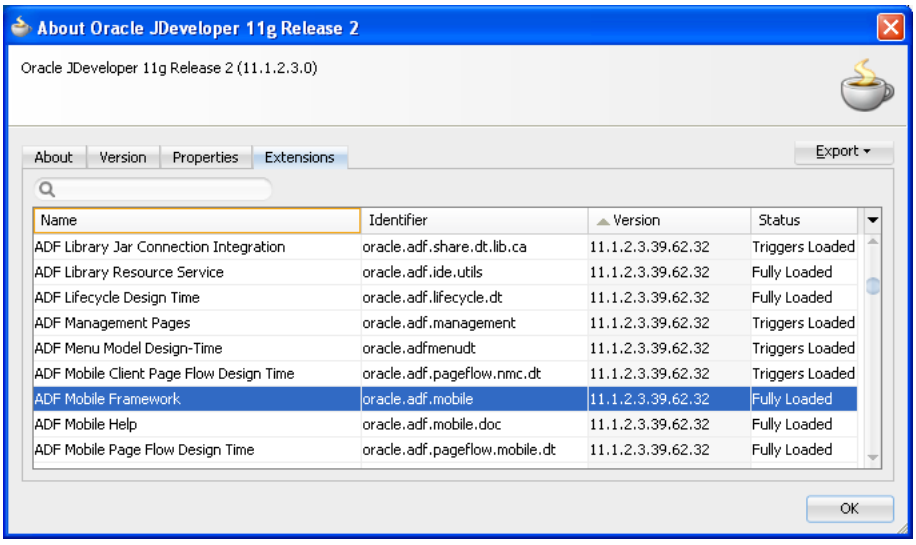
6. Click **Next**, and then click **Finish**.
7. Restart JDeveloper.
8. Check whether or not ADF Mobile was successfully added to JDeveloper:
   - Select **File > New** from the main menu to open the **New Gallery**.
   - Select the **All Features** tab.
   - In the **Categories** pane, expand the **Client Tier** node and make sure it contains **ADF Mobile** (see Figure 3–3).

Figure 3–3 Verifying ADF Mobile Installation



In addition, verify that you installed the correct version of ADF Mobile. To do so, select **Help > About** from the main menu, then select the **Extensions** tab on the About Oracle JDeveloper dialog, and then examine the extension list entries by scrolling down to **ADF Mobile**, as Figure 3–4 shows.

Figure 3–4 Verifying ADF Mobile Version



For platform-specific information, see the following:

- Section 3.4.1, "How to Configure JDeveloper for iOS Development"
- Section 3.5.1, "How to Configure JDeveloper for Android Development"

3.3.1 How to Configure the Development Environment for Platforms and Form Factors

Before you start developing an ADF Mobile application, you must configure JDeveloper Preferences for appropriate platforms (see Section 3.3.1.2, "Configuring the Environment for Target Platforms") and form factors (see Section 3.3.1.1, "Configuring the Environment for Form Factors").

3.3.1.1 Configuring the Environment for Form Factors

A form factor is a specific device configuration. Each form factor is identified by a name that you specify for it and contains information on the specified resolution denoted by pixel width and pixel height.

Form factors defined in preferences are used in the ADF Mobile AMX page Preview tab (see Section 7.3.2.2, "Using the Preview"). You can select or switch between various form factors to see how an ADF Mobile AMX page is rendered in various form factors. You can also see multiple form factors applied to the same page using the split screen view.

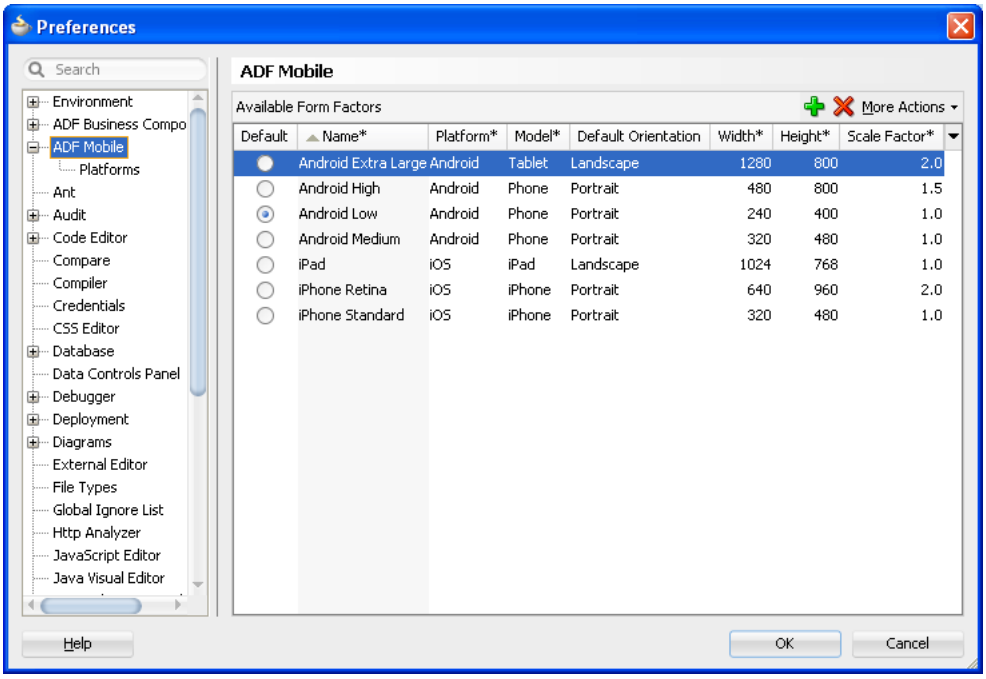
Before you begin:

Download and install JDeveloper and the ADF Mobile extension, as described in Section 3.3, "Setting Up JDeveloper."

To configure the form factors:

1. Open Preferences by selecting **Tools > Preferences** from the main menu in JDeveloper.
2. In the **Preferences** dialog that Figure 3–5 shows, select **ADF Mobile** from the tree on the left.

Figure 3–5 Defining Form Factors



The **ADF Mobile** page is populated with default settings.

This preference page allows you to create and manage a set of named form factors that combine a screen resolution size and platform.

3. To create a new form factor, click **New**, and then set the following:
   - **Name**: a meaningful string that is used to identify the form factor.
   - **Platform**: the platform of the mobile device.
   - **Model**: the type of the mobile device.
   - **Default Orientation**: the default device orientation used in the ADF Mobile AMX page Preview tab. It might be Portrait or Landscape. Select this setting from the drop-down list of values. The default value is Portrait and it is prepopulated during creation of the new form factor.
   - **Resolution Width**: width, in pixels. This value must be a positive integer, and its input is validated.
   - **Resolution Height**: height, in pixels. This value must be a positive integer, and its input is validated.
   - **Scale Factor**: the display scale factor. This value must be either one of 1.0, 2.0, or 3.0.

   Note: If you do not set the name and resolution for your form, ADF Mobile will display an error message.

4. If you need to revert to default settings, click **More Actions > Restore Defaults**.
5. Click **OK** to finalize your settings.

3.3.1.2 Configuring the Environment for Target Platforms

To start developing for one of the platforms supported by ADF Mobile, you need to provide JDeveloper with such information as the name of the platform and directories on your development computer that are to house the platform-specific tools and data.

Before you begin:

Download and install JDeveloper and the ADF Mobile extension, as described in Section 3.3, "Setting Up JDeveloper."

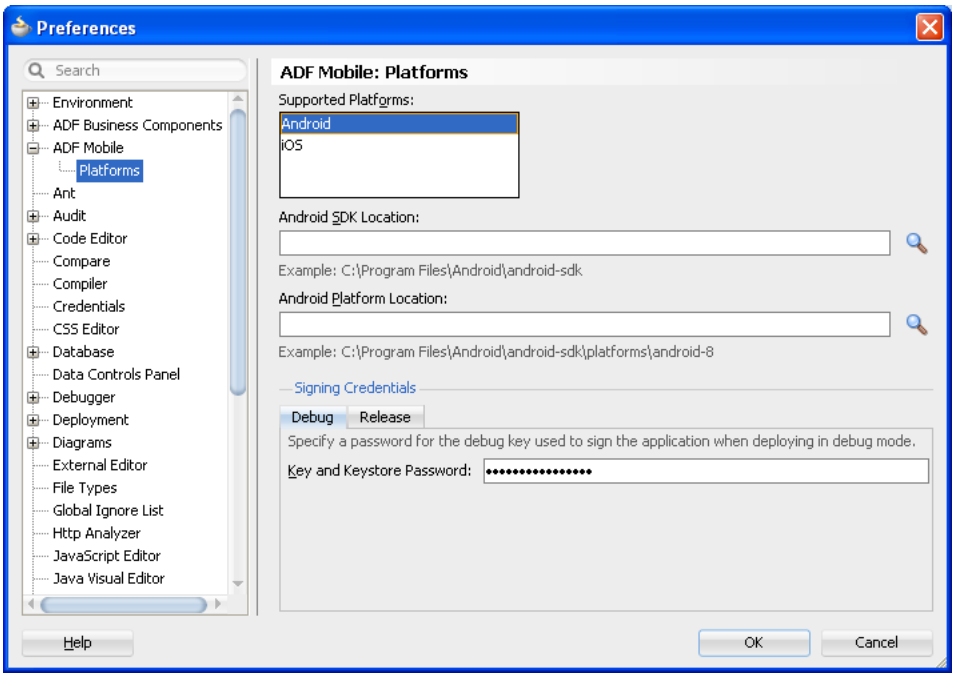
To configure your development environment for the target platforms:

1. Select **Tools > Preferences** from JDeveloper's main menu to open Preferences.
2. In the **Preferences** dialog that Figure 3–5 shows, select **ADF Mobile** from the tree. The **ADF Mobile** page that opens is populated with default form factor settings.
3. Select **Platforms** from the tree to open the **ADF Mobile: Platforms** page that contains the path and configuration parameters for the supported platforms, as Figure 3–6 and Figure 3–7 show.

   Each platform-specific page hosts the preferences for the platform SDK (Android or iOS), collecting any necessary information such as the path that ADF Mobile needs to compile and deploy either Android or iOS projects:

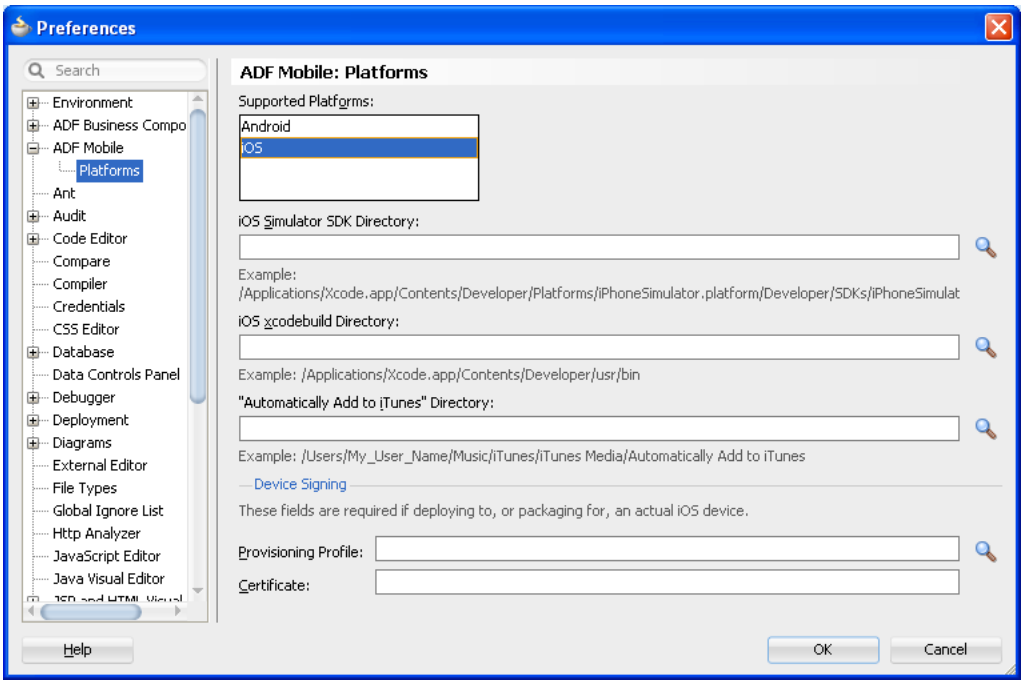
   - If you select **Android** from the list of supported platforms, specify the Android SDK location on your computer, the local directory of your target Android platform, and provide information on the signing credentials (see Figure 3–6).

Figure 3–6 Configuring Platform Preferences for Android



- If you select **iOS** from the list of supported platforms (see Figure 3–7), specify the following:
  - iOS Simulator SDK location on your Mac OS-powered computer
  - Location of the Xcodebuild utility (see Section 16.4.1, "How to Deploy an iOS Application to an iOS Simulator")
  - Location of the iTunes media files, including the mobile applications that are synchronized to the iOS-powered device
  - The iOS-powered device signing information (see Section 16.2.4.2, "Setting the Device Signing Options")

Figure 3–7 Configuring Platform Preferences for iOS



3.4 Setting Up Development Tools for iOS Platform

In addition to general-purpose tools listed in Section 3.2.1, "What You Need to Create an Application," you might want to set up an iPhone or iPad when getting ready for development of an ADF Mobile application for the iOS platform (see Section 3.4.3, "How to Set Up an iPhone or iPad").

Since iPhone and iPad simulators are included in the iOS SDK installation, you do not need to separately install them. For more information, see Section 3.4.4, "How to Set Up an iPhone or iPad Simulator."

3.4.1 How to Configure JDeveloper for iOS Development

After installing components listed in Section 3.2.1, "What You Need to Create an Application," you have to configure JDeveloper to make use of them.

To configure JDeveloper for iOS development:

1. In JDeveloper, select **Tools** > **Preferences** > **ADF Mobile** from the main menu.
2. In the Preferences page, specify the **Platforms**. For more information, see Figure 3–7, "Configuring Platform Preferences for iOS" and Section 3.3.1.2, "Configuring the Environment for Target Platforms."
3. Click **OK**.

3.4.2 How to Prepare Xcode

After installing Xcode, you have to run it at least once and complete the Apple licensing and setup dialogs. If these steps are not performed, any build and deploy cycle from JDeveloper to Xcode or device simulator will fail with a "Return code 69" error.

3.4.3 How to Set Up an iPhone or iPad

In your ADF Mobile application development and deployment, you can use either the iPhone, iPad, or their simulators (see Section 3.4.4, "How to Set Up an iPhone or iPad Simulator"). If you are planning to use an actual iPhone or iPad, which is preferable for testing (see Section 18.2, "Testing ADF Mobile Applications"), you need to connect it to your computer to establish a link between the two devices.

To deploy to an iOS-powered device, you need to have an iOS-powered device with a valid license, certificates, and distribution profiles. For more information, see Chapter 16, "Deploying ADF Mobile Applications."

3.4.4 How to Set Up an iPhone or iPad Simulator

In your ADF Mobile application development and deployment, you can use either the iOS-powered device itself (see Section 3.4.3, "How to Set Up an iPhone or iPad") or its simulator. Deploying to a simulator is usually much faster than deploying to a device, and it also means that you do not have to sign the application first.

A simulator can be invoked automatically, without any additional setup.

Note: Before attempting to deploy your application from JDeveloper to a device simulator, you must first run the simulator.

If you are planning to use web services in your application and you are behind a corporate firewall, you might need to configure the external network access. You do so by modifying the network settings in the System Preferences on your development computer. For more information, see the "Setting Browser Proxy Information" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

3.5 Setting Up Development Tools for Android Platform

In addition to the general-purpose tools listed in Section 3.2.1, "What You Need to Create an Application," you might want to set up an Android-powered device when getting ready for development of an ADF Mobile application for the Android platform (see Section 3.5.3, "How to Set Up an Android-Powered Device").

Since emulators are included in the Android SDK installation, you do not need to separately install them. However, you cannot use an emulator until you create its configuration. For more information, see Section 3.5.4, "How to Set Up an Android Emulator."

To develop for the Android platform, you can use any operating system that is supported by both JDeveloper and Android.

For more information, see the Android Developers web site at `http://developer.android.com/tools/index.html`.

3.5.1 How to Configure JDeveloper for Android Development

After you installed and configured components listed in Section 3.2.1, "What You Need to Create an Application," you have to configure JDeveloper to make use of them:

To configure JDeveloper for Android development:

1. Follow instructions relevant to Android provided in Section 3.3, "Setting Up JDeveloper" and Section 3.3.1, "How to Configure the Development Environment for Platforms and Form Factors."
2. Click **OK**.

3.5.2 How to Install the Android SDK

Android SDK includes development tools that you need to build applications for Android-powered devices. Since the Android SDK is modular, it allows you to download components separately depending on your target Android platform and your application requirements.

When choosing the platform, keep in mind that ADF Mobile supports Android 2.3 or later.

Before you begin:

Ensure that your environment meets the operating system, JDK version, and hardware requirements listed in `http://developer.android.com/sdk/index.html`.

> **Note:** Ant, Eclipse, and Linux requirements are not applicable to the ADF Mobile development environment.

To install the Android SDK:

1. Download the Android SDK starter package from `http://developer.android.com/sdk/index.html`.
2. Complete the installation by following the instructions provided in `http://developer.android.com/sdk/installing.html`. Skip step 3 in the Android SDK installation instructions, as configuring Eclipse is not required for the ADF Mobile application development.

3.5.3 How to Set Up an Android-Powered Device

In your ADF Mobile application development and deployment, you can use either the Android device itself, which is preferable for testing (see Section 18.2, "Testing ADF Mobile Applications"), or an emulator (see Section 3.5.4, "How to Set Up an Android Emulator").

For information on how to set up the Android-powered device, follow the instructions from the "Using Hardware Devices" section of the Android Developers web site at `http://developer.android.com/tools/device.html`.

> **Note:** You might experience issues when using USB connectivity for the device-based debugging. For more information, see Section 18, "Testing and Debugging ADF Mobile Applications."

Your target Android-powered device might not be listed in the USB device driver's `.inf` file, resulting in the failure to install the Android Debug Bridge (ADB). You can eliminate this issue as follows:

1. Find the correct values for your device.

2. Update the `[Google.NXx86]` and `[Google.NTamd64]` sections of the `android_winusb.inf` file.

For more information, see
http://developer.android.com/sdk/win-usb.html.

3.5.4 How to Set Up an Android Emulator

In your ADF Mobile application development and deployment, you can use either the Android device itself (see Section 3.5.3, "How to Set Up an Android-Powered Device") or its emulator. Deploying to an emulator is usually much faster than deploying to a device, and it also means that you do not have to sign the application first.

For information on how to create an emulator configuration called Android Virtual Device (AVD), follow the instructions from the "Managing Virtual Devices" section of the Android Developers web site at
http://developer.android.com/tools/devices/index.html.

You need to create an AVD for each Android platform on which you are planning to test your application.

For information on how to use the emulator, see the "Using the Android Emulator" section in the Android Developers web site at
http://developer.android.com/tools/devices/emulator.html.

3.5.4.1 Saving the Emulator State

You may find it useful to be able to save the emulator state or reuse the saved state. To do so, you manipulate the `avd` files or folders that are located in the `C:\Users\`*`username`*`\.android\avd` directory (on a Windows computer). Each `avd` folder contains several files, such as `userdata.img`, `userdata.qemu.img`, and `cache.img`. You can copy the `cache.img` file to another emulator's `avd` folder to use that state with another emulator. Alternatively, you can run commands listed in Table 3–1 from the command line.

Table 3–1 Commands for Saving the Android Emulator State

Command	Description
`-snapstorage <file>`	File that contains all the emulator state snapshots. Default value: *`datadir`*`/snapshots.img`
`-no-snapstorage`	Disables all of the emulator state snapshot functionality by preventing a snapshot storage file from being mounted.
`-snapshot <name>`	Name of the emulator state snapshot within the storage file for the autostart and autosave. Default value: `default-boot`
`-no-snapshot`	Performs a full boot without performing autosave; `vmload` and `vmsave` operate on snapstorage.
`-no-snapshot-save`	Does not autosave to snapshot on exit; abandons the changed state.
`-snapshot-list`	Shows a list of available snapshots.
`-no-snapshot-update-time`	Does not try to correct the snapshot time on restore.
`-wipe-data`	Resets the use data image by copying it from `initdata`.

For example, to use the saved state from the `testsnap` snapshot and auto-save back to it on exit, the following command should be executed from `C:\Android\`*`android sdk directory`*`\tools>`:

```
emulator -avd Android_2.2_1 -snapshot testsnap
```

> **Caution:** When using this utility, keep in mind that in the process of loading, all contents of the system, including the user data and SD card images, will be overwritten with the contents they held when the snapshot was made. Unless saved in a different snapshot, any changes will be lost.

3.5.4.2 Creating, Saving, and Reusing the SD Card

Execute the following commands to create, save, and reuse the SD card:

- To create an SD card:

  ```
  C:\android sdk directory\tools>mksdcard -l SD500M 500M C:\Android\sd500m.img
  ```

- To list existing AVDs:

  ```
  C:\android sdk directory\tools>android list avd
  ```

  This produces a listing similar to the following:

  ```
  Name: Android_2.2_1
  Path: C:\Users\username\.android\avd\Android_2.2_1.avd
  Target: Android 2.2 (API level 8)
  Skin: WVGA800
  Sdcard: 200M
  ------------
  Name: Android_2.3.1
  Path: C:\Users\username\.android\avd\Android_2.3.1.avd
  Target: Android 2.3.1 (API level 9)
  Skin: WVGA800
  ------------
  Name: Android_2.3.3
  Path: C:\Users\username\.android\avd\Android_2.3.3.avd
  Target: Android 2.3.3 (API level 10)
  Skin: WVGA800
  ------------
  Name: Android_3.0
  Path: C:\Users\username\.android\avd\Android_3.0.avd
  Target: Android 3.0 (API level 11)
  Skin: WXGA
  ```

- To start the Android_2.2_1 with the SD card that has just been created:

  ```
  C:\Android\android sdk directory\tools>emulator -avd Android_2.2_1 -sdcard
  C:\Android\sd500m.img
  ```

- To list the running Android emulator instances:

  ```
  C:\Android\android sdk directory\platform-tools>adb devices
  ```

- To copy a test image to the SD card (this requires the emulator to restart):

  ```
  C:\Android\sdk\platform-tools>adb push test.png sdcard/Pictures
  85 KB/s (1494 bytes in 0.017s)
  ```

3.5.4.3 Configuring the Network

From the Android emulator, you can access your host computer through the 10.0.2.2 IP. To connect to the emulator from the host computer, you have to execute the `adb` command from a command line on your development computer or from a script to set up the port forwarding.

To forward socket connections, execute

`adb forward` *`local remote`*

using the following forward specifications:

- `tcp:`*`port`*
- `localabstract:`*`unix domain socket name`*
- `localreserved:`*`unix domain socket name`*
- `localfilesystem:`*`unix domain socket name`*
- `dev:`*`character device name`*
- `jdwp:`*`process pid`* (remote only)

For example, an arbitrary client can request connection to a server running on the emulator at port 55000 as follows:

```
adb -e forward tcp:8555 tcp:55000
```

In this example, from the host computer, the client would connect to `localhost:8555` and communicate through that socket.

For more information, see the "Android Debug Bridge" section in the Android Developers web site at `http://developer.android.com/tools/help/adb.html`.

3.5.4.4 Configuring the Network Proxy

If your development computer is behind a corporate firewall, you might need to execute the following command to start the emulator and initiate its connection with the browser:

```
emulator -avd myavd -http-proxy myproxy
```

4

Getting Started with ADF Mobile Application Development

This chapter describes how to use the Oracle JDeveloper wizards and tools to create a basic ADF Mobile application and also describes the artifacts that are automatically generated when you create an application.

This chapter includes the following sections:

4.1 Introduction to Declarative Development for ADF Mobile Applications

You can create, deploy, and test an ADF Mobile application without writing a line of code because the JDeveloper design experience is enhanced to include support of ADF Mobile application development.

4.2 Creating an Application Workspace

The ADF Mobile extension provides JDeveloper with the application templates that seed the completed project with basic files. The first steps in building an ADF Mobile application are to assign it a name and to specify a directory where its source files will be saved. By creating an application with the application templates provided by JDeveloper, the workspace is automatically organized into projects, along with the required configuration files.

4.2.1 How to Create a Workspace for an ADF Mobile Application

You create an application using the application creation wizard.

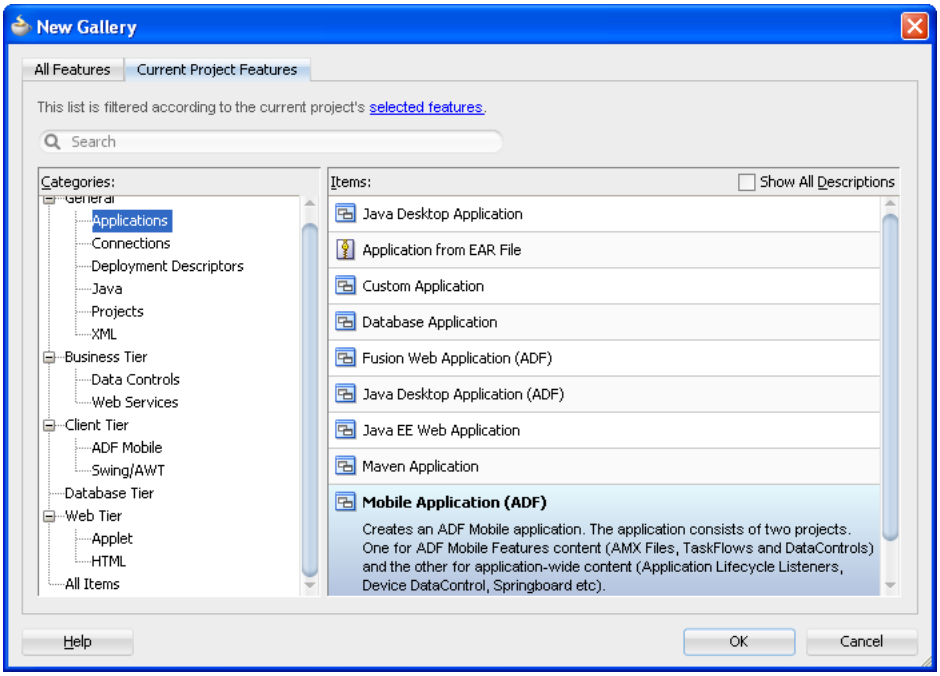
Before you begin:

You must download the ADF Mobile application extension. For more information, see Section 3.3, "Setting Up JDeveloper." You may need to download and configure the ADF Mobile application extension for all target platforms. Complex applications with native features (that is, features that integrate with an operating system's native controls) may require additional `SHLIB_PATH` and `CLASSPATH` variables.

To create an ADF Mobile application:

1. Choose **File**, then **New**, and then **Mobile Application (ADF)**, as shown in Figure 4–1.

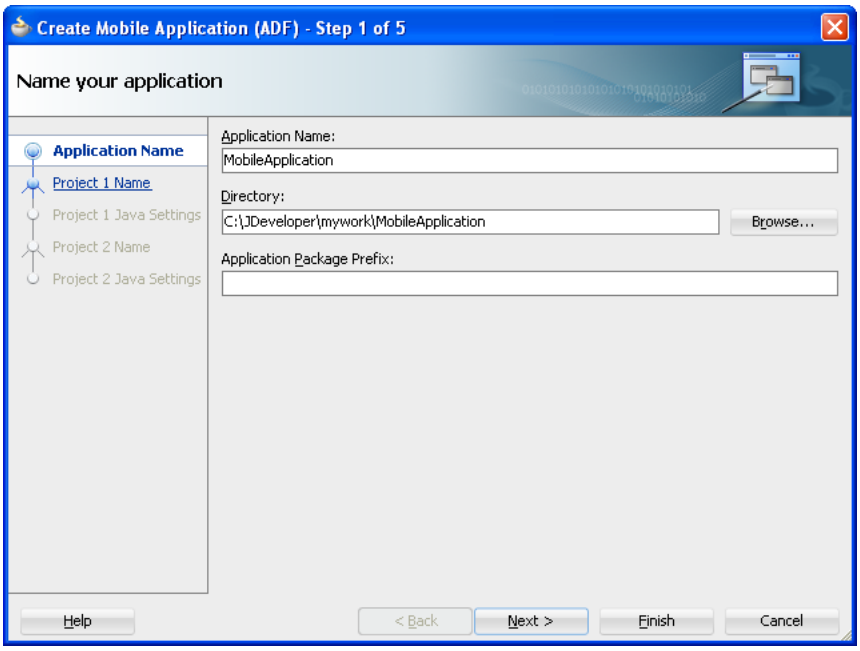
Figure 4–1 Selecting the ADF Mobile Application Template



2. In the **Application Name** field, enter a name for the application, such as *MobileApplication* in Figure 4–2. If needed, enter a new location for the project in the **Directory** field. Ensure that the application package is unique by entering a prefix for it in the **Application Package Prefix** field. Click **Next**.

   The application is the top-level structure of the ADF Mobile application. It organizes the different tiers of projects that you define in the subsequent pages of this wizard.

Figure 4–2 Naming the ADF Mobile Application



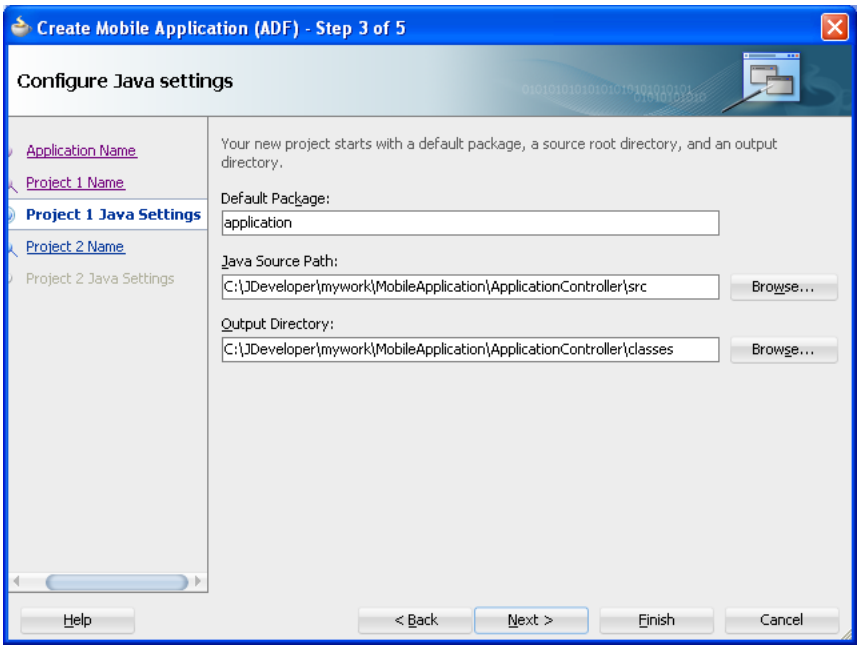
3. In the Project 1 Name page, change the name and location of the application controller project (if needed), as shown in Figure 4–3. Otherwise, accept the default name of the project, *ApplicationController*. This Project Feature window of the page lists the technologies available to the application controller project.

   This project stores all of the application-wide resources. For more information, see Table 4–1.

Figure 4–3 The Application Controller Project and Its Project Features

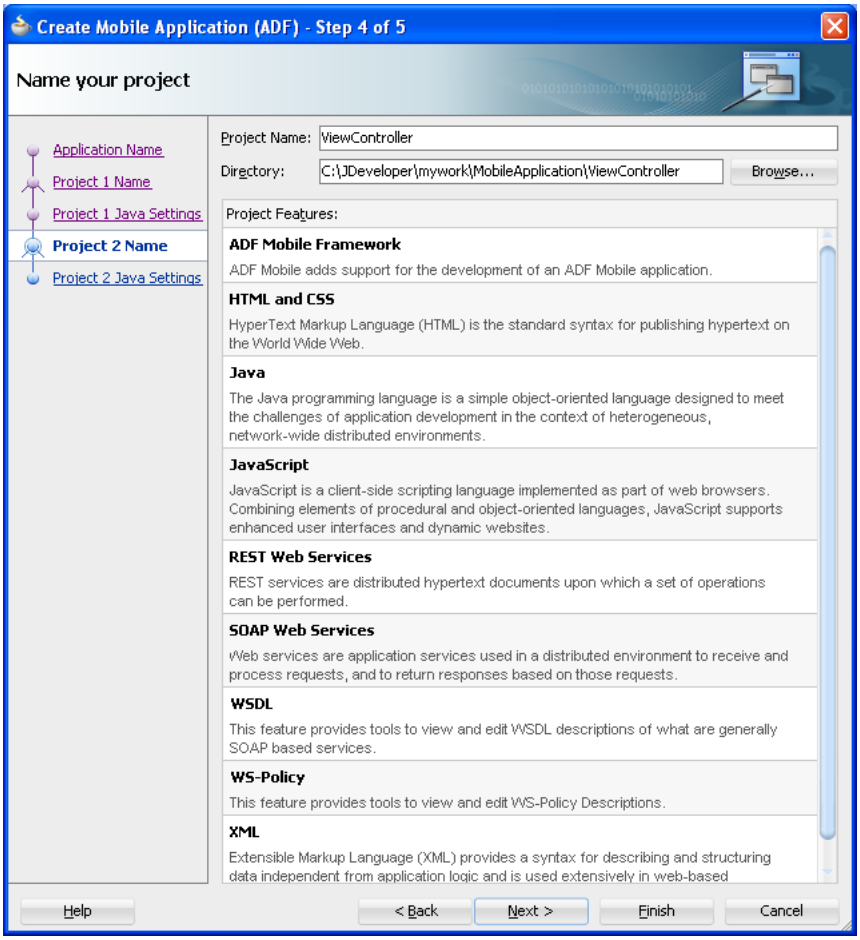
4. Click **Next** to the Java Settings page for the application controller project (Project 1 Java Settings, shown in Figure 4–4). Accept, or change, the default package name for the application controller project (`application`) and the location for the Java `SOURCEPATH` directory (`src`) and the Java output directory (`classes`).

Figure 4–4 Configuring the Java Settings for the Application Controller Project



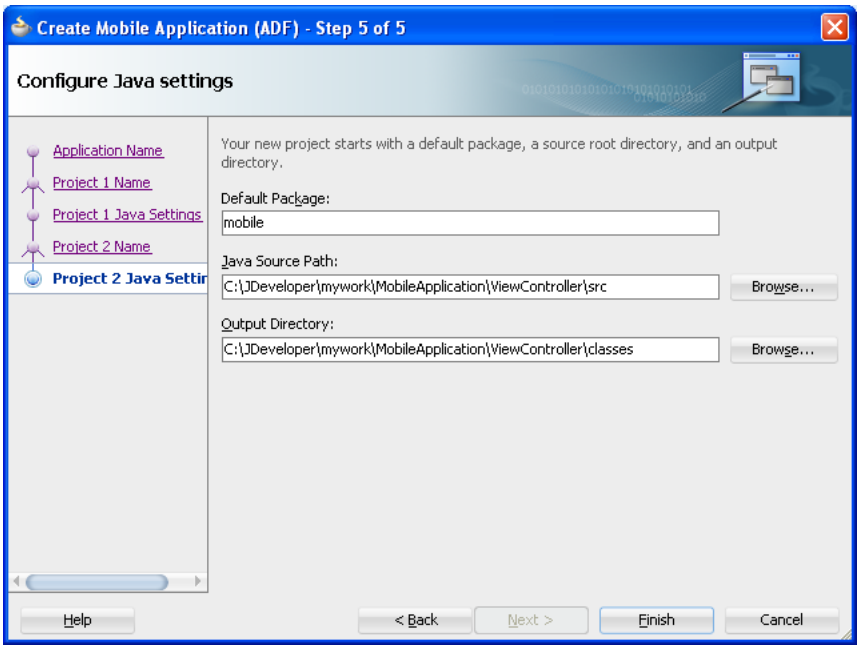
5. Click **Next** to traverse to the Project 2 Name page, where you can, if needed, rename the view controller project. The page's Project Features window, shown Figure 4–5, lists the technologies available to the ADF Mobile view controller project. For information on the artifacts created within view controller project, see Table 4–2.

Figure 4–5 The ADF Mobile View Controller Project



6. Click **Next**. In the Java settings page for the view controller project, shown in Figure 4–6, accept (or change) the default package name for the application controller project (`mobile`), the location for the project's Java `SOURCEPATH` directory (`src`) and the Java output directory (`classes`).

Figure 4–6 The Java Settings for the View Controller Project



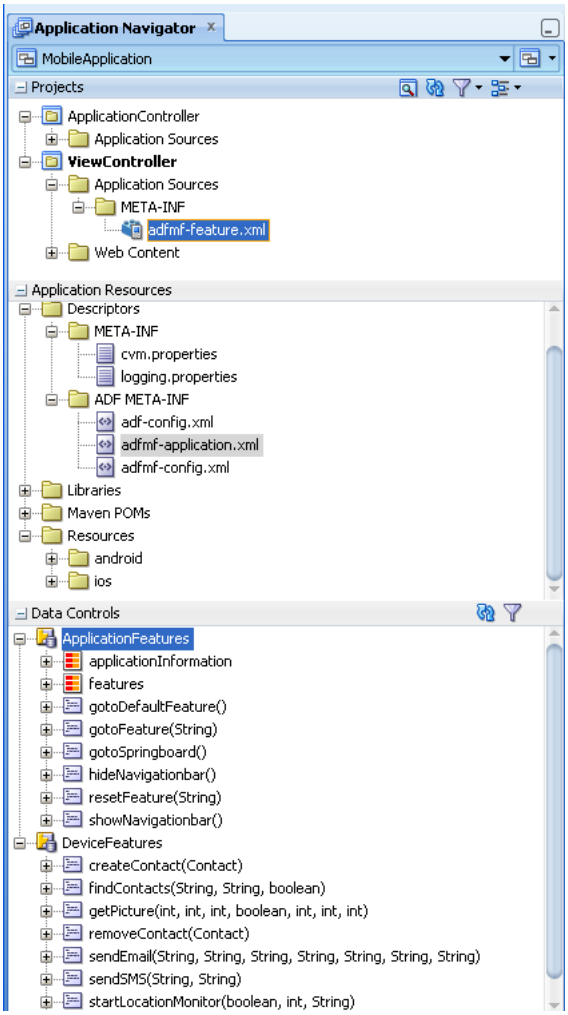
7. Click **Finish** to complete the creation of the ADF Mobile application and its projects.

   Tip: In addition to creating an ADF Mobile application following the above-mentioned steps, you can open the HelloWorld sample application (located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer) and view the artifacts that JDeveloper generates after you complete the application creation wizard.

4.2.2 What Happens When You Create an ADF Mobile Application

After you complete an ADF Mobile application project, JDeveloper adds application-level and project-level artifacts, which you access from the Application Navigator shown in Figure 4–7. In addition, JDeveloper creates the DeviceFeatures data control. The PhoneGap Java API is abstracted through this data control, thus enabling the application features implemented as ADF Mobile AMX to access various services embedded on the device. JDeveloper also creates the ApplicationFeatures data control, which enables you to build a springboard page. By dragging and dropping the operations provided by the DeviceFeatures data control into an ADF Mobile AMX page (which is described in Section 9.5, "Using the DeviceFeatures Data Control"), you add functions to manage the user contacts stored on the device, create and send both e-mail and SMS text messages, ascertain the location of the device, use the device's camera, and retrieve images stored in the device's file system.

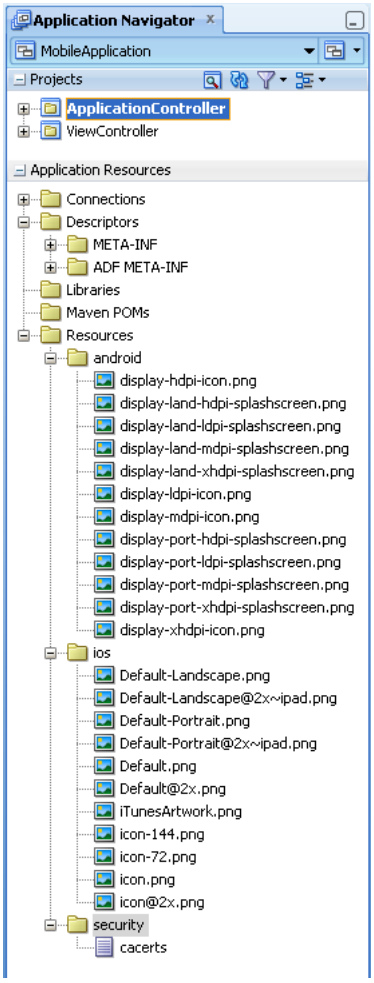
Figure 4–7 JDeveloper-Generated ADF Mobile Application Artifacts



4.2.2.1 About the Application-Level Resources

JDeveloper generates the files for ADF Mobile application in the application controller project. These files, described in Table 4–1, contain configuration files for describing the metadata of the of the ADF Mobile application. You access these files from the Application Resources pane of the Application Navigator, shown in Figure 4–8.

Figure 4–8 Mobile Application Artifacts Accessed from the Application Resources Pane



The application controller project, which houses the application-wide resources, provides the presentation layer of the ADF Mobile application in that it includes metadata files for configuring how the application will display on a mobile device. This project dictates the security for the ADF Mobile application and can include the application's login page, an application-wide resource. The application controller project is essentially a consumer of the view controller project, which defines the application features and their content. For more information, see Section 4.2.2.2, "About the View Controller Project."

Table 4–1 Mobile Application-Level Artifacts Accessed Through Application Resources

Artifact(s)	File Location	Description
`adfmf-application.xml`	*`application workspace_directory`*`\.adf\Meta-INF` For example: `JDevloper\mywork\Mobile Application\.adf\META-INF`	A stub XML application descriptor file that enables you to define the ADF Mobile application. Similar to the application descriptors for ADF Fusion Web applications, this file enables you to define the content for an application, its navigation behavior, and its user authentication requirements. For more information, see Section 5.2, "About the Mobile Application Configuration File."
`adfmf-config.xml`	*`application workspace_directory`*`\.adf\Meta-INF` For example: `JDeveloper\mywork\Mobile Application\.adf\META-INF`	Use to configure the default skin used for ADF Mobile applications. For more information, see Section 5.11, "Skinning ADF Mobile Applications."
Application images	*`application workspace_directory`*`\Application Resources\resources\ios` For example: `JDeveloper\mywork\Mobile Application\resources\ios`	A set of images required for the deployment of iOS and Android applications. These include PNG images for application icons and splash screens. Deployment to an iOS-powered device, such as an iPhone, requires a set of images in varying sizes. The default iOS images provided with the project include: ▪ images used when the device is in both landscape and portrait orientations ▪ images used for retina displays (that is, `icon.png` and `icon@2x.png`) ▪ an iPad image (`icon-72.png`) To override these images, see Section 16.2.4.3, "Adding a Custom Image to an iOS Application."

Table 4–1 (Cont.) Mobile Application-Level Artifacts Accessed Through Application

Artifact(s)	File Location	Description
`cacerts`	*application workspace_directory*`\Application Resources\resources\Security\cacerts` For example: `JDeveloper\mywork\Mobile Application\resources\Security\cacerts`	The `cacerts` certificate file, a system-wide keystore that identifies the CA certifcates to JVM 1.4. You can update this file using the Java keytool utility. You can create a custom certificate file using keytool as described in Section 17.5, "Adding Private Certificates." Any certificate file must reside within the `Security` directory.
`logging.properties`	*application workspace_directory*`\src\.META-INF\logging.properties` For example: `JDeveloper\mywork\Mobile Application\src\META-INF\logging.properties`	Enables you to set the application error logging, such as the logging level and logging console. For more information, see Section 18.5, "Using and Configuring Logging."

Table 4–1 (Cont.) Mobile Application-Level Artifacts Accessed Through Application

Artifact(s)	File Location	Description
`cvm.properties`	*application workspace_directory*`\src\.META-INF\cvm.properties` For example: `JDeveloper\mywork\Mobile Application\src\META-INF\cvm.properties`	The configuration file for the Java virtual machine, JVM 1.4. Use this file to configure the application startup and heap space allotment, as well as Java and JavaScript debugging options. For more information, see Section 18.3.4, "How to Enable Debugging of Java Code and JavaScript."
`adf-config.xml`	*application workspace_directory*`\Mobile Application\.adf\META-INF` For example: `JDeveloper\mywork\Mobile Application\.adf\META-INF`	Used to configure application-level settings, including the Configuration Service parameters. For more information, see the "adf-config.xml" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. See also Section 10.7, "Administering Web Services."
`connections.xml`	*application workspace_directory*`\Mobile Application\.adf\META-INF` For example: `JDeveloper\mywork\Application34\.adf\META-INF`	The repository for all of the connections defined in the ADF Mobile application. See also the "connections.xml" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Within the application controller project itself, shown in Figure 4–9, JDeveloper creates the following artifacts, listed in Table 4–2.

Figure 4–9 The Application Controller Project

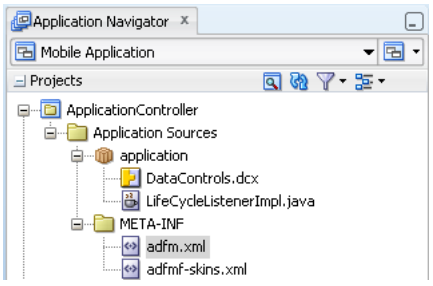


Table 4–2 Application Controller Artifacts

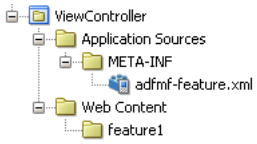
Artifact(s)	File Location	Description
`adfmf-skins.xml`	*application_workspace*`\ApplicationController\src\META-INF` For example: `JDeveloper\mywork\Mobile Application\ApplicationController\src\META-INF`	Defines the available skins and also enables you to define new skins. For more information, see Section 5.11, "Skinning ADF Mobile Applications."
`adfm.xml`	*application_workspace*`\ApplicationController\adfmsrc\META-INF` For example: `JDeveloper\mywork\Mobile Application\ApplicationController\adfmsrc\META-INF`	Maintains the paths (and relative paths) for the `.cpx`, `.dcx`, `.jpx`, and `.xcfg` files (registries of metadata). For more information, see the "adfm.xml" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
`DataControls.dcx`	*application_workspace*`\ApplicationController\adfmsrc\` For example: `JDeveloper\mywork\Mobile Application\ApplicationController\adfmsrc\`	The data controls registry. For information on using the DeviceFeature data control, which leverage the services of the device, see Section 9, "Using Bindings and Creating Data Controls." For information on the ApplicationFeatures data control, which enables you to create a springboard page that calls the embedded application features, see Section 5.4.5, "What You May Need to Know About Custom Springboard Application Features with ADF Mobile AMX Content." For more information see the "Oracle ADF Data Binding Files" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

4.2.2.2 About the View Controller Project

The view controller project (which is generated with the default name, `ViewController`, as illustrated in Figure 4–10) houses the resources for the application features. Unlike the application controller project described in Section 4.2.2.1, "About the Application-Level Resources," the view controller project's metadata files describe the resources at the application feature-level, in particular the various application features that can be aggregated into an ADF Mobile application so that they can display on a mobile device within the springboard of the ADF Mobile application itself or its navigation bar at runtime. Further, the application feature metadata files describe whether the application feature is comprised of HTML or ADF Mobile AMX pages. In addition, the view controller project can include these application pages as well as application feature-level resources, such as icon images to represent the application feature on the springboard and navigation bar defined for the ADF Mobile application.

The view controller project can be decoupled from the application controller project and deployed as an archive file for reuse in other mobile applications as described in Section 5.12, "Working with Feature Archive Files." In rare cases, an application controller project can consume more than one view controller project.

Figure 4–10 The View Controller Project



As shown in Table 4–3, these resources include the configuration file for application features called `adfmf-feature.xml`.

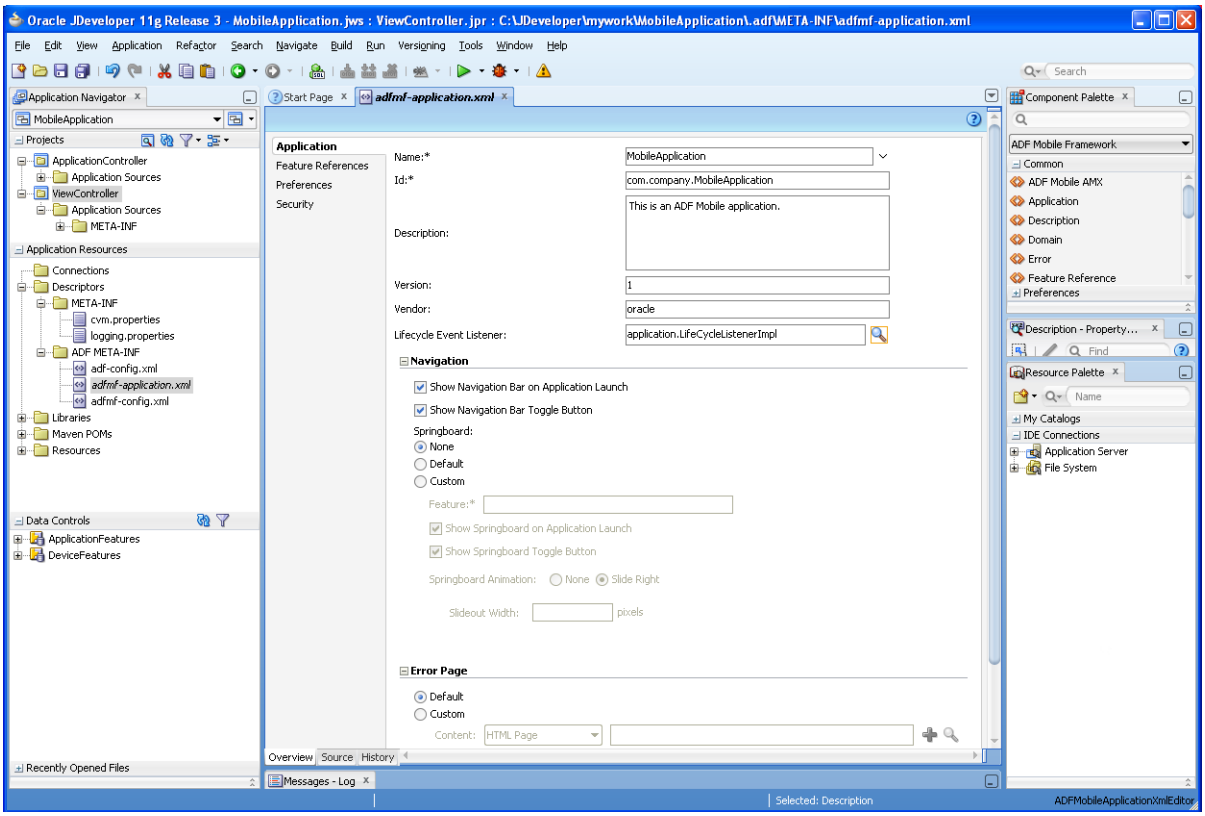
Table 4–3 View Controller Artifacts

Artifact(s)	File Location	Description
`adfmf-feature.xml`	*`application_workspace`*`\src\META_INF\adfmf-feature.xml` For example: `JDeveloper\mywork\Mobile Application\ViewController\src\META-INF`	A stub XML descriptor file that enables you to define application features. For more information, see Section 5.7, "About the Mobile Feature Application Configuration File." After you have configured the Mobile Preferences as described in Section 3.3.1, "How to Configure the Development Environment for Platforms and Form Factors," you can deploy this application using the default deployment profile settings. For more information, see Chapter 16, "Deploying ADF Mobile Applications."
Application-Specific Content	*`application_workspace`*`\ViewController\public_html` For example: `JDeveloper\mywork\Mobile Application\ViewController\public_html`	The application features defined in `adfmf-feature.xml` display in the `public_html` directory. Mobile contents can include ADF Mobile AMX pages, CSS files, and task flows. Any custom images that you add to an application feature must be located within this directory. For more information, see Section 5.9.2, "What You May Need to Know About Selecting External Resources."

4.2.3 What You May Need to Know About Editing ADF Mobile Applications and Application Features

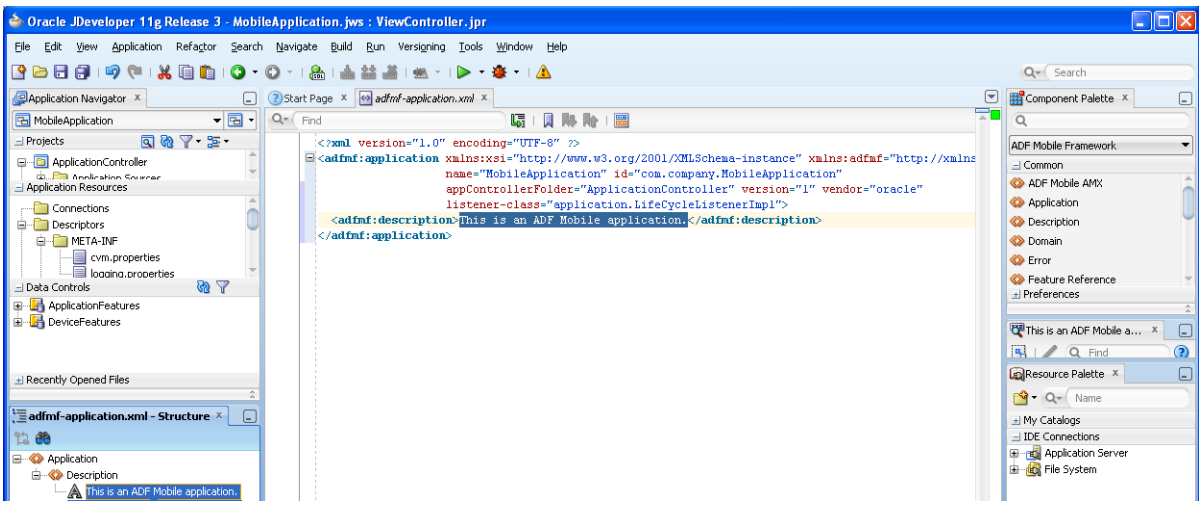
Creating an application results in the generation of the `adfmf-application.xml` file, which enables you to configure the mobile application and also the `adfmf-features.xml` file, which you use to add, remove, or edit the application features embedded within the mobile application. The ADF Mobile extension provides you with overview editors for both of these files, enabling you to declaratively change them. Figure 4–11 shows an example of the overview editor of the `adfmf-application.xml` file.

Figure 4–11 The Overview Editor of the Mobile Application



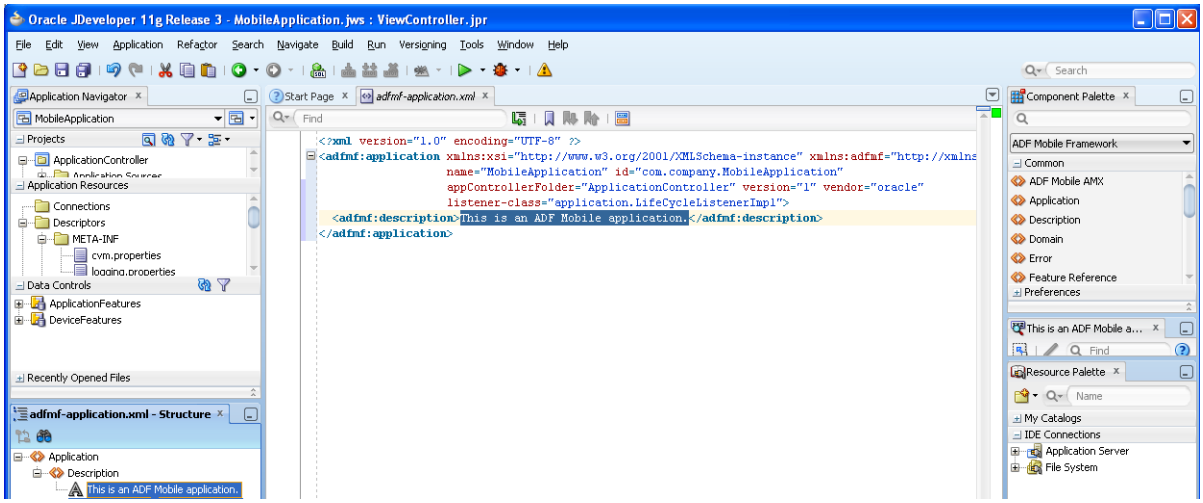
As shown in Figure 4–11, the `adfmf-application.xml` file is located in the Application Navigator in the **Application Resources** panel, under the **Descriptors** and **ADF META-INF** nodes. You can open this file by double-clicking it from this location. When you access this file, JDeveloper not only opens the associated overview editor, but also displays the pertinent page components in the component palette, which you can drag and drop into either the Source page of the editor or the Structure window, as shown in Figure 4–12. Section 5.2, "About the Mobile Application Configuration File" describes the components of the `adfmf-application.xml` page.

Figure 4–12 Using the Source Editor, Structure Window, and Properties Editor for the ADF Mobile Application



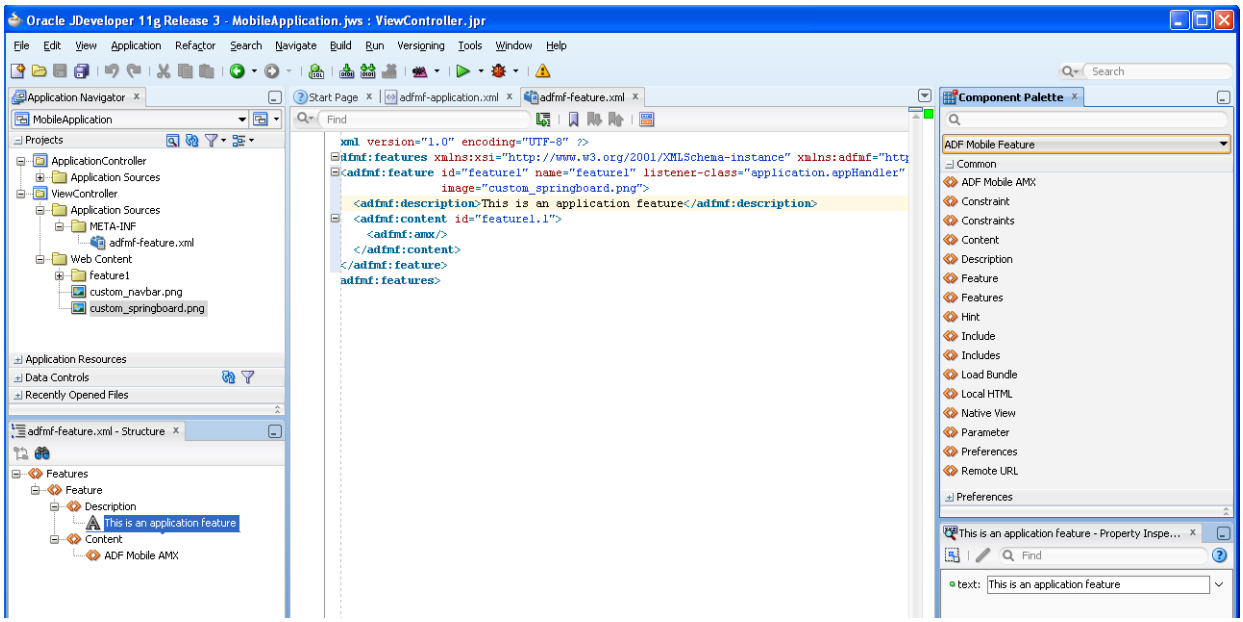
As illustrated in Figure 4–13, the `adfmf-feature.xml` configuration file is located in the Application Navigator in the **Project** panel under the **view controller** and **META-INF** nodes. You use this file to compose the content for the ADF Mobile application.

Figure 4–13 The Overview Editor for Application Features



Like the overview editor for the `adfmf-application.xml` file, JDeveloper presents the ADF Mobile components used for building the elements of the `adfmf-features.xml` configuration file, which are described in Section 5.7, "About the Mobile Feature Application Configuration File." You can use the Overview page or you can drag and drop components from the Component palette into the Structure Window or into the Source editor itself. When you select the `adfmf-feature.xml` file, JDeveloper populates the Component palette with ADF Mobile Feature components.

Figure 4–14 Using the Source Editor, Structure Window, and Component Palette for Application Features



4.2.4 Creating an Application Workspace for an ADF Mobile AMX Application

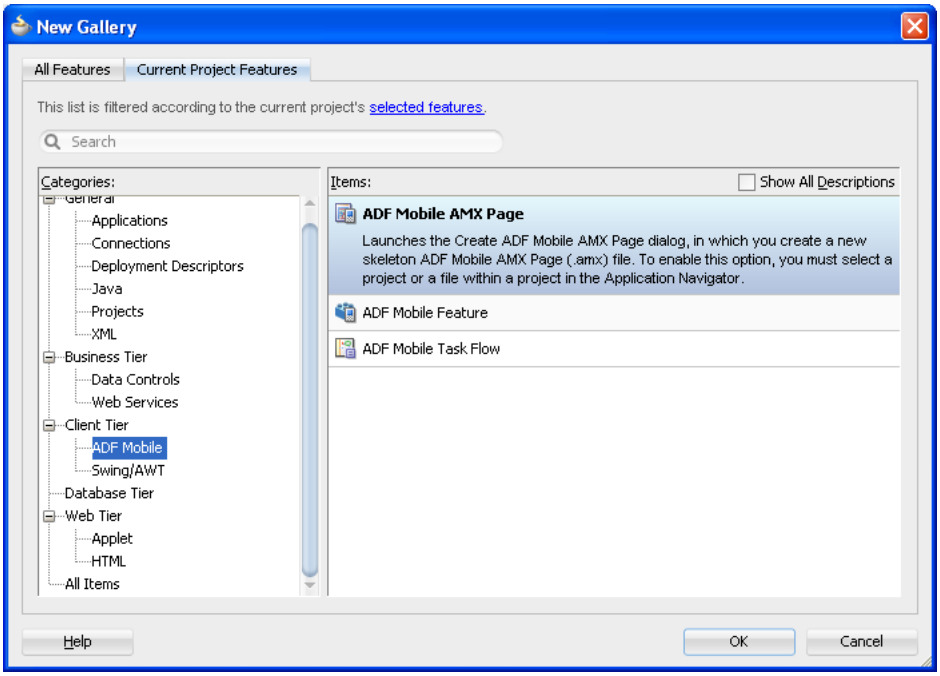
As described in Chapter 7, "Creating ADF Mobile AMX Pages," the ADF Mobile AMX components enable you to build pages that run identically to those authored in a platform-specific language. These pages may be created by the application assembler, who creates the ADF Mobile application and embeds application features within it, or they can be developed by another developer and then incorporated into the ADF Mobile application either as an application feature or as a resource to an ADF Mobile application.

The project in which you create the ADF Mobile AMX page determines if the page is used to deliver the content for a single application feature or be used as a resource to the entire ADF Mobile application. For example, a page created within the application controller project, as shown in Figure 4–17, would be used as an application-wide resource. An ADF Mobile AMX page created within a view controller project, on the other hand, would be used only to deliver content to an application feature.

An ADF Mobile task flow can likewise be used to deliver the content to an application feature. As shown in Figure 4–15, ADF Mobile provides wizards for adding ADF Mobile AMX pages, task flows, and application features.

To access these wizards, you first select a project within the Application Navigator and then choose **File** and then **New**. You select one of the wizards after selecting ADF Mobile within the Client tier.

Figure 4–15 Wizards for Creating Resources for Application Features



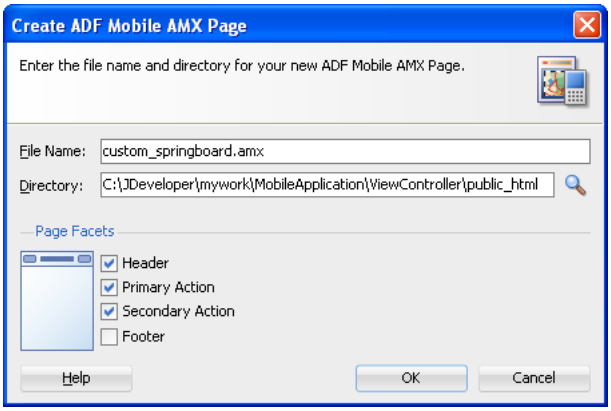
4.2.4.1 How to Create an ADF Mobile AMX Page

You can use the ADF Mobile AMX Page wizard to create AMX pages used as application feature content and separately as a resource to the ADF Mobile application. For more information on application feature content, see Section 5.9.1, "How to Define the Application Content."

To create an ADF Mobile AMX page as content for an application feature:

1. In the Application Navigator, right-click the view controller project.
2. Choose **File** and then **New**.
3. From the Client Tier node in the New Gallery, choose **ADF Mobile AMX Page** and then click **OK**.
4. Complete the Create ADF Mobile AMX Page dialog, shown in Figure 4–16, by entering a name in the **File Name field**. In the **Directory** field, enter the file location, which must be within the `public_html` folder of the view controller project.

Figure 4–16 Creating an ADF Mobile AMX Page in a View Controller Project



5. Select (or deselect) the Facets within the Panel Page that are used to create a header and footer. Click **OK**.

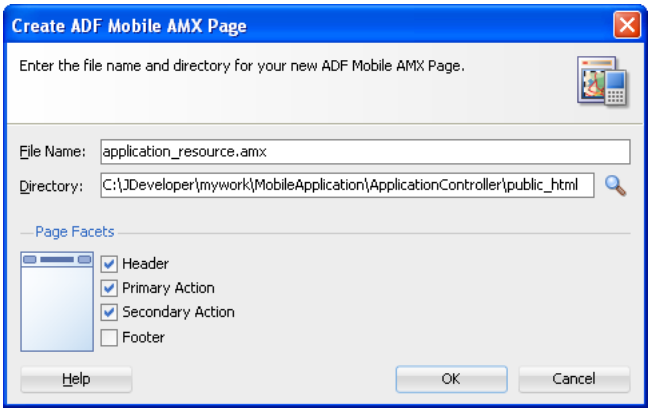
   For more information, see Section 8.2.2, "How to Use a Panel Page Component."

6. Build the ADF Mobile AMX page. For more information using the ADF Mobile AMX components, see Section 7.3.1.2, "Creating ADF Mobile AMX Pages." See also Section 5.9.1, "How to Define the Application Content."

To create an ADF Mobile AMX page as a resource to an ADF Mobile application:

1. In the Application Navigator, select the application controller project.
2. Choose **File** and then **New**.
3. From the Client Tier node in the New Gallery, select **ADF Mobile AMX Page**, and then click **OK**.
4. Complete the Create ADF Mobile AMX Page dialog, shown in Figure 4–17, by entering a name in the **File Name** field. In the **Directory** field, enter the file location, which must be within the `public_html` folder of the application controller project. Click **OK**.

Figure 4–17 Creating an ADF Mobile AMX Page in an Application Controller Project



5. Build the ADF Mobile AMX page. For more information, see Section 7.3.1.2, "Creating ADF Mobile AMX Pages."

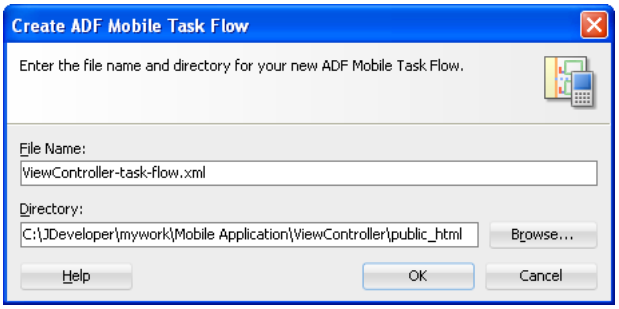
4.2.4.2 How to Create ADF Mobile Task Flows

You can deliver the content for an application feature as an ADF Mobile task flow.

To create an ADF Mobile Task Flow as content for an application feature:

1. In the Application Navigator, select the view controller project.
2. Choose **File** and then **New**.
3. From the Client Tier node in the New Gallery select **ADF Mobile Task Flow** and then click **OK**.
4. Complete the Create ADF Mobile AMX Page dialog, shown in Figure 4–18, by entering a name in the **File Name** field. In the **Directory** field, enter the file location, which must be within the `public_html` folder of the view controller project. Click **OK**.

Figure 4–18 Creating an ADF Mobile Task Flow in a View Controller Project



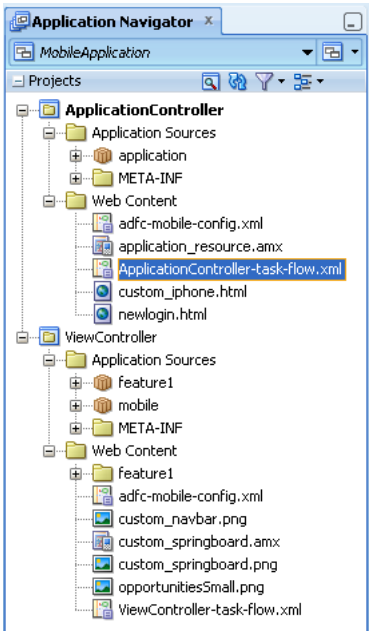
5. Build the task flow. See also Section 7.2, "Creating Task Flows."

4.2.4.3 What Happens When You Create ADF Mobile AMX Pages and Task Flows

JDeveloper places the ADF Mobile AMX pages and task flows in the **Web Content** node of the view controller project, as shown by `custom_springboard.amx` and `ViewController-task-flow.xml` (the default name for a task flow created within this project) in Figure 4–19. These artifacts are referenced in the `adfmf-feature.xml` file as described in Section 5.5, "Configuring the Application Features within a Mobile Application." Figure 4–19 also illustrates that other resources, such as customized

application splash screen (or launch) images and navigation bar images, are also housed in the **Web Content** node. For more information, refer to Table 4–3.

Figure 4–19 ADF Mobile AMX Pages and Task Flows within Application Controller and View Controller Projects



JDeveloper places the ADF Mobile AMX page and task flow as application resources to the ADF Mobile application in the **Web Content** node of the application controller project. As illustrated in Figure 4–19, the file for the ADF Mobile AMX page is called `application_resource.amx` and the task flow file is called `ApplicationController-task-flow.xml` (the default name).

Part III

Developing ADF Mobile Applications

Describes how to declaratively develop an ADF Mobile application using the overview editors as well as how a web page can invoke the PhoneGap APIs hosted within ADF Mobile.

Part III contains the following chapters:

5

Defining an ADF Mobile Application

This chapter describes using the overview editors to define the display behavior of the ADF Mobile application's springboard and navigation bar and how to designate content by embedding application features.

This chapter includes the following sections:

5.1 Introduction to Defining ADF Mobile Applications

An ADF Mobile application can have one or more view controller-type projects, each of which describes a set of features in an `adfmf-feature.xml` file. As described in Chapter 4, "Getting Started with ADF Mobile Application Development," ADF Mobile provides you with the `adfmf-application.xml` configuration file for the mobile application itself, and the `adfmf-feature.xml` file, which you use to define the content of the application. While you can manually change these files, ADF Mobile provides two overview editors that enable you to build these files declaratively.

5.1.1 Using the Overview Editor for adfmf-applications.xml

The overview editor enables you to configure the `adfmf-application.xml` file to describe a mobile application and its resources. Each page of the editor enables you to add or update the elements of the configuration file.

5.2 About the Mobile Application Configuration File

The `adfmf-application.xml` configuration file enables you to set the basic configuration of the ADF Mobile application by designating its display name, a unique application ID (to prevent naming collisions) and also by selecting the application features that will display on the application springboard (the equivalent of a home page on a smartphone). Further, this file enables you to create the user preferences pages for the mobile application. This file, which is generated by JDeveloper after you complete the application creation wizard as described in Section 4.2, "Creating an Application Workspace," is comprised of the elements listed in Table 5–2.

Table 5–1 Elements of the Application Descriptor File

Element	Description
`<adfmf:application>`	The root element of `adfmf-application.xml`.
`<adfmf:description>`	A description of the application.
`<adfmf:featureReference>`	A feature reference denotes which of the application features packaged in the FAR (Feature archive file) or defined in the `adfmf-feature.xml` file is relevant to the content of the mobile application. You define the character and content of ADF Mobile applications by selecting feature references. For more information about FARs, see Section 5.12, "Working with Feature Archive Files."
`<adfmf:preferences>`	Enables you to set the user preference options and behavior at the application level. You can also set how user preferences display and behave for the application features in the `adfmf-feature.xml` file. For more information, see Chapter 13, "Enabling User Preferences."
`<adfmf:login>`	Enables you to set the login page for an application feature. For more information, see Chapter 17, "ADF Mobile Application Security."
`<adfmf:navigation>`	Enables you to define the behavior of the navigation bar and the springboard. A springboard is a home page in which all of the application icons and labels for the embedded application features are organized in a List View. A springboard provides a top-level view of all of the applications available to a user, who can page through and select applications. For more information, see Section 5.4, "Configuring the Springboard and Navigation Bar Behavior."

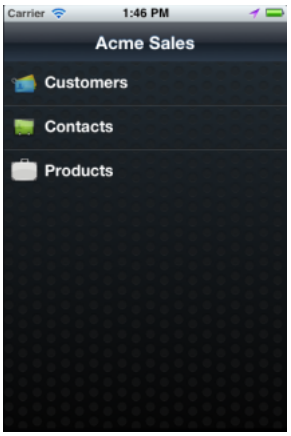
Example 5–1 illustrates the `adfmf-application.xml` file for an application called Acme Sales, a mobile application whose content includes a customer contacts application (`<adfmf:featureReference id="customers" showOnNavigationBar="true"/>`) which displays on the springboard, as shown in Figure 5–2.

Example 5–1 The adfmf-application.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                  xmlns:adfmf="http://xmlns.oracle.com/adf/mf"
                  name="Acme Sales"
                  id="com.company.AcmeSales"
```

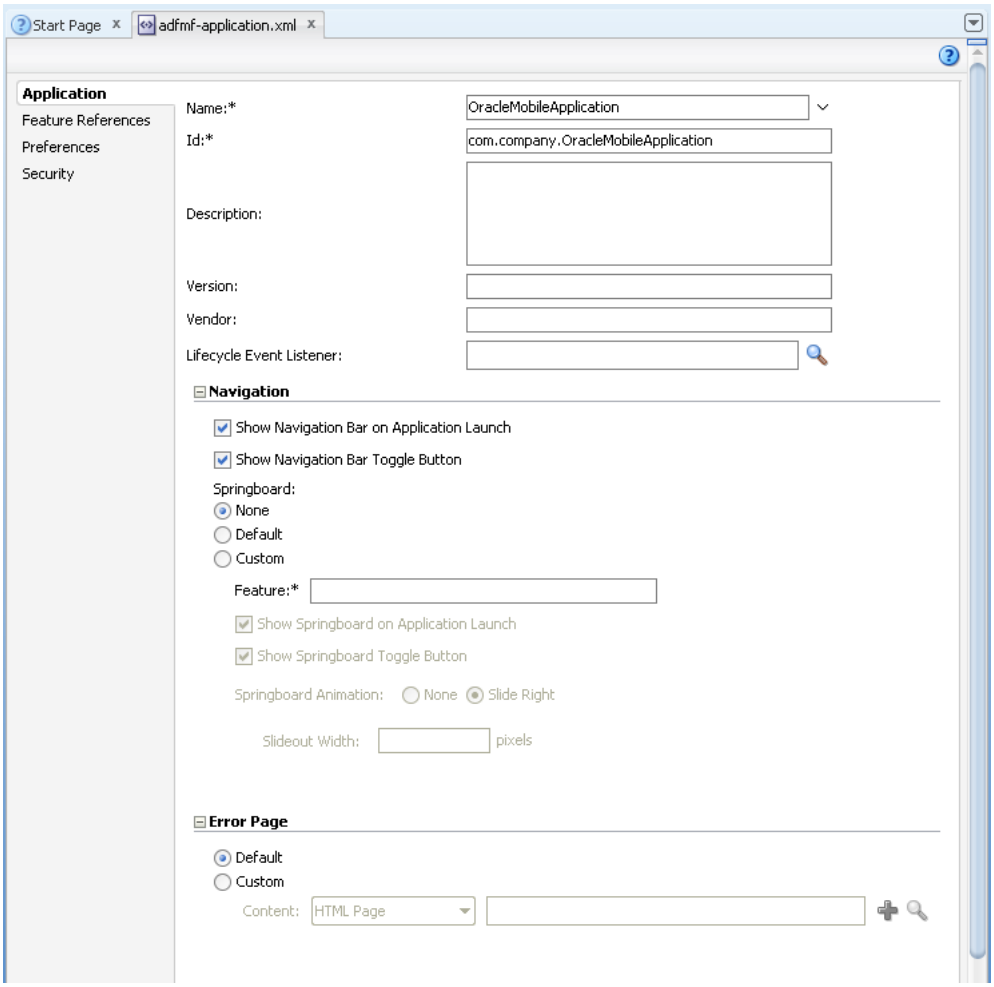
```
                  appControllerFolder="ApplicationController"
                  version="1.0"
                  vendor="Oracle"
                  listener-class="application.LifeCycleListenerImpl">
    <adfmf:description>Sample Application</adfmf:description>
<adfmf:featureReference id="Customers"
                        allowDeviceAccess="true"
                        showOnSpringboard="true"
                        showOnNavigationBar="true"/>
<adfmf:featureReference id="HCM"
                        allowDeviceAccess="true"
                        showOnSpringboard="true"
                        showOnNavigationBar="true"/>
<adfmf:featureReference id="PROD"
                        allowDeviceAccess="true"
                        showOnSpringboard="true"
                        showOnNavigationBar="true"/>
    <adfmf:preferences>
        <adfmf:preferenceGroup id="Root.Security" label="Security">
            <adfmf:preferenceText id="OracleSecurityProvider_serviceURL"
                                  label="Security URL"
                                  default="http://somecomputer.example.com:someaddress/
                                           idaas_rest/rest/tokenservice1/tokens/getprofile"/>

            <adfmf:preferenceText id="OracleADFMobile_UserName" label="User Name" default="sean"/>
            <adfmf:preferenceText id="OracleADFMobile_Password" label="Password" default="welcome1"
                                                                secret="true"/>
        </adfmf:preferenceGroup>
    </adfmf:preferences>
    <adfmf:navigation>
        <adfmf:springboard enabled="false"/>
        <adfmf:navigationBar enabled="true"/>
    </adfmf:navigation>
</adfmf:application>
```

Figure 5–1 Application Features Displaying in the Mobile Application's Springboard



You can modify these elements declaratively using the overview editor, shown in Figure 5–2, or manually using the Source editor. You can use either of these approaches interchangeably.

Figure 5–2 The Overview Editor for the adfmf-application.xml File



5.3 Setting the Basic Information for an ADF Mobile Application

The Application page enables you to set the name, application ID, and how the mobile application displays.

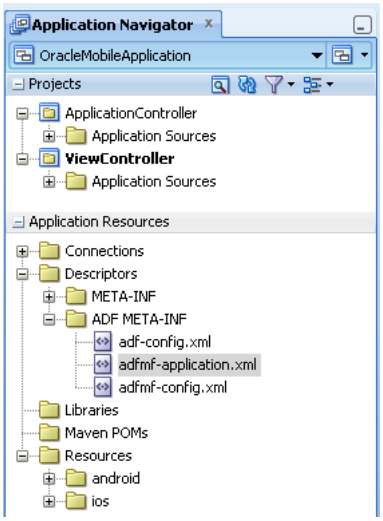
5.3.1 How to Set the ID and Display Behavior for a Mobile Application

The Application page of the overview editor, shown in Figure 5–2, enables you to name the mobile application and control the display of the mobile application within the springboard and navigation bar.

Before you begin:

Open the overview editor for the `adfmf-application.xml` file by double-clicking the `adfmf-application.xml` file (located in the **ADF META-INF** node of the Application Resources panel, as shown in Figure 5–3).

Figure 5–3 Selecting the adfmf-application.xml File in the Application Navigator.



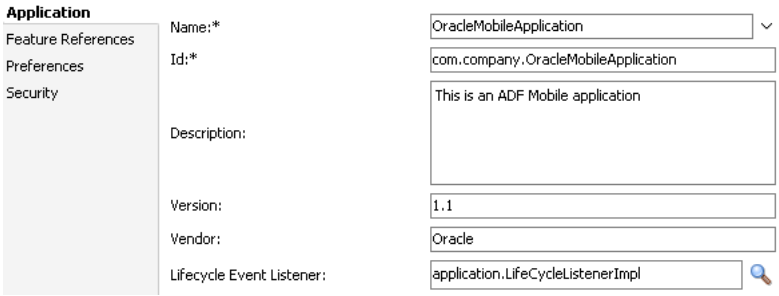
To set the basic information for a mobile application:

1. Choose the **Application** page and if needed, choose the Overview tab.

> **Note:** By default, the editor opens the Application page.

Figure 5–4 shows the portion of the application page where you define the basic information.

Figure 5–4 Setting the Basic Information for the ADF Mobile Application



2. Enter a display name for the application in the **Name** field. This attribute can be localized. For more information, see Section 5.10, "Working with Resource Bundles."

> **Note:** ADF Mobile uses the value entered in this field as the name for the iOS archive file (that is, the `.ipa` file) that it creates when you deploy the application to an iOS-powered device. For more information, see Section 16.2.4, "How to Create an iOS Deployment Profile."

3. Enter a unique ID in the **Id** field.

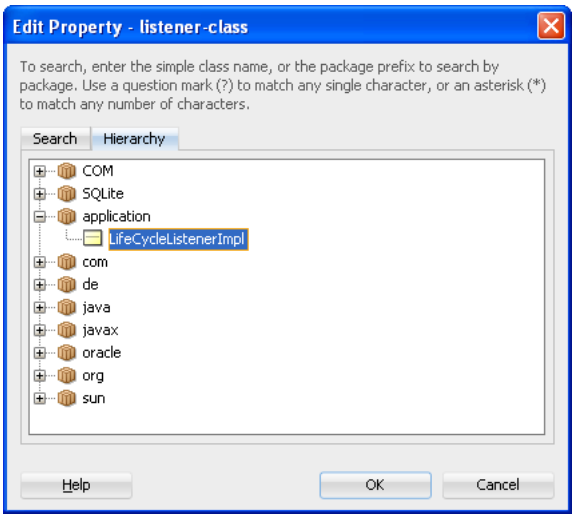
To avoid naming collisions, Android and iOS use reverse package names, such as *com.company.application*. JDeveloper prepends *com.company* as a reverse package to the application name, but you can overwrite this value with another as long as it is unique and adheres to the ID guidelines for both iOS- and Android-powered devices. For iOS application, see the "Creating and Configuring App IDs" section in *iOS Team Administration Guide* (available from the iOS Developer Library at `http://developer.apple.com/library/ios`). For Android, refer to the information provided about the Android manifest file available from the Android Developers website (`http://developer.android.com/guide/topics/manifest/manifest-intro.html`). You can overwrite this ID in the deployment profiles described in Section 16.2.3, "How to Create an Android Deployment Profile" and Section 16.2.4, "How to Create an iOS Deployment Profile."

Note: To ensure that an application deploys successfully to an Android-powered device or emulator, the ID must begin with a letter, not with a number. For example, an ID comprised of a wholly numeric value, such as 925090 (*com.company.925090*) will prevent the application from deploying. An ID that begins with letters, such as hello925090 (*com.company.hello925090*) will enable the deployment to succeed.

---

4. Enter a short, but detailed summary of the application that describes the application's purpose in the **Description** field.
5. Enter the version in the **Version** field.
6. Enter the name of the vendor who originated this application in the **Vendor** field.
7. Click **Browse** in the **Lifecycle Event Listener** field to invoke the Edit Property dialog, shown in Figure 5–5, to register the event listener that enables runtime calls for start, stop, activate, and deactivate events. You must enter the fully qualified class name of the event listener. This is an optional field.

Figure 5–5 Retrieving the Application Event Listener



The default application listener class is `application.LifeCycleListenerImpl`. ADF Mobile does not register this class by default, because it starts the JVM and therefore may not be preferable for

each ADF Mobile application. You must instead register this class manually using the Edit Property dialog, shown in Figure 5–5. After you close this dialog, JDeveloper updates `<adfmf:application>` element with a `listener-class` attribute, as illustrated in Example 5–2.

Example 5–2 Adding the listener-class Attribute

```
<adfmf:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:adfmf="http://xmlns.oracle.com/adf/mf"
                   name="OracleMobileApplication"
                   id="com.company.OracleMobileApplication"
                   appControllerFolder="ApplicationController"
                   listener-class="application.LifeCycleListenerImpl"
                   version="1.1"
                   vendor="Oracle">
   <adfmf:description>This is an ADF Mobile application</adfmf:description>
   <adfmf:featureReference id="feature1"/>
</adfmf:application>
```

For more information, see Section 5.6, "About Lifecycle Event Listeners."

5.4 Configuring the Springboard and Navigation Bar Behavior

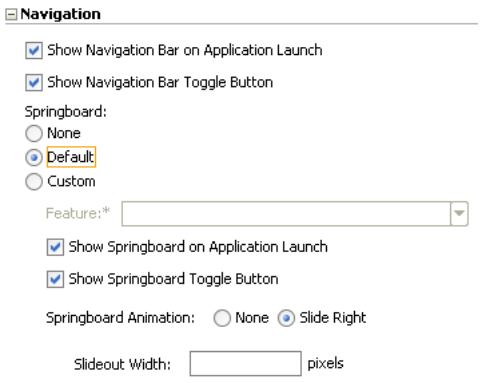
You can configure the ADF Mobile application to control the display behavior of the springboard and the navigation bar in the following ways:

- Hide or show the springboard and navigation bar to enable the optimal usage of the mobile device's real estate. These options override the default display behavior for the navigation bar, which is shown by default unless otherwise specified by the application feature.
- Enable the springboard to slide from the right. By default, the springboard does not occupy the entire display, but instead slides from the left, pushing the active content (which includes the navigation bar's Home button and application features) to the right.

5.4.1 How to Configure Application Navigation

The Navigation options of the Applications page, shown in Figure 5–6, enable you to hide or show the navigation bar, select the type of springboard used by the application, and define how the springboard reacts when users page through applications.

Figure 5–6 The Navigation Options of the Application Page



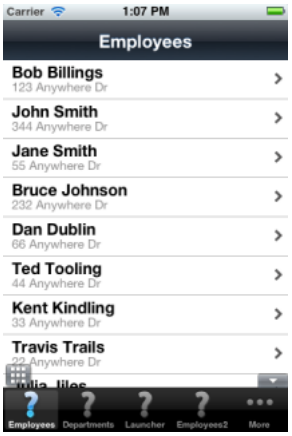
Before you begin:

You must select the Application page of the `adfmf-application.xml` overview editor.

To set the display behavior for the navigation bar:

1. Select **Show Navigation Bar on Application Launch** to enable the mobile application to display its navigation bar, rather than the springboard, by default after it is launched, as shown in Figure 5–7.

Figure 5–7 The Navigation Bar, Shown By Default (on iPhone)



If you clear this option, then you hide the navigation bar when the application starts, presenting the user with the springboard as the only means of navigation. Because the navigation bar serves the same purpose as the springboard, hiding it can, in some cases, remove redundant functionality.

2. Select **Show Navigation Bar Toggle Button** to hide the navigation bar when the content of a selected application feature is visible. Figure 5–8 illustrates this option, showing how the navigation bar illustrated in Figure 5–7 becomes hidden by the application feature content.

Figure 5–8 Hiding the Navigation Bar (on iPhone)

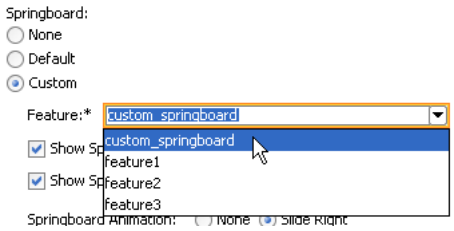
This option is selected by default; the navigation bar is shown by default if the show or hide state is not specified by the application feature.

To set the display behavior for the springboard:

1. Select the type of springboard (if any):
   - **None**—Select this option if the springboard should not be displayed in the application.
   - **Default**—Select to display the default springboard provided by ADF Mobile, which uses styles illustrated in Figure 5–1. The default springboard is implemented as an ADF Mobile AMX page. For more information, see Section 5.4.5, "What You May Need to Know About Custom Springboard Application Features with ADF Mobile AMX Content."
   - **Custom**—Select to use a customized springboard, which can be implemented either as an HTML page or as an ADF Mobile AMX page. This application is declared as an application feature in the `adfmf-feature.xml` file, which is located within a view controller project. For more information, see Section 5.8.1, "How to Define the Basic Information for the Application Feature." For information on enabling navigation within a customized springboard written in HTML, see Chapter 6, "Controlling the Display of Application Features."
     - **Feature**—Select the application feature used as a springboard, as shown in Figure 5–7.

> **Note:** A custom springboard application feature cannot display within the ADF Mobile application's navigation bar, nor can it display as one of the application features that display in its springboard. For more information, see Section 5.5.1, "How to Designate the Content for a Mobile Application."

Figure 5–9 Selecting an Application Feature as a Custom Springboard



2. Select **Show Springboard on Application Launch** to enable the mobile application to display the springboard to the end user after the mobile application has been launched. (This option is only available for the **Default** or **Custom** options.)
3. Select **Show Springboard Toggle Button** to enable the display of the springboard button, shown in Figure 5–10, that displays within an application feature. Figure 5–7, "The Navigation Bar, Shown By Default (on iPhone)" shows this button within the context of an application feature. This option is only available for the **Default** or **Custom** options.

Figure 5–10 The Springboard Toggle Button (Used on iPhones)

To set the slide out behavior for the springboard:

1. Select **Springboard Animation** and then choose **Slide Right**. The springboard occupies an area determined by the number of pixels (or the percent) entered for the **Slideout Width** option. If you select **None**, then the springboard cannot slide from the right (that is, ADF Mobile does not provide the animation to enable this action). The springboard takes the entire display area.

   Note: The slide out option is only applicable when you select either the **Custom** or **Default** springboard options.

2. Set the width (in pixels). The default width of a springboard on an iOS-powered device is 320 pixels. On Android-powered devices, the springboard occupies the entire screen by default, thereby taking up all of the available width.

   Note: If the springboard does not occupy the entire area of the display, then an active application feature occupies the remainder of the display. For more information, see Section 5.4.3, "What Happens When You Set the Animation for the Springboard."

5.4.2 What Happens When You Configure the Navigation Options

Setting the springboard and navigation bar options updates or adds elements to the `adfmf:application.xml` file's `<adfmf:navigation>` element. For example, selecting **None** results in the code updated with `<springboard enabled="false">` as illustrated in Example 5–3.

Example 5–3 Preventing the Displaying the Springboard

```
<adfmf:application>
  ...
  <adfmf:navigation>
     <adfmf:navigationBar enabled="true"/>
     <adfmf:springboard enabled="false"/>
  </adfmf:navigation>
</adfmf:application>
```

Tip: By default, the navigation bar is enabled, but the springboard is not. If you update the XML manually, you can enable the springboard as follows:

```
<adfmf:application>
  ...
<adfmf:navigation>
    <adfmf:springboard enabled="true"/>
  </adfmf:navigation>
  ...
</adfmf:application>
```

Example 5–4 illustrates how the `enabled` attribute is set to `true` when you select **Default**.

Note: Because the springboard fills the entire screen of the device, the navigation bar and the springboard do not appear simultaneously.

Example 5–4 Enabling the Display of the Default Springboard

```
<adfmf:application>
  ...
  <adfmf:navigation>
     <adfmf:navigationBar enabled="true"/>
     <adfmf:springboard enabled="true"/>
  </adfmf:navigation>
</adfmf:application>
```

If you select **Custom** and then select the application feature used as the springboard, the editor populates the `<adfmf:navigation>` element as illustrated in Example 5–5. The `id` attribute refers to an application feature defined in the `adfmf-feature.xml` file that is used as a custom springboard.

Example 5–5 Configuring a Custom Springboard

```
<adfmf:navigation>
    <adfmf:springboard enabled="true">
      <adfmf:springboardFeatureReference id="springboard"/>
    </adfmf:springboard>
  </adfmf:navigation>
```

5.4.3 What Happens When You Set the Animation for the Springboard

Example 5–6 shows the navigation block of the `adfmf-application.xml` file, where the springboard is set to slide out and occupy a specified area of the display (213 pixels).

Example 5–6 Configuring Springboard Animation

```
<adfmf:navigation>
   <adfmf:navigationBar enabled="true"
                        displayHideShowNavigationBarControl="true"/>
   <!-- default interpretation of width is pixels -->
   <adfmf:springboard enabled="true"
                      animation="slideright"
                      width="213"
                      showSpringbaordAtStartup="true"/>
</adfmf:navigation>
```

The following line disables the animation:

```
<adfmf:springboard enabled="true" animation="none"/>
```

The following line sets the springboard to occupy 100 pixels from the left of the display area and also enables the active application feature to occupy the remaining portion of the display:

```
<adfmf:springboard enabled="true" animation="slideright" width="100px"/>
```

In addition to the animation, Example 5–6 demonstrates the following:

- The use of the `showSpringboardAtStartup` attribute, which defines whether the springboard displays when the application starts. (By default, the springboard is displayed.)
- The use of the `navigationBar`'s `displayHideShowNavigationBarControl` attribute.

To prevent the springboard from displaying, set the `enabled` attribute to `false`.

5.4.4 What You May Need to Know About Custom Springboard Application Features with HTML Content

The default HTML springboard page provided by ADF Mobile uses the following technologies, which you may also want to include in a customized login page:

- Cascading Style Sheets (CSS)—Defines the colors and layout.
- JavaScript—The `<script>` tag embedded within the springboard page contains methods described in Chapter 6, "Controlling the Display of Application Features" that call PhoneGap APIs. In addition, the HTML page uses JavaScript to respond to the callbacks and to detect page swipes. When swipe events are detected, JavaScript enables the dynamic modification of the style sheets to animate the page motions.
- WebKit—Provides smooth animation of the icons during transitions between layouts as well as between different springboard pages. For more information on the WebKit rendering engine, see `http://www.webkit.org/`.

Springboards written in HTML are application features declared in the `adfmf-feature.xml` file and referenced in the `adfmf-application.xml` file.

5.4.5 What You May Need to Know About Custom Springboard Application Features with ADF Mobile AMX Content

Like their HTML counterparts, springboards written using ADF Mobile AMX are application features that are referenced by the ADF Mobile application, as described in Section 5.5.1, "How to Designate the Content for a Mobile Application."

> **Note:** A custom springboard page (authored in either HTML or ADF Mobile AMX) must reside within a view controller project which also contains the `adfmf-feature.xml` file. For more information, see Section 5.5.1, "How to Designate the Content for a Mobile Application."

The default springboard (`adfmf.default.springboard.jar`, located in *`jdev_install`*`\jdeveloper\jdev\extensions\oracle.adf.mobile\lib`) is an ADF Mobile AMX page that is bundled in a Feature Archive (FAR) JAR file and deployed with other FARs that are included in the ADF Mobile application. This JAR file includes all of the artifacts associated with a springboard, such as the `DataBindings.cpx` and `PageDef.xml` files. This file is only available after you select **Default** as the springboard option in the `adfmf-application.xml` file. Selecting this option also adds this FAR to the application classpath. For more information, see Section 16.5, "Deploying Feature Archive Files (FARs)."

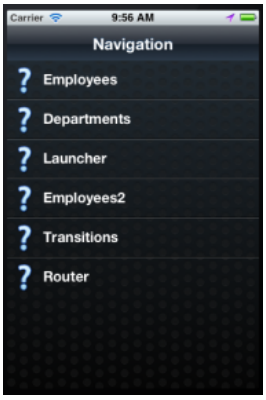
The default springboard (`springboard.amx`, illustrated in Example 5–7) is implemented as an ADF Mobile AMX application feature.

Example 5–7 The Default Springboard page, springboard.amx

```
<?xml version="1.0" encoding="UTF-8" ?>
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
   <amx:panelPage id="pp1">
      <amx:facet name="header">
         <amx:outputText value="#{bindings.name.inputValue}" id="ot3"/>
      </amx:facet>
      <amx:listView var="row"
                    value="#{bindings.features.collectionModel}"
                    fetchSize="#{bindings.features.rangeSize}"
                    id="lv1"
                    styleClass="amx-springboard">
         <amx:listItem showLinkIcon="false"
                       id="li1"
                       actionListener="#{bindings.gotoFeature.execute}">
            <amx:tableLayout id="tl1"
                             width="100%">
               <amx:rowLayout id="rl1">
                  <amx:cellFormat id="cf11"
                                  width="46px"
                                  halign="center">
                     <amx:image source="#{row.image}"
                                id="i1"
                                inlineStyle="width:36px;height:36px"/>
                  </amx:cellFormat>
                  <amx:cellFormat id="cf12"
                                  width="100%"
                                  height="43px">
                     <amx:outputText value="#{row.name}"
                                     id="ot2"/>
                  </amx:cellFormat>
               </amx:rowLayout>
            </amx:tableLayout>
               <amx:setPropertyListener from="#{row.id}"
                                        to="#{pageFlowScope.FeatureId}"/>
         </amx:listItem>
      </amx:listView>
   </amx:panelPage>
</amx:view>
```

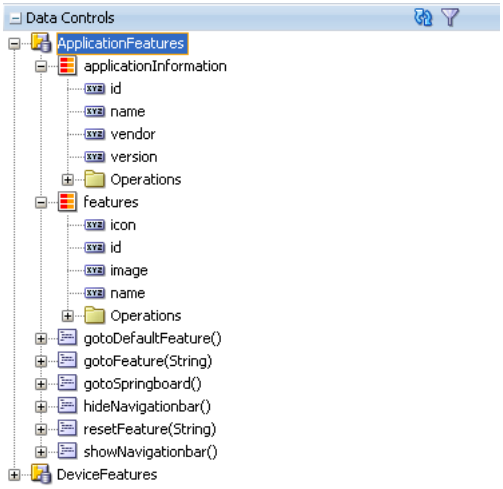
As shown in Figure 5–11, an ADF Mobile AMX file defines the springboard using a List View whose List Items are the ADF Mobile application's embedded application features. These application features, once deployed, are displayed by their names and associated icons. Selecting an application feature calls the `AdfmfContainerUtilities.gotoFeature` method. For more information, see Section 6.2.7, "gotoFeature." See also Section 8.2.7, "How to Use List View and List Item Components."

Figure 5–11 The Default Springboard



ADF Mobile provides the basic tools to create a custom springboard (or augment the default one) in the ApplicationFeatures data control. This data control, illustrated in Figure 5–12, enables you to declaratively build a springboard page using its data collections of attributes that describe both the ADF Mobile application and its application features.

Figure 5–12 The ApplicationFeatures Data Control



The methods of the ApplicationFeatures data control enable you to add navigation functions. These `adfmf.containerUtilities` methods are described in Table 5–2. For more information, see Section 6.2, "The ADF Mobile Container Utilities API." See also Chapter 9, "Using Bindings and Creating Data Controls."

Table 5–2 ApplicationFeature Methods

Method	Description
`gotoDefaultFeature`	Navigates to default application feature.
`gotoFeature`	Navigates to a specific application as designated by the parameter that is passed to this method.
`gotoSpringboard`	Navigates to the springboard.
`hideNavigationbar`	Hides the navigation bar.
`showNavigationbar`	Displays the navigation bar (if it is hidden).

Table 5–2 (Cont.) ApplicationFeature Methods

Method	Description
`resetFeature`	Resets the application feature that is designated by the parameter passed to this method.

5.5 Configuring the Application Features within a Mobile Application

Each ADF Mobile application must have at least one application feature.

5.5.1 How to Designate the Content for a Mobile Application

Figure 5–13 shows the Feature References page, which enables you to build the content for the mobile application. As noted in Section 5.2, "About the Mobile Application Configuration File," the mobile application descriptor file's `<adfmf:featureReference>` element designates these application features.

Example 5–8 Designating Feature References the in the adfmf-application.xml File

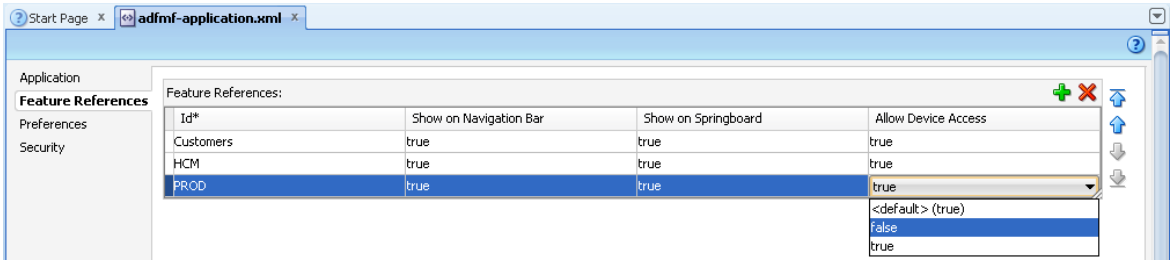
```
<adfmf:featureReference id="Customers"
                        allowDeviceAccess="true"
                        showOnSpringboard="true"
                        showOnNavigationBar="true"/>
<adfmf:featureReference id="HCM"
                        allowDeviceAccess="true"
                        showOnSpringboard="true"
                        showOnNavigationBar="true"/>
<adfmf:featureReference id="PROD"
                        allowDeviceAccess="true"
                        showOnSpringboard="true"
                        showOnNavigationBar="true"/>
```

Example 5–8 shows some of the defined feature references and their associated attributes. The overview editor displays these feature references in the Feature References table. Figure 5–13 shows the defined feature references for HCM and PROD that represent the Customers and Products application features, respectively. Using this page, you enter the references to the application feature and set its display within the mobile application's springboard, as shown in Figure 5–1.

In addition, the page enables you to set the order in which the application features display on the navigation bar and springboard.

Note: Because building a mobile application is an iterative process, you can add, delete or update feature references as new FARs become available (and new application features are added to the `adfmf-feature.xml` file).

Figure 5–13 Designating Application Features Using the Feature References Page



Before you begin:

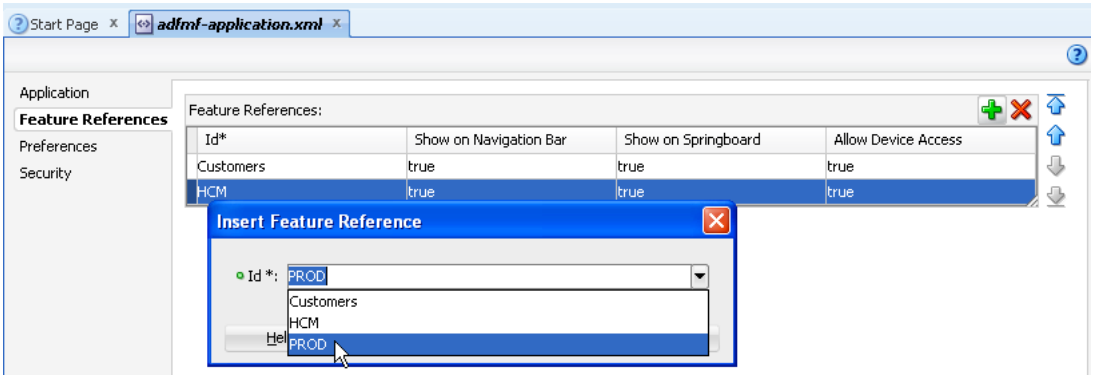
You must have application features configured in the `adfmf-feature.xml` file, as described in Section 5.8, "Setting the Basic Configuration for the Application Feature." In addition, you must open the `adfmf-application.xml` overview editor.

To designate feature references:

1. Click the **Feature References** page of the `adfmf-application.xml` overview editor.
2. Click **Add**.
3. In the Insert Feature Reference dialog, select the ID of the application feature from the dropdown list, as shown in Figure 5–14.

   This dialog lists all of the application features described in the `<adfmf:feature>` elements of the `adfmf-feature.xml` file Using this dialog ensures that the `id` attributes of both the `<adfmf:featureReference>` and `<adfmf:feature>` elements match. See also Section 5.5.2, "What You May Need to Know About Feature Reference IDs and Feature IDs."

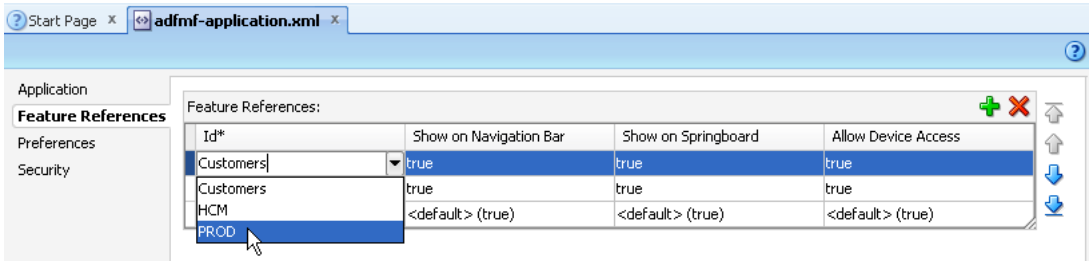
Figure 5–14 Selecting the Content for the Mobile Application



4. If needed, use the up- and down-arrows shown in Figure 5–14 to arrange the display order of the feature references, or use the dropdown list in rows of the Id column, shown in Figure 5–15, to reorder the feature references.

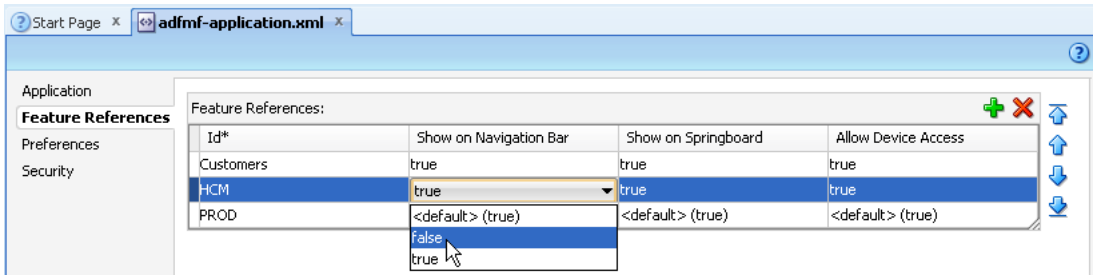
> **Note:** The top-most ID in the Feature References table is the first application feature to display within the ADF Mobile application. See, for example, the Employees application feature illustrated in Figure 5–7.

Figure 5–15 Reordering Feature References



5. Set the springboard and navigation bar display behavior for the application feature by selecting `true` or `false` from the dropdown lists in the rows of the **Show on Navigation Bar** and **Show on Springboard** columns. Figure 5–16 shows selecting these options to prevent an application feature from displaying in the navigation bar.

Figure 5–16 Changing the Navigation Options



By default, display is enabled (that is, `true`) for the springboard and the navigation bar. If both the navigation bar and springboard attributes are set to `false`, then the application feature only displays if it is in the first position. See also Section 6.2.12, "hideNavigationBar" and Section 6.2.13, "showNavigationBar."

Note: Because springboard applications do not display on the navigation bar or within the springboard of an ADF Mobile application, **Show on Navigation Bar** and **Show on Springboard** must both be set to `false` for feature references for custom springboard application features. See also Section 5.4.5, "What You May Need to Know About Custom Springboard Application Features with ADF Mobile AMX Content."

---

6. To enable a domain configured within the `connections.xml` file to access the device services enabled by the ADF Mobile PhoneGap API, select `true` in the **Allow Device Access** column. By default, all application features allow such access. To prevent domains from accessing device services, select `false`. For more information, see Chapter 12, "Implementing Application Features as Remote URLs."

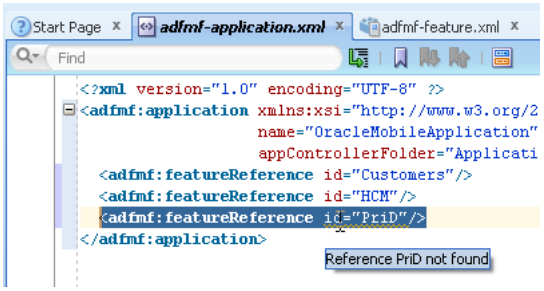
5.5.2 What You May Need to Know About Feature Reference IDs and Feature IDs

An ADF Mobile application can contain many projects and can therefore also include multiple `adfmf-feature.xml` files. In the application configuration file, a feature reference relates to a feature described by an `adfmf-feature.xml` file. The ID of an `<adfmf:featureReference>` identifies where the corresponding application

feature is defined. In other words, the `id` attributes for `<adfmf:featureReference>` elements in the `adfmf-application.xml` file must be the same as the `id` attribute defined for `<adfmf:feature>` element in `adfmf-feature.xml`. For example, `<adfmf:featureReference id="customers">` in `adfmf-application.xml` is defined by `<adfmf:feature id="customers">` in `adfmf-feature.xml`. JDeveloper audits the references between the application and feature descriptor files in the same mobile application, and warns you when it finds discrepancies. As shown in Figure 5–17, JDeveloper highlights the mismatch within the code between a feature reference and the features declared in the features application descriptor file with a wavy underline and a *Reference not Found* warning, which in this case is a feature reference ID defined as `PRiD` rather than the correct `PROD`. For more information on JDeveloper's syntax auditing, see the "Auditing and Profiling Applications" chapter in *Oracle Fusion Middleware User's Guide for Oracle JDeveloper*.

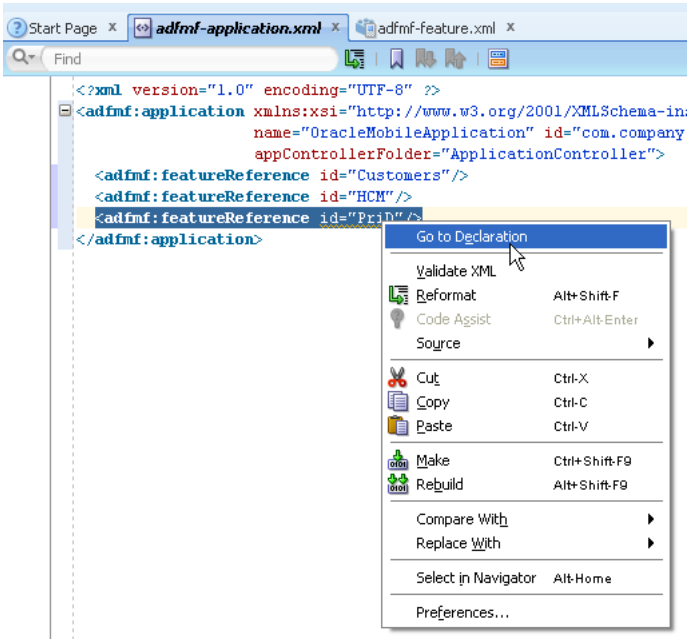
Note: The feature IDs must be unique across applications. Since the application features can be reused, use proper naming conventions to ensure that feature IDs are unique.

Figure 5–17 Auditing id Attributes



If the ID for an `<adfmf:featureReference>` is defined in the `adfmf-feature.xml` file, you can use **Go To Declaration** in the context menu, as shown in Figure 5–18, to traverse from the `id` of a `<adfmf:featureReference>` in the `adfmf-application.xml` file to the corresponding `id` of the `<adfmf:feature>` in the `adfmf-feature.xml` file.

Figure 5–18 Finding Application Feature Declarations in the adfmf-feature.xml File



5.6 About Lifecycle Event Listeners

Within the ADF Mobile runtime, classes implement various `LifeCycleListener` methods to communicate event notifications sent from the native operating system frameworks to the JVM. These event notifications describe various states (starting, stopping, or hibernating) for both the mobile application and its embedded application features. ADF Mobile invokes the class functions to the JVM using a generic invoke message.

The overview editors for both the `adfmf-application.xml` and `adfmf-feature.xml` files enable you to declaratively add a lifecycle listener class that ADF Mobile calls when events occur. After you enter a fully qualified class name (including the package) using the Class and Package Browser in the overview editor's LifeCycle Event Listener Class field, JDeveloper populates the XML page with the `listener-class` attribute. For ADF Mobile applications, this attribute is included in the `<adfmf:application>` element, as shown in Example 5–9.

Example 5–9 The listener-class Attribute in the adfmf-application.xml File

```
<adfmf:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:adfmf="http://xmlns.oracle.com/jdev/adfmf"
                   id="app-20110308"
                   name="#{strings.Title}"
                   vendor="Oracle"  version="1.0"
                   listener-class="oracle.adfmf.application.AppListener">

...

</adfmf:application>
```

Within the `adfmf-feature.xml` file, the `listener-class` attribute is contained within the `<adfmf:feature>` attribute, as shown in Example 5–10.

Example 5–10 The listener-class Attribute in the adfmf-feature.xml File

```
...

   <adfmf:feature id="mycompany.phonelist"
                  name="Phone List"
                  icon="mycompany.phonelist/PhoneIcon.png"
                  listener-class="oracle.adfmf.feature.LifeCycleListener">
                  image="mycompany.phonelist/PhoneImage.png"
      <adfmf:content id="general">

         <adfmf:amx file="mycompany.phonelist/Phone.amx"/>
      </adfmf:content>

   </adfmf:feature>

...
```

The LifeCycle Events sample application provides an example of declaring the event listener class in both the `adfmf-application.xml` and `adfmf-feature.xml` files. This sample application is in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

jdev_install`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples`

5.6.1 Events in ADF Mobile Applications

Lifecycle listener classes for ADF Mobile applications must implement the `start`, `stop`, `activate`, and `deactive` methods of the `LifeCycleListener` interface, illustrated in Example 5–11.

Example 5–11 The LifeCycleListener Interface for ADF Mobile Applications

```
package oracle.adfmf.application;

public interface LifeCycleListener
{
   void start();
   void stop();
   void activate();
   void deactivate();
}
```

The `AppListener` class, shown in Example 5–12, uses the `LifeCycleListener` method calls for starting or stopping events as well as those used when the application is about to hibernate (`deactivate`) or return from hibernating (`activate`).

Example 5–12 Implementation of the LifeCycleListener Interface

```
package some.package;
import oracle.adfmf.application.LifeCycleListener;

public class AppListener implements LifeCycleListener
{
    public AppListener()
    {
        super();
    }
    public void start()
    {
```

```
        // Perform application startup tasks...
    }
    public void stop()
    {
        // Perform tasks to stop the application...
    }
    public void activate()
    {
        // Perform appplication activation tasks...
    }
    public void deactivate()
    {
        // Perform application deactivation tasks...
    }
}
```

Tip: For an example of implementing the `LifeCycleListenerImpl.java` interface at the application level, see the `AppHandler.java` file in the LifeCycleEvents sample application. This file is located in the **application** node within the application controller project's Application Sources folder.

5.6.2 Timing for Mobile Application Events

ADF Mobile calls application lifecycle events at specific times during the ADF Mobile application's startup, shutdown, and hibernation. Table 5–3 describes when these events are fired and also notes their Objective-C counterparts.

Table 5–3 ADF Mobile LIfecycle Events

ADF Mobile Application Event	Timing	Relation to iOS Application Delegate Methods
`start`	Called after the ADF Mobile application has completely loaded the application features and immediately before presenting the user with the initial application feature or the springboard.	This event does not correspond to a specific application delegate method. It is called after the `adfmf-application.xml` and `adfmf-feature.xml` files have loaded, just prior to hiding the splash screen.
`stop`	Called as the ADF Mobile application begins its shutdown.	This is called in the `applicationWillTerminate:` method on the application delegate.
`activate`	Called as the ADF Mobile application activates from being situated in the background.	This is called in the `applicationDidBecomeActive:` method on the application delegate.
`deactivate`	Called as the ADF Mobile application deactivates and moves into the background.	This is called in the `applicationWillResignActive:` method on the application delegate.

5.6.3 About Application Feature Lifecycle Listener Classes

An ADF Mobile application feature lifecycle listener class, such as `FeatureListenerPhoneList` illustrated in Example 5–14, must implement the `activate` and `deactivate` methods of the `LifeCycleListener` interface, shown in Example 5–13, to hide the application feature when it hibernates or to display it when it returns from hibernating.

Example 5–13 Application Feature LifeCycleListener Interface

```
package oracle.adfmf.feature;
public interface LifeCycleListner
{
   void activate();
   void deactivate();
}
```

Example 5–14 illustrates a class called `FeatureListenerPhoneList` that uses the `activate` and `deactive` functions by implementing the `LifeCycleListener` to show and hide the application feature as described in Table 5–4.

Example 5–14 Implementing the Application Feature Lifecycle

```
package some.package;

import oracle.adfmf.feature.LifeCycleListener;

public class FeatureListenerPhoneList implements LifeCycleListener
{
    public FeatureListenerPhoneList()
    {
        super();
    }
    public void activate()
    {
        // Perform application activation tasks...
    }
    public void deactivate()
    {
        // Perform application deactivation tasks...
    }

}
```

Tip: The LifeCycle Events sample application provides an example of using the `LifeCycleListener` interface at the feature application level through the `Feature1Handler.java` and `Feature2Handler.java` files. These files are located within the view controller project's Application Sources folder.

5.6.4 Timing for Activate and Deactivate Events in the Application Feature Application Lifecycle

Table 5–4 describes when the activate and deactivate events are fired for an application feature.

Table 5–4 The activate and deactivate Events

Event	Timing
`activate`	Called immediately before the display of the application feature to end users.
`deactivate`	Called after the application feature has been hidden from the end user.

5.7 About the Mobile Feature Application Configuration File

The `adfmf-feature.xml` file, an example of which is illustrated in Example 5–15, enables you to configure the actual mobile application features that are referenced by the `<adfmf:featureReference>` element in the corresponding `adfmf-application.xml` file.

Example 5–15 The adfmf-feature.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:adfmf="http://xmlns.oracle.com/adf/mf">
    <adfmf:feature id="PROD"
                   name="Products"
                   icon="images/logo.png">
        <adfmf:constraints>
        </adfmf:constraints>
        <adfmf:description>ADF Products</adfmf:description>
        <adfmf:content id="PROD.1">
            <adfmf:amx file="PROD/Products_page.amx"/>
        </adfmf:content>
    </adfmf:feature>
    <adfmf:feature id="HCM"
                   name="Contents"
                   icon="images/directory.png">
        <adfmf:description>People Finder</adfmf:description>
        <adfmf:content id="HCM.1">
            <adfmf:remoteURL connection="Connections1"/>
        </adfmf:content>
    </adfmf:feature>
    <adfmf:feature id="Customers"
                   name="Customers"
                   icon="images/people.png">
        <adfmf:constraints>
            <adfmf:constraint property="user.roles"
                              operator="equal"
                              value="field sales"/>
            <adfmf:constraint property="user.roles"
                              operator="not"
                              value="consultant"/>
        </adfmf:constraints>
        <adfmf:description>Customers</adfmf:description>
        <adfmf:content id="Customers.2">
            <adfmf:localHTML url="Customers/customers_page.html"/>
        </adfmf:content>
    </adfmf:feature>
</adfmf:features>
```

By defining the elements of the `adfmf-feature.xml` file, you set the behavior for the application features by, in turn, defining the child elements of the `<Feature>`

element, the top-most element in the XML file under the root element, `<adfmf:features>`. The `<Feature>` element itself describes the basic information of the application feature, including its name, version, and whether or not it participates in security. For the latter, see Chapter 17, "ADF Mobile Application Security." The child elements of the `<Feature>` elements are listed in Table 5–5. Like the overview editor for the `adfmf-application.xml` descriptor file, you can update this file with these elements (or edit them) declaratively using the overview editor for the `adfmf-feature.xml` file, described in Section 5.8, "Setting the Basic Configuration for the Application Feature."

Table 5–5 Child Elements of <Feature> Element

Element	Description
`<adfmf:content>`	Describes how to implement the content of an application feature. The content of an application feature can be delivered as ADF Mobile AMX pages, locally stored HTML pages, or remotely served web applications. For more information on designating content as a web application, see Chapter 12, "Implementing Application Features as Remote URLs."
`<adfmf:constraint>`	Determines whether a given application feature can be displayed in the application runtime. Constraints can be used to allow or prevent the use of an application feature based on such criteria as user roles or device properties. For more information, see Chapter 14, "Setting Constraints."

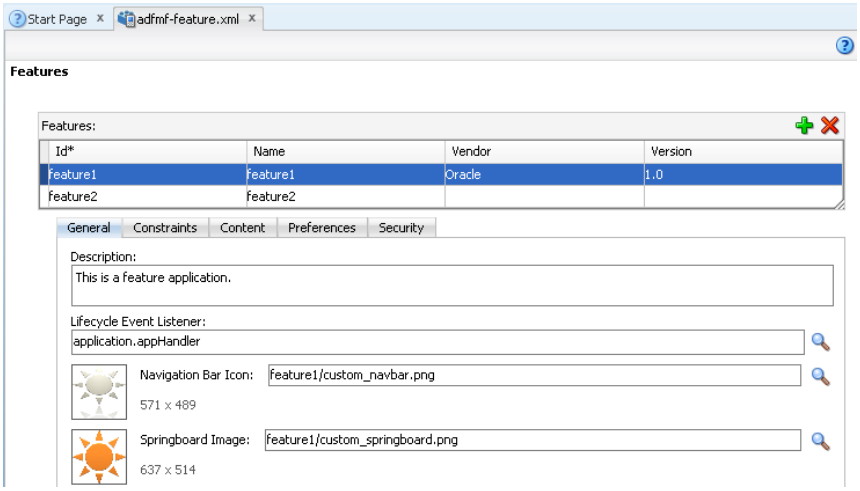
5.8 Setting the Basic Configuration for the Application Feature

Each mobile application must have at least one application feature. Because each application feature can be developed independently from one another (and also from the mobile application itself), the overview editor for the `adfmf-feature.xml` file enables you to define the child elements of `<adfmf:features>` to differentiate the application features by assigning each a name, an ID, and setting how their content can be implemented. Using the overview editor for application features, you can also control the runtime display of the application feature within mobile application and designate when an application feature requires user authentication.

5.8.1 How to Define the Basic Information for the Application Feature

The General tab of the overview editor, shown in Figure 5–19, enables you to add an application feature, designate its basic information, and its display icons.

Figure 5–19 The General Tab of the Feature Application



Before you begin:

If an application feature uses custom images for the navigation bar and springboard rather than the default ones provided by ADF Mobile, you must create these images to the specifications described the Android Developers website (`http://developer.android.com/guide/practices/ui_guidelines/icon_design.html`) and in the "Custom Icon and Image Creation Guidelines" chapter in *iOS Human Interface Guidelines*, which is available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

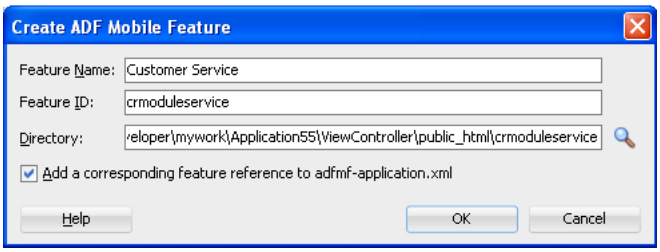
You place these images in the view controller project's `public_html` directory. See also Section 5.9.2, "What You May Need to Know About Selecting External Resources."

In addition, you must open the `adfmf-feature.xml` file and select the General tab.

To set the basic information for the application feature:

1. Choose the General tab.
2. Click **Add** in the Features table.
3. Complete the Create ADF Mobile Feature dialog, shown in Figure 5–17, and then click **OK**. To complete this dialog:
   - Enter a display name for the application feature in the **Feature Name** field.
   - Enter a unique identifier for the application feature in the **Feature ID** field.
   - If needed, change the location for the application feature to any directory within the `public_html` directory (the default parent directory). Enter this location in the **Directory** field.
   - To include the newly defined application feature in the mobile application, add a new `<adfmf:featureReference>` element to the `adfmf-application.xml` file with the `id` attribute that matches the value that you entered in the **Feature ID**. Choose **Add a corresponding feature reference to adfmf-application.xml**. The table in the Feature References page, shown in Figure 5–13, includes the feature reference after you complete the dialog. See also Section 5.5.2, "What You May Need to Know About Feature Reference IDs and Feature IDs."

Figure 5–20 Adding an Application Feature



4. In the General tab of the overview editor, enter the originator of the application feature in the **Vendor** field. This is an optional value.
5. Enter the version number of the application feature in the **Version** field. This is an optional value.
6. Enter a brief description of the application's purpose in the **Description** field. This is an optional value.
7. Enter the fully qualified class name (including the package, such as `oracle.adfmf.feature`) using the Class and Package Browser in the **Lifecycle Event Listener** field to enable runtime calls for start, stop, hibernate, and return to hibernate events. For more information, see Section 5.6, "About Lifecycle Event Listeners." This is an optional value.
8. In the Navigation Bar Icon and Springboard Image fields, browse to, and select, images from the project to use as the icon in the navigation bar and also an image used for the display icon in the springboard. You can also drag and drop the image files from the Application navigator into the file location field. This is an optional value.

5.9 Defining the Content Types for an Application Feature

Adding a child element to the `<adfmf:content>` element, shown in Example 5–16, enables you to define if the application feature is implemented as local HTML, remote URL, a Mobile ADF AMX file. You can implement each application feature as one or all of the content types.

Example 5–16 The <adfmf:content> Element

```
<adfmf:content id="Feature1">
      <adfmf:amx file="FeatureContent.amx">
</adfmf:content>
```

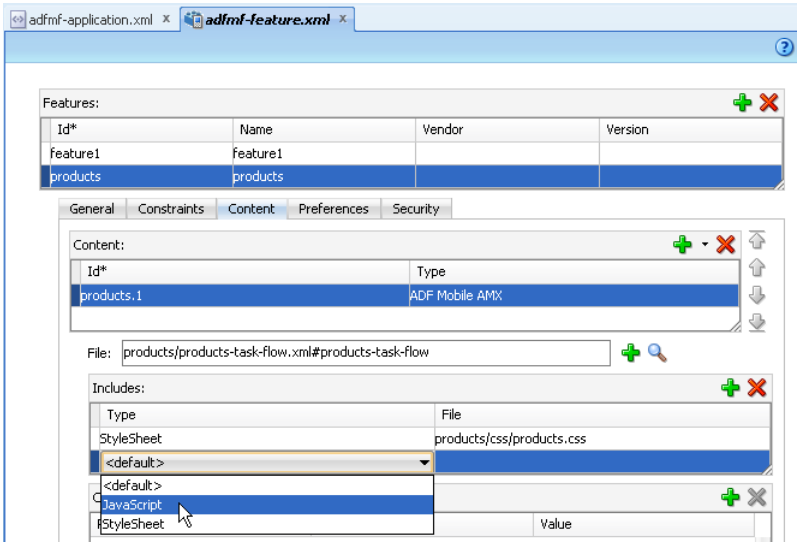
5.9.1 How to Define the Application Content

The Content tab of the overview editor, shown in Figure 5–21, provides you with dropdown lists and fields for defining the target content-related elements and attributes shown in Example 5–16. The fields within this tab enable you to set constraints that can control the type of content delivered for an application feature as well as the navigation and springboard icon images that it uses.

Each content type has its own set of parameters. As shown in Figure 5–21, for example, you must specify the location of the ADF Mobile AMX page or task flow for the application features that you implement as an ADF Mobile AMX. In addition, you can optionally select a CSS file to give the application feature a look and feel that is distinct

from other application features (or the ADF Mobile application itself), or select a JavaScript file that controls the actions of the ADF Mobile AMX components.

Figure 5–21 Defining the Implementation of the Application Feature



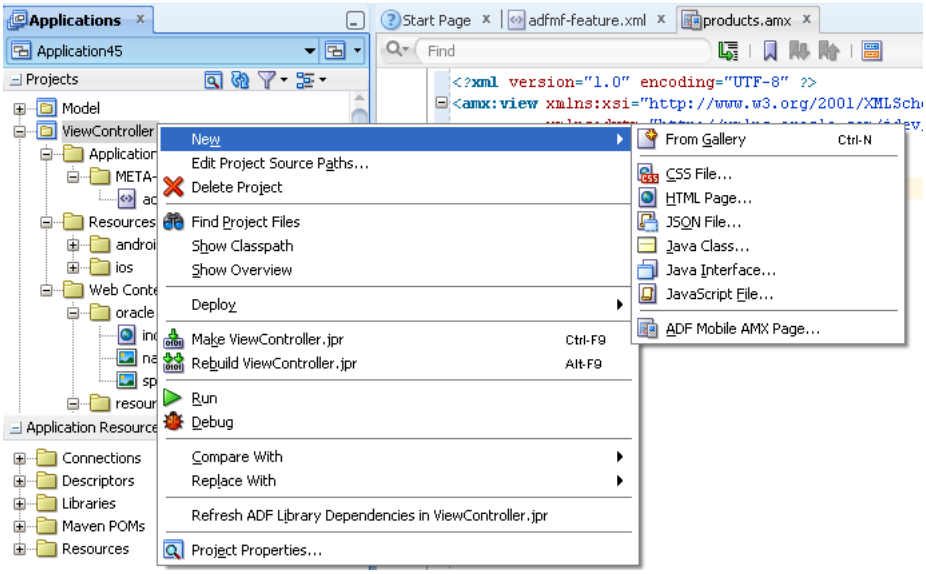
Before you begin:

Each content type has its own prerequisites, as follows:

- **ADF Mobile AMX**—An application feature implemented as ADF Mobile AMX requires an existing view (that is, a single ADF Mobile AMX page) or a bounded or unbounded task flow. Including a JavaScript file provides rendering logic to the ADF Mobile AMX components or overrides the existing rendering logic. Include a style sheet (CSS) that includes selectors to specify a custom look and feel, one that overrides the look and feel of the skinning files provided for the embedded application features at the ADF Mobile application level. By including a CSS, you ensure that the entire application feature has its own look and feel.

  If you create the ADF Mobile AMX pages as well as the mobile application that contains them, you can create both using the wizards in the New Gallery. You access these wizards by first highlighting the view controller project in the Application Navigator and then by choosing **New**. Alternatively, you can create an ADF Mobile AMX page using the context menu shown in Figure 5–22 that appears when you right-click the view controller project in the Application Navigator and then choose **New**. For information on building an ADF Mobile AMX page, see Chapter 7, "Creating ADF Mobile AMX Pages."

Figure 5–22 Context Menu for Creating an ADF Mobile Page

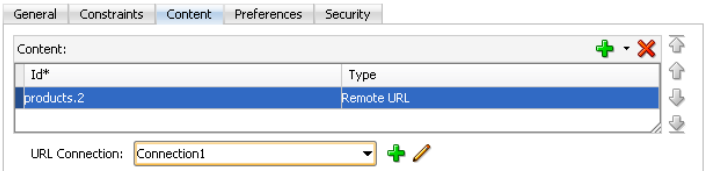


- **Remote URL**—The remote URL implementation requires a valid web address as well as a hosted mobile application. For more information, see Chapter 12, "Implementing Application Features as Remote URLs."
- **Local HTML**—An application feature implemented as local HTML requires an existing bundle of HTML files and supporting resources.

To define the application content as Remote URL or Local HTML:

1. Select an application feature listed in the Features table in the `adfmf-feature.xml` file.
2. Click **Content**.
3. Click **Add** to create a new row in the Content table (see, for example, *products.1* in Figure 5–21).
4. Select one of the following content types to correspond with the generated ID:
   - **Remote URL**
   - **Local HTML**
5. Define the content-specific parameters:
   - For remote URL content, select the connection, as shown in Figure 5–23, that represents address of the web pages on the server (and the location of the launch page).

Figure 5–23 Selecting the Connection for the Hosted Application

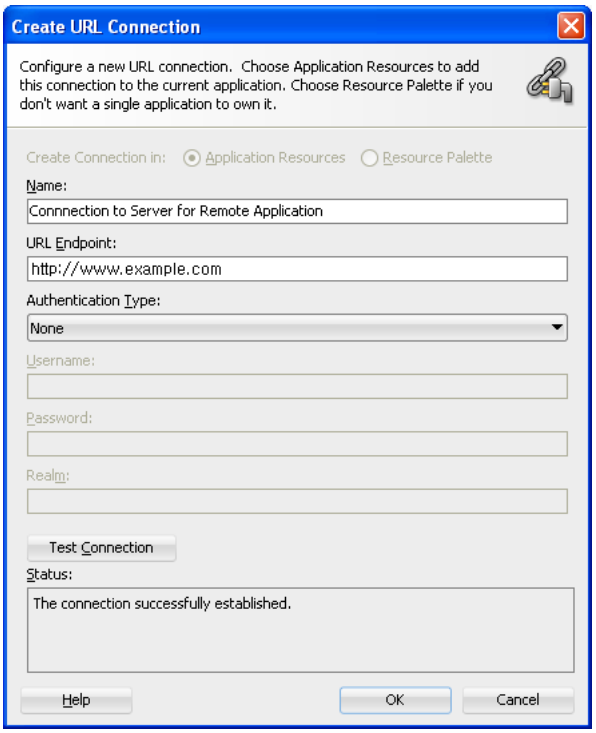


You can create this connection by first clicking **Add** and then completing the Create URL Connection dialog, shown in Figure 5–24. For more information

on this dialog, see the online help for Oracle JDeveloper. This connection is stored in the `connections.xml` file.

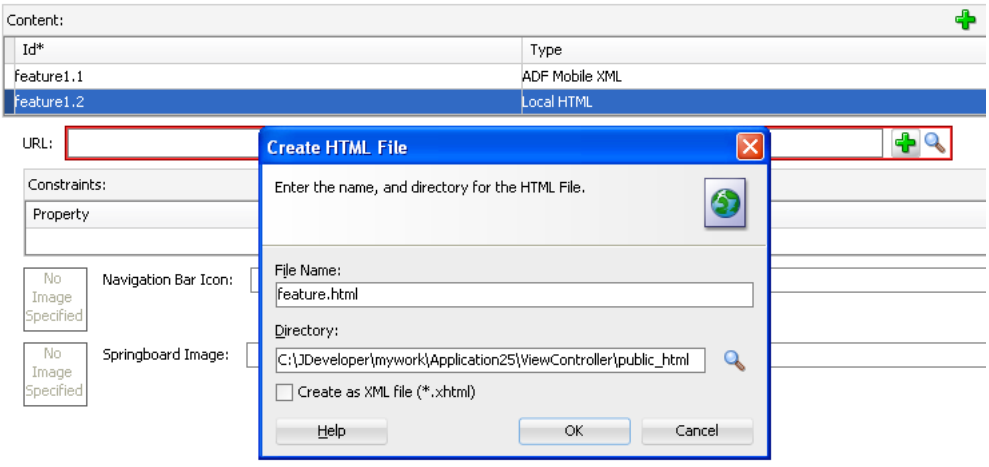
Note: This connection can only be created as an application resource.

Figure 5–24 Creating a URL Connection



- For local HTML content, enter the location of the local bundle or create the HTML page by clicking **Add** in the URL field, completing the dialog as shown in Figure 5–25, and then building the page using JDeveloper's HTML editor. Because this is an application feature, this page is stored within the Web Content folder of the view controller project.

Figure 5–25 Creating the Local HTML Page as the Content for an Application Feature

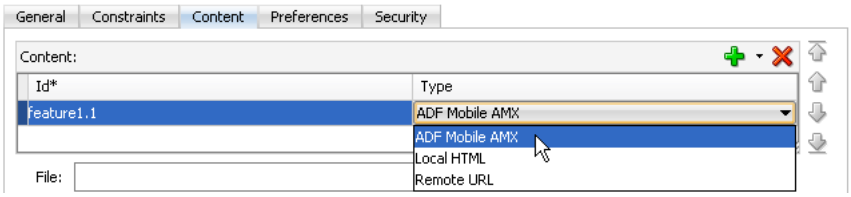


6. If needed, do the following:
   - Enter constraints that describe the conditions under which this content is available to users. For more information, see Chapter 14, "Setting Constraints."
   - Select navigation bar and springboard images.

To designate the application feature content as ADF Mobile AMX:

1. Select the application feature.
2. Click **Content**.
3. If needed, click **Add** to create a row in the Content table and the choose **ADF Mobile AMX** from the dropdown list in the Type column as shown in Figure 5–26.

Figure 5–26 Selecting ADF Mobile AMX as the Content Type

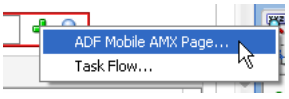


4. In the File field, click **Browse**, and choose either **ADF Mobile AMX Page** or **Task Flow**, as shown in Figure 5–27, to navigate to, and retrieve, the location of the ADF Mobile AMX page or the bounded task flow. Alternatively, you can drag and drop an ADF Mobile AMX page from the Application Navigator into the file location field.

Figure 5–27 Browsing for the File Location in the File Field

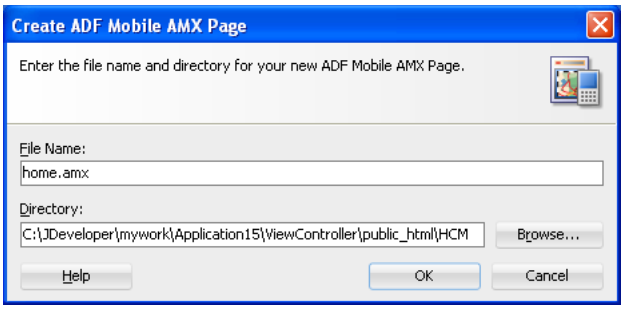
5. Alternatively, create the page or task flow. To do this:
   a. Click **Add** in the File field and then choose either **ADF Mobile AMX** or **Task Flow** as shown in Figure 5–28.

Figure 5–28 Adding an ADF Mobile AMX Page or Task Flow from the File Field



   b. Complete the *create* dialogs, such as the Create ADF Mobile AMX Page dialog shown in Figure 5–29.

Figure 5–29 The Create ADF Mobile AMX Page Dialog



c. Build the ADF Mobile AMX page or Task flow using an editor.

6. If needed, do the following:

   - Enter the JavaScript files by clicking **Add** in the Includes table, choose **JavaScript**, and then browse to the location of the file. For more information, see Section 5.11.4, "Overriding the Default Skin Styles."
   - Override the default style sheet designated as `adfmf-config.xml` by first clicking **Add** and then by choosing **Stylesheet**. Browse to the location of the file. For more information, see Section 5.11, "Skinning ADF Mobile Applications."
   - Enter the constraints, as described in Chapter 14, "Setting Constraints."
   - Select navigation bar and springboard images.

   Note: The images, style sheet, and JavaScript files must reside within the `public_html` folder to enable deployment. See Section 5.9.2, "What You May Need to Know About Selecting External Resources."

5.9.2 What You May Need to Know About Selecting External Resources

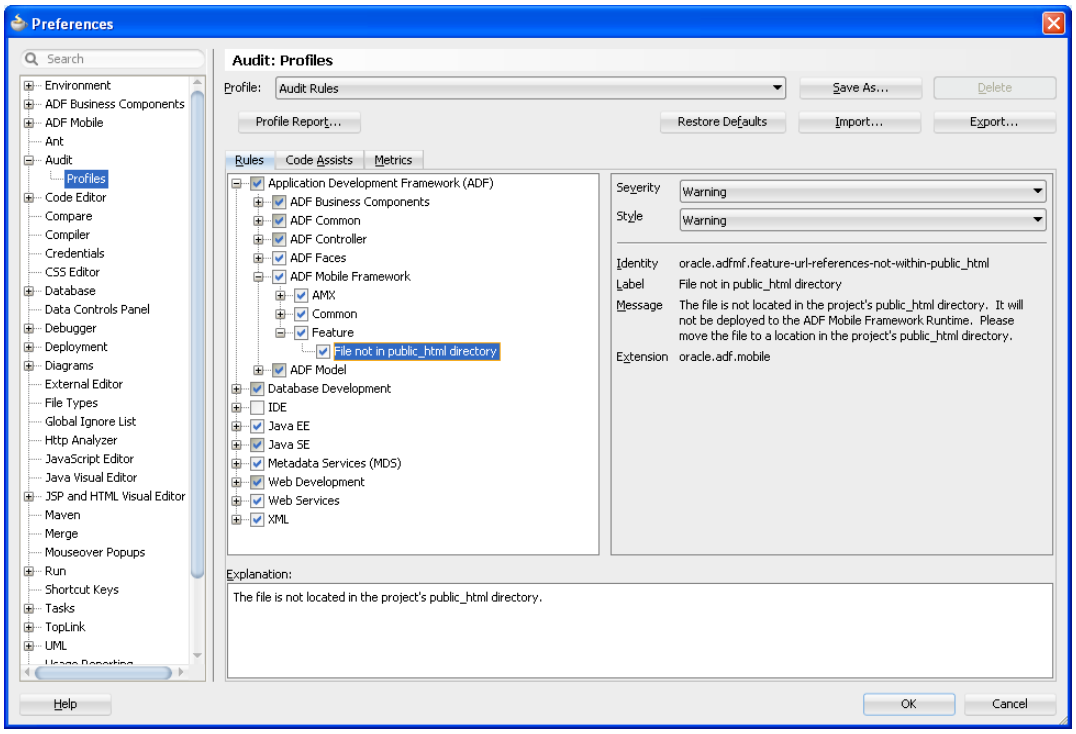
To enable deployment, all resources referenced by the following attributes must be located within the `public_html` directory of either the view controller or application controller projects.

- The `icon` and `image` attributes for `<adfmf:feature>` (for example, `<adfmf:feature id="PROD" name="Products" icon="feature_icon.png" image="springboard.png">`). See also Section 5.8.1, "How to Define the Basic Information for the Application Feature."
- The `icon` and `image` attributes for `<adfmf:content>` (for example, `<adfmf:content id="PROD" icon="feature_icon.png" image="springboard_image.png">`). See also Section 5.9, "Defining the Content Types for an Application Feature."
- The `file` attribute for `<adfmf:amx>` (for example, `<adfmf:amx file="PRODUCT/home.amx" />`). See also Section 5.9, "Defining the Content Types for an Application Feature."
- The `url` attribute for `<adfmf: localHTML>` (for example, `<adfmf:localHTML url="oracle.hello/index.html"/>`). See also Section 5.9, "Defining the Content Types for an Application Feature" and Section 17.4.10, "What Happens When You Designate a Custom Login Page."

- The file attribute defined for `type=stylesheet` and `type=JavaScript` in `<adfmf:includes>` (for example, `<adfmf:include type="JavaScript" file="myotherfile.js"/>`). See also Section 5.11, "Skinning ADF Mobile Applications."

ADF Mobile does not support resources referenced from another location, meaning that you cannot, for example, enter a value outside of the `public_html` directory using `../` as a prefix. To safeguard against referencing resources outside of `public_html`, ADF Mobile includes an audit rule called *File not in public_html directory*. You can access the ADF Mobile audit profiles, shown in Figure 5–30, from the Audit Profiles node in Preferences by choosing **Tools** > **Preferences** > **Audit** > **Profiles**.

Figure 5–30 ADF Mobile Audit Profiles



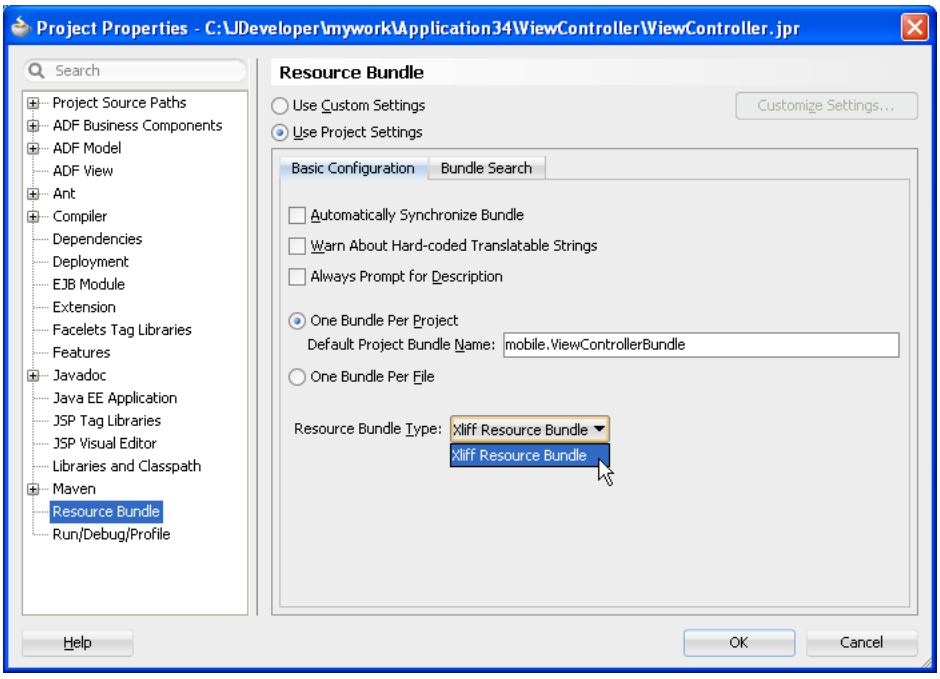
When this profile is selected, JDeveloper issues a warning if you change the location of a resource. As shown in Figure 5–31, JDeveloper displays such a warning when the default values are overridden. For information on auditing, see the "Auditing and Profiling Applications" chapter in *Oracle Fusion Middleware User's Guide for Oracle JDeveloper*.

Figure 5–31 The External Resource Warning

5.10 Working with Resource Bundles

ADF Mobile uses the standard localization structures of ADF to specify English language text resources (although you can use any tool to generate resource bundle files in other languages). You can configure an ADF Mobile application to store translatable UI strings at both the application and project level, as described in the "Working with Resource Bundles" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. ADF Mobile uses only XLIFF (XML Localization Interchange File Format) files for localization, meaning that JDeveloper produces an `.xlf` file to store the strings. (Also, **xliff Resource Bundle** is the only available resource bundle type in the Resource Bundle Settings page for projects, shown in Figure 5–32. For more information on this page, see the online help for Oracle JDevelolper.)

Figure 5–32 The Project Properties Resource Bundle Page



At the application level, you can localize strings for such attributes as application names or preference page labels, which are listed in Table 5–6.

Table 5–6 Localizable ADF Mobile Application Attributes

Element	Attribute(s)
<adfmf:Application>	name
<adfmf:PreferenceGroup>	label
<adfmf:PreferencePage>	label
<adfmf:PreferenceBoolean>	label
<adfmf:PreferenceText>	label
<adfmf:PreferenceNumber>	label
<adfmf:PreferenceList>	label
<adfmf:PreferenceValue>	name

At the project (view controller) level, you can localize application feature-related attributes listed in Table 5–7.

Table 5–7 Localizable Application Feature Attributes

Element	Attribute
<adfmf:Feature>	name
<adfmf:Constraint>	value
<adfmf:Parameter>	value
<adfmf:PreferencePage>	label
<adfmf:PrefrenceGroup>	label
<adfmf:PreferenceBoolean>	label
<adfmf:PreferenceText>	label
<adfmf:PreferenceNumber>	label
<adfmf:PreferenceList>	label
<adfmf:PreferenceValue>	name

You can create resource bundles for attributes of such ADF Mobile AMX components as the `text` attribute of `<amx:commandButton>`. Table 5–8 lists these ADF Mobile AMX (`amx`) components. For more information see Section 8.7, "Localizing UI Components."

Table 5–8 Localizable Attributes of ADF Mobile AMX Components

Component	Attribute
<amx:inputDate>	label
<amx:inputNumberSlider>	label
<amx:panelLabelAndMessage>	label
<amx:selectBooleanCheckBox>	label
<amx:selectBooleanSwitch>	label
<amx:selectItem>	label
<amx:selectManyCheckBox>	label

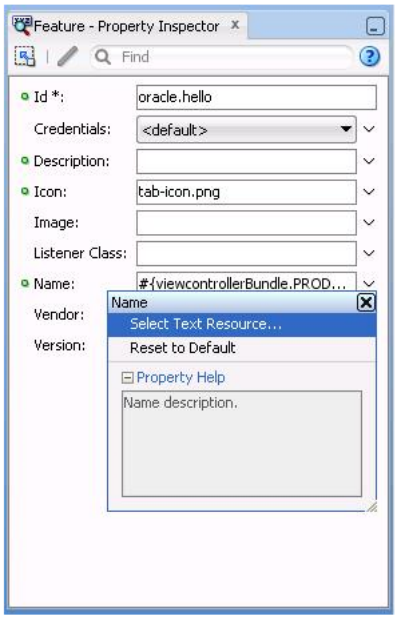
Table 5–8 (Cont.) Localizable Attributes of ADF Mobile AMX Components

Component	Attribute
`<amx:selectManyChoice>`	`label`
`<amx:selectOneButton>`	`label`
`<amx:selectOneChoice>`	`label`
`<amx:selectOneRadio>`	`label`
`<amx:commandButton>`	`text`
`<amx:commandLink>`	`text`
`<amx:goLink>`	`text`
`<amx:inputText>`	`label`, `value`, `hintText`
`<amx:outputText>`	`value`

5.10.1 How to Create Project-Level Resource Bundles

At the project-level, you create resource bundle files when you use the resource bundle dialog, accessed by clicking **Select Text Resource** in the Property Inspector, shown in Figure 5–33. This dialog enables you to automatically create text resources in the base resource bundle for the `adfmf-feature.xml` attributes listed in Table 5–7.

Figure 5–33 Localizing Feature Application Attributes



Before you begin:

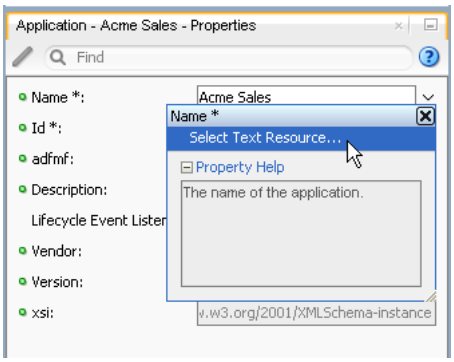
You may need to configure the resource bundle properties as described in the "How to Set Message Bundle Options" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To use strings in a resource bundle:

1. Select an attribute in the Property Inspector, such as `Name` in Figure 5–34.

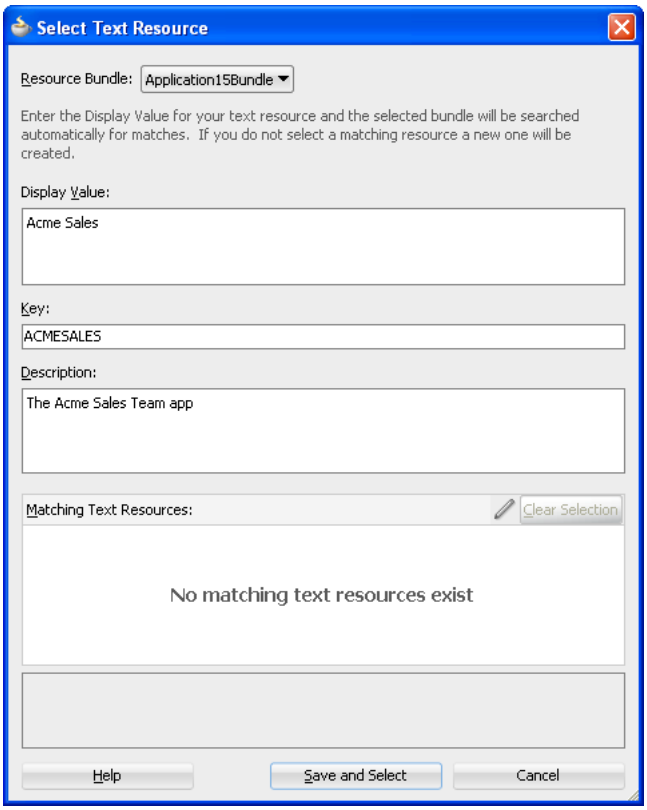
2. Choose **Select Text Resource**. Figure 5–34 shows the application name attribute selected in the Property Inspector.

Figure 5–34 Selecting the Text Resource Dialog



3. In the Select Text Resource dialog, shown in Figure 5–35, create a new string resource by entering a display name, key, and then click **Save and Select**.

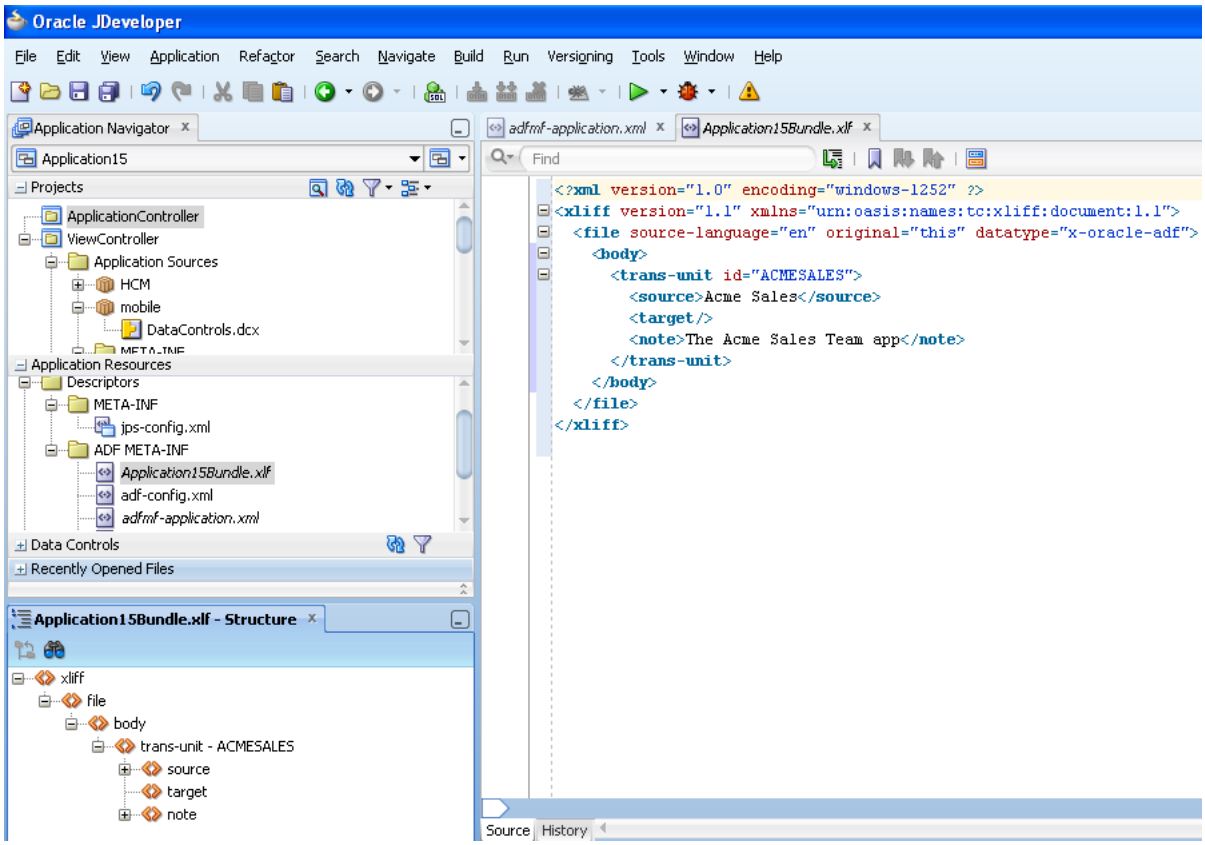
Figure 5–35 Select Resource Dialog



5.10.2 What Happens When You Add a Resource Bundle

After you add a resource in the Select Text Resource dialog, JDeveloper creates a new application-level resource bundle containing the specified display name string and key file in the ADF Meta-INF file, as shown by `Application15Bundle.xlf` in Figure 5–36.

Figure 5–36 The Application Resource Bundle



If an attribute has been localized for the first time, JDeveloper adds an `<adfmf:loadbundle>` element whose `basename` attribute refers to the newly created resource bundle.

JDeveloper also changes the localized attribute string to an EL expression that refers to the key of the string defined in the resource bundle. For example, JDeveloper changes an application name attribute called Acme Sales to `name="#{application15Bundle.ACME_SALES}"` based on the ACME_SALES value entered for the Key in the Select Resource Dialog.

JDeveloper adds each additional string that you localize to the same resource bundle file because there is only one resource bundle file at the application level.

Each `adfmf-application.xml` and `adfmf-feature.xml` file contains only one `adfmf:loadBundle` element. When you deploy an application, the resource bundles are converted into the language format expected by the runtime.

5.10.2.1 What Happens When You Create Project-Level Resource Bundles for ADF Mobile AMX Components

You can use the project- level resource bundles to localize strings for the attributes of ADF Mobile AMX components as described in Section 8.7, "Localizing UI Components." When you localize a component, such as the `text` attribute for a `<amx:commandButton>` in Example 5–17, JDeveloper transforms the string into an EL expression (such as `#{viewcontrollerBundle.MYBUTTON` in Example 5–17).

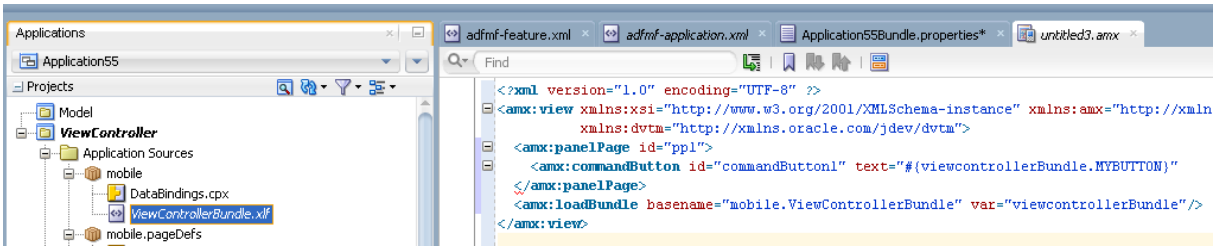
Example 5–17 Localizing an ADF Mobile XML Component

```
<?xml version="1.0" encoding="UTF-8" ?>
```

```
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:amx="http://xmlns.oracle.com/jdev/amx"
          xmlns:dvtm="http://xmlns.oracle.com/jdev/dvtm">
  <amx:panelPage id="pp1">
    <amx:commandButton id="commandButton1" text="#{viewcontrollerBundle.MYBUTTON}"
  </amx:panelPage>
  <amx:loadBundle basename="mobile.ViewControllerBundle"
var="viewcontrollerBundle"/>
</amx:view>
```

In addition, JDeveloper creates the resource bundle under the project default package, similar to `ViewControllerBundle.xlf` in Figure 5–37. In the generated `<amx:loadBundle>` element, the `basename` represents this package as, for example, `mobile.ViewControllerBundle`.

Figure 5–37 The Project-Level Resource Bundle



5.10.3 What You May Need to Know About Localizing Image Files

If an image contains text or reflects a specific country or region (for example, a picture of a flag), you can specify an alternate image as part of the localization process. You cannot hard-code the image, such as `<amx:image="image2" source="feature1/test.jpg">`. Instead, you must edit the `ViewControllerBundle.xlf` file manually by adding a `<trans-unit>` element for the image path, as illustrated in Example 5–18.

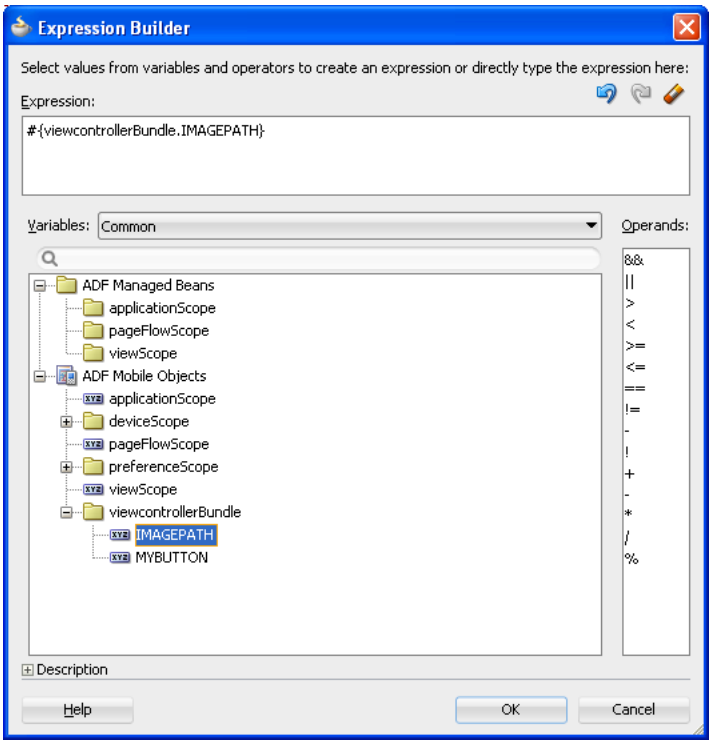
Example 5–18 Defining the Resource for an <amx:image> Component

```
<trans-unit id="IMAGEPATH">
     <source>/feature1/test.jpg</source>
     <target/>
</trans-unit>
```

> **Note:** The image location defined in the `<source>` element in Example 5–18 is relative to the location of the application feature file in `ViewController\public_html`. Alternatively, you can enter the name of the image file, such as `<source>test.jpg</source>`. See also Section 5.9.2, "What You May Need to Know About Selecting External Resources."

After you update `ViewControllerBundle.xlf`, use the Expression Builder to define an EL expression for the `source` attribute for the `<amx:image>` component, as shown in Figure 5–38.

Figure 5–38 Creating the EL Expression for a Localized <amx:image> Component



5.10.4 What You May Need to Know About XLIFF Files for iOS Applications

A family of one or more XLIFF files must exist at the location described by the `<adfmf:loadBundle>` element. A family of XLIFF files includes the following:

- Base XLIFF File—The name of the base XLIFF file must match the last token in the path specified by the `basename` attribute. This file bears the `.xlf` extension, such as `ViewControllerBundle.xlf`. In the following definition, the file is called `ViewControllerBundle`:

  ```
  <adfmf:loadBundle var="stringBundle" basename="view.ViewControllerBundle"/>
  ```

- Zero, or more, localized XLIFF Files—There can be zero (or many) localized XLIFF files for each base XLIFF file. For each file:
  - The file extension must be `.xlf`.
  - Must be co-located in the same directory as the corresponding base XLIFF file.
  - The file name is in the following format:

    ```
    <BASE_XLIFF_FILE_NAME>_<LANGUAGE_TOKEN>.xlf
    ```

    Where:

    * `<BASE_XLIFF_FILE_NAME>` is the base XLIFF file name, without the `.xlf` extension.
    * `<LANGUAGE_TOKEN>` is in the following format:

      ```
      <ISO-639-lowercase-language-code>
      ```

Note: ADF Mobile does not support countries or regions.

For example, for Spain, the language token is `es`. For more information, see the "Manually Defining Resource Bundles and Locales" section in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework.*

For example, localized file names for XLIFF files referencing a base XLIFF named `stringBundle.xlf` for language codes `en`, `es`, and `fr` would be:

* `stringBundle_en.xlf`
* `stringBundle_es.xlf`
* `stringBundle_fr.xlf`

5.10.5 Internationalization for iOS Applications

The localizable elements of the `adfmf-application.xml` and `adfmf-feature.xml` files reference internationalized strings through the use of EL-like strings in the attributes listed in Table 5–7 and Table 5–6. Because these configuration files are read early in the application lifecycle, these strings are not evaluated as EL statements at runtime. Instead, these strings are taken as the full key for the translated string in the native device translation infrastructure.

Although the Expression Language syntax is `"${BUNDLE_NAME.STRING_KEY}"`, ADF Mobile uses the entire string enclosed by `"#{}"` as the key to look up the translated string. These strings are in the form of `#{bundleName.['My.String.ID']}`, where the XLIFF string is separated by periods and `#{bundleName.['MyStringID']}`, which is used only for string identifiers that are not separated by periods. Example 5–19 illustrates the latter, such as `#{strings.CONTACTS}`, that modify the `name` attribute. For the iOS native framework, the deployment ensures that the content of that statement matches the proper key in the `*.string` file used for translation.

Only the attributes that are displayed to the end user, or control the location of content displayed to the end user, support the use of internationalized strings. These include the following attributes:

- The `<adfmf:application>` element's `name` attribute
- The `<adfmf:feature>` element's `name` attribute
- The `<adfmf:feature>` element's `icon` attribute
- The `<adfmf:feature>` element's `image` attribute
- The `<adfmf:content>` element's `icon` attribute
- The `<adfmf:content>` element's `image` attribute

Example 5–19 shows an application feature with `name`, `icon`, and `image` attributes use internationalization strings.

Example 5–19 Internationalization Using Strings

```
<adfmf:feature id="CTCS" name="#{strings.CONTACTS}"
                         icon="#{strings.CONTACTS_ICON}"
                         image="#{strings.CONTACTS_IMAGE}">
        <adfmf:constraints>
            <adfmf:constraint property="user.roles"
```

```
                        operator="contains"
                        value="employee" />
        </adfmf:constraints>
        <adfmf:description>The HTML Device Contacts</adfmf:description>
        <adfmf:loadBundle basename="mobile.adfmf-stringsBundle"
                          var="strings"/>
        <adfmf:content id="CTCS.Generic">
            <adfmf:constraints />
            <adfmf:localHTML url="contacts.html" />
        </adfmf:content>
    </adfmf:feature>
```

When you define the `<adfmf:loadBundle>` elements, as shown in Example 5–20, you create the mapping of bundle names to actual bundles. The bundle name is used when the expression is evaluated.

Example 5–20 Mapping Bundle Names Using <adfmf:loadBundle>

```
<adfmf:constraint property="user.roles"
                  operator="contains"
                  value="employee" />
        </adfmf:constraints>
        <adfmf:description>The HTML Device Contacts</adfmf:description>
        <adfmf:loadBundle basename="mobile.adfmf-featureBundle"
                          var="mobileBundle"/>
        <adfmf:loadBundle basename="mobile.adfmf-stringsBundle"
                          var="strings"/>
```

ADF Mobile's runtime holds the `<adfmf:loadBundle>` elements until the first access of the JVM. After the ADFChannel for the application feature is created, a message is sent to the JVM to initialize the mapping of basenames to EL names for all of the bundles. Example 5–21 illustrates the structure of the message sent to the JVM:

Example 5–21 The Message Initializing Mapping of Basenames to EL Names

```
{classname:"oracle.adfmf.framework.api.Model",method:"setBundles",

params:[[{basename:"mobile.adfmf-featureBundle",elname:"mobileBundle"},{basename:"
mobile.adfmf-stringsBundle",elname:"strings"}]]}
```

5.11 Skinning ADF Mobile Applications

ADF Mobile's use of CSS (Cascading Style Sheet) language-based skins ensures that all application components within an ADF Mobile application (including those used in its constituent application features) share a consistent look and feel. Changes that affect the presentation of an application can be made by creating or extending a skin rather than through re-configuring an ADF Mobile AMX or HTML component. Any changes made to a skin take effect when an application starts, because ADF Mobile applies skins at runtime.

As noted in Section 4.2.2, "What Happens When You Create an ADF Mobile Application," the artifacts resulting from the creation of an application controller project include two skinning-related files, `adfmf-config.xml` and `adfmf-skins.xml`. You use these files to control the skinning for the ADF Mobile application. The `adfmf-config.xml` file designates the default skin family used to render application components and the `adfmf-skins.xml` file enables you to customize the default skin family or define a new one.

5.11.1 About the adfmf-config.xml File

After you create an ADF Mobile application, JDeveloper populates the `adfmf-config.xml` file to the ADF Mobile application's **META-INF** node. The file itself is populated with the base ADF Mobile skin family, `MobileFusionfx`, illustrated in Example 5–22.

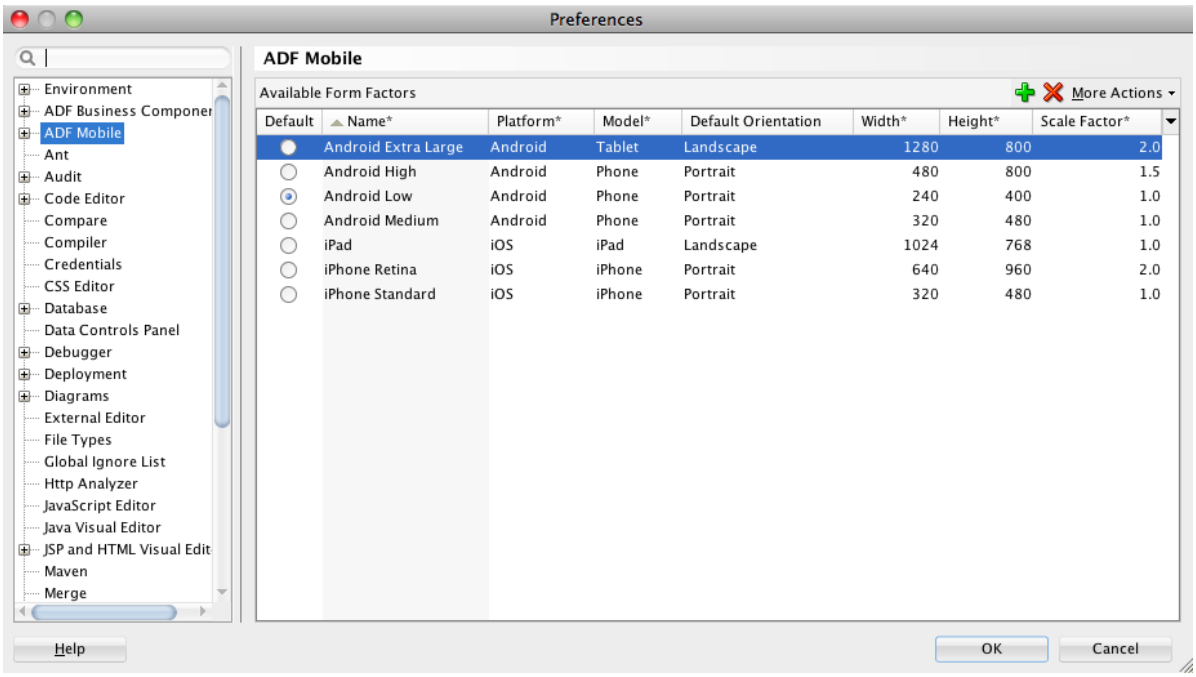
Example 5–22 The Default Skin, mobileFusionFX in the adfmf-config.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-config xmlns="http://xmlns.oracle.com/adf/mf/config">
  <skin-family>mobileFusionFx</skin-family>
<skin-version>1</skin-version>
</adfmf-config>
```

Note: The `<skin-family>` element does not support EL expressions. The `<skin-version>` element has only one value, `1`.

ADF Mobile defines skins for the `mobileFusionFx` skin family. The skinning hierarchy descends from `mobileFusionFx`, the top-most skin, the next skin is the platform (iOS), and device-type skins (iPad and iPhone) are at the third level. Figure 5–39 shows the list of seeded form factors in ADF Mobile preference page, which illustrates this hierarchy in terms of platform and model. For more information on how skins are applied at various levels, see Section 5.11.5, "What You May Need to Know About Skinning."

Figure 5–39 The Skinning Hierarchy



5.11.2 About the adfmf-skins.xml File

The `adfmf-skins.xml` file, located in the **META-INF** node of the application controller project, uses the `<skin>` and the `<skin-addition>` elements. Use the

<skin> element to extend a skin family. The <skin-addition> element adds a style sheet to an existing skin.

By default, this file is empty, but the elements listed in Table 5–9 describe the child elements that you can use to populate this file to extend mobileFusionFx or to define the CSS files that are available to the application. You use the <skin> element to create new skins or extend an existing skin. Do not use the <extends> element when defining a new skin.

Table 5–9 Child Elements of the <skin> Element

Elements	Description
<id>	Because ADF Mobile skinning customizes the default mobileFusionFx skin, the identifier syntax reflects the version of the operating system. This is a required element. To define this element in terms of mobileFusionFx, mobileFusionFx.iOS identifies a style sheet that renders components on the Apple iPad and the iPhone. mobileFusionFx.iPad and mobileFusionFx.iPhone render exclusively to the iPad or iPhone, respectively.
<family>	Identifies the skin family. This is a required element.
<extends>	Use this element to customize the default skin, mobileFusionFx, to a specific device. <?xml version="1.0" encoding="UTF-8" ?> <adfmf-skins xmlns="http://xmlns.oracle.com/adf/mf/config"> <skin> <id>mobileFusionFx.iPhone</id> <family>mobileFusionFx</family> <extends>mobileFusionFx.iOS</extends> <style-sheet-name>styles/myiphone.css</style-sheet-name> </skin> </adfmf-skins> A <skins> element with no <extends> clause means that a new skin is defined.
<style-sheet-name>	The file location of the CSS file (a relative URL). This is an optional element.
<version>	An optional element that can be used if you change the CSS to fix bugs.

Table 5–10 lists elements that you can use to define the <skin-addition> element in an ADF Mobile CSS when you must integrate style sheet into an existing skin.

Table 5–10 The <skin-addition> Child Elements

Element	Description
<skin-id>	The name of the existing skin which you are injecting into the style sheet. This identifier must match the value for the <id> subelement of <skin>.
<style-sheet-name>	The relative URL for the style sheet for the skin.

Example 5–23 illustrates designating the location of the CSS file in the <style-sheet-name> element and the target skin family in <skin-id>.

Example 5–23 Using the <skin-addition> Element

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-skins xmlns="http://xmlns.oracle.com/adf/mf/config">
```

```
  <skin-addition>
    <skin-id>mobileFusionFx.iPhone</skin-id>
    <style-sheet-name>skins/mystyles.iphone.addition1.css</style-sheet-name>
  </skin-addition>
</adfmf-skins>
```

5.11.2.1 About the ADF Mobile Skin Identifiers

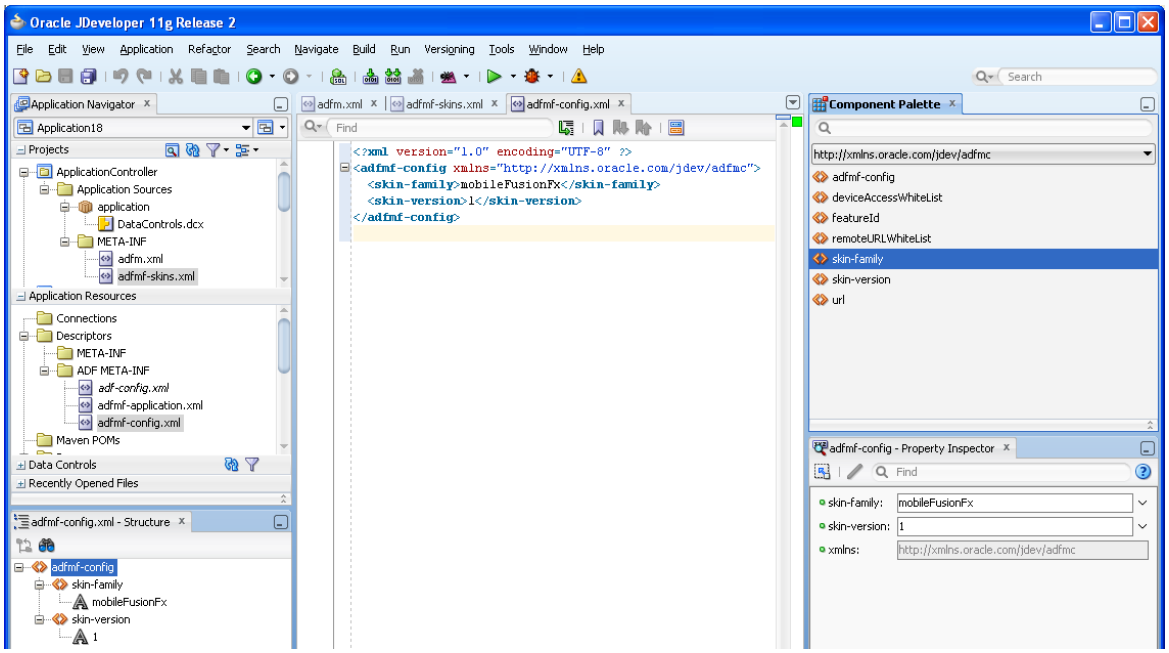
The skin identifier (`<skin-id>`) is identical to the `<skin-family>` element and can be appended with the operating system, such as iOS, iPhone, or iPad. ADF Mobile first searches for the skin identifier in the format of `<skin-family>.<OS>`, such as `mobileFusionFx.iPad`. If it cannot find the skin, it searches instead for a skin identifier of `<skin-family>` which falls back to the default skin family, `mobileFusionFx`. (This top-level skin is referenced by the `<skin-id>` which equals the `<skin-family>`). The `<skin-family>` must always be identified in the `adfmf-config.xml` file.

5.11.3 How to Change the Default Skin Family

By editing the `adfmf-config.xml` and `adfmf-skins.xml` files, which respectively designate the current skin used by the ADF Mobile application and then describing the skin families available to it, you can customize, add, and test skins.

You can edit the source file directly, or use the Structure window, shown in Figure 5–40, to update the `adfmf-skins.xml` and `adfmf-config.xml` files. You maintain all of the skins and skin additions within the `adfmf-skins.xml` file and then designate the default skin for rendering the application components in the `adfmf-config.xml` file. You create new CSS files within the application controller project using the Create Cascading Style Sheet dialog. Because the CSS files are associated with the `<skin-id>`, you then update the `<skin-id>` elements in `adfmf-skins.xml` with the new CSS references. If a skin family is not specified in `adfmf-config.xml`, then it cannot be used for skinning.

Figure 5–40 Configuring the Skinning Files

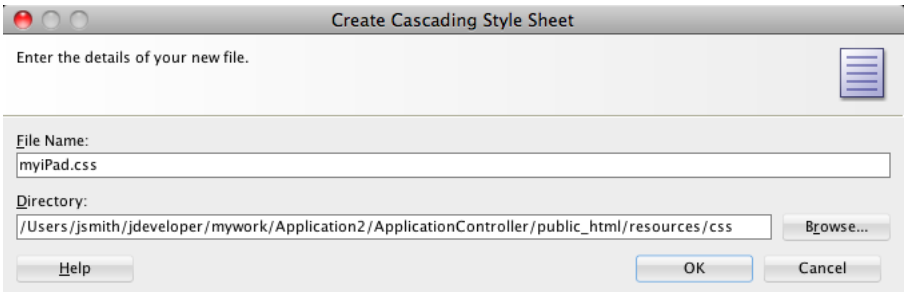


To customize the default skin:

1. If needed, create a new CSS file:
   a. Right-click the application controller project and then choose **New**, or click **File** > **New**.
   b. In the New Gallery, select **HTML** (located within **Web Tier**) and then **CSS File**. Click **OK**.
   c. Complete the Create Cascading Style Sheet dialog by entering a name for the CSS file and also a directory location.

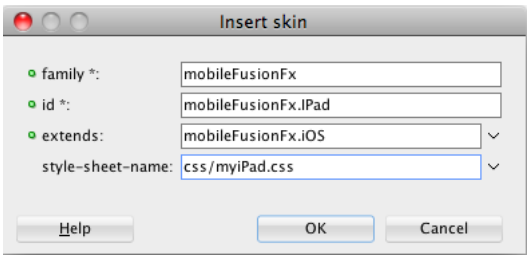
   Note: The CSS file must reside somewhere within the `public_html` folder of the application controller project (such as `/Users/`*`user name`*`/jdeveloper/mywork/`*`Mobile Application`*`/ApplicationController/public_html/resources/css`). The `adfmf-skins.xml`, where you reference this CSS, is also located within the application controller project (`application_workspace/ApplicationController/src/META-INF`).

Figure 5–41 The Create Cascading Style Sheet Dialog



2. Open the `adfmf-skins.xml` file.
3. Drag and drop a `<skin>` component from the ADF Mobile Skins Component Palette to the Structure window.
4. Populate the `<skin>` element with the elements described in Table 5–9 by completing the Insert skin dialog, shown in Figure 5–42, as follows:
   - **family**—Enter the base skin family. For these styles to be applied, the base skin family must be the same as the one defined by the `<skin-family>` element in the `adfmf-config.xml` file.
   - **id**—Enter the identifier for the skin.
   - **extends**—Enter the name of the parent style. For the `mobilefusionFX` family, enter the base family, appended with the operating system (`mobileFusionFx.iOS`).
   - **style-sheet-name**—Retrieve the style sheet.

Figure 5–42 The Insert Skin Dialog

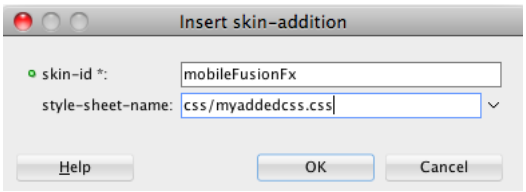


5. Click **OK**.
6. Define the CSS within the `adfmf-config.xml` file. The context of the CSS definition within this file depends on whether this style sheet is used to extend the `mobilefusionfx` skin family or be added to it.

To add a new style sheet to a skin

1. Drag and drop a `<skin-addition>` element from the Component Palette to the Structure window.
2. Populate the `<skin-addition>` element with the elements described in Table 5–10 by completing the Insert skin-addition dialog, shown in Figure 5–43.
   - Enter the identifier of the skin to which you want to add a new style.
   - Retrieve the location of the CSS file.

Figure 5–43 The Insert skin-addition Dialog



3. Click **OK**.

> **Caution:** Creating custom styles that use DOM-altering structures can cause ADF Mobile applications to hang. Specifically, the `display` property causes rendering problems in the HTML that is converted from ADF Mobile AMX. This property, which uses such values as `table`, `table-row`, and `table-cell` to convert components into a table, may result in table-related structures that are not contained within the appropriate parent table objects. Although this problem may not be visible within the application user interface itself, the logging console reports it through a Signal 10 exception.

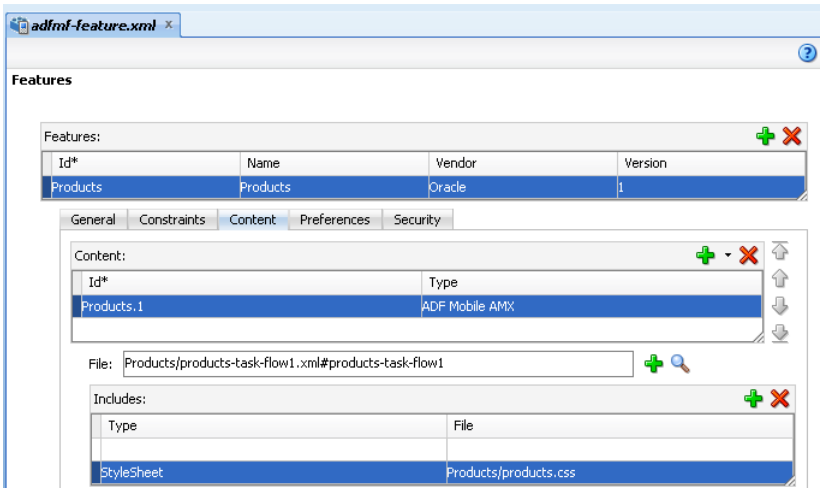
5.11.4 Overriding the Default Skin Styles

For an ADF Mobile AMX application, you can designate a specific style for the application feature implemented as ADF Mobile AMX, thereby overriding the default skin styles set at the application-level within the `adfmf-config.xml` and `adf-skins.xml` files. You add individual styles to the application feature using a CSS file as the *Includes* file.

5.11.4.1 How to Apply New Style Classes to an Application Feature

The Includes table in the overview editor for the `adfmf-feature.xml` files enables you to add a cascading style sheet (CSS) to an ADF Mobile AMX application feature.

Figure 5–44 The Includes Table



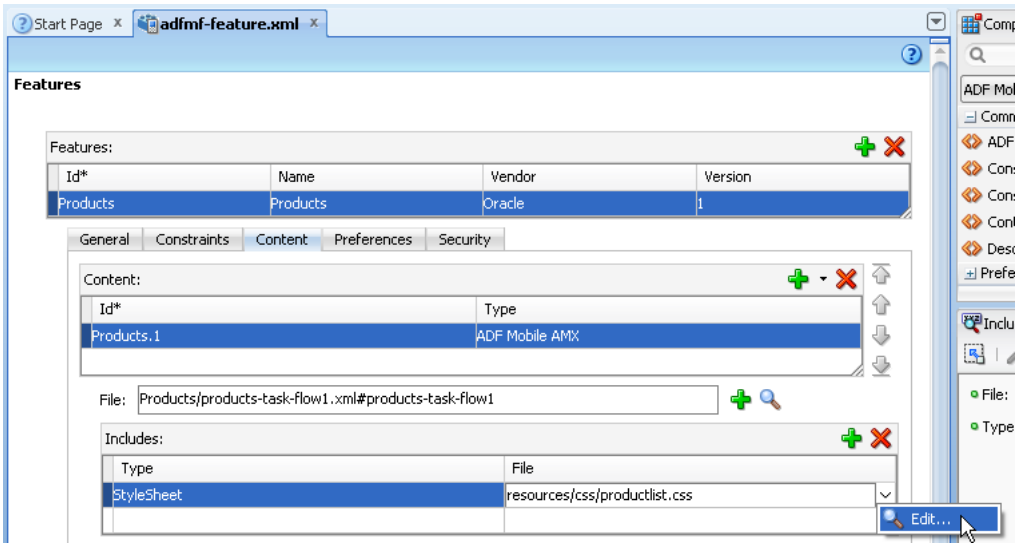
Before you begin:

Create an ADF Mobile task flow as described in Section 7.2, "Creating Task Flows." Create or add a Cascading Style Sheet as described in the "Importing ADF Skins from an ADF Library JAR" and "About Creating an ADF Skin" sections in *Oracle Fusion Middleware Creating ADF Skins with Oracle ADF Skin Editor*.

How to add a style to an application feature:

1. Click **Add** to create a new row in the Includes table.
2. Choose **Stylesheet** from the dropdown menu in the Type column, as shown in Figure 5–45.

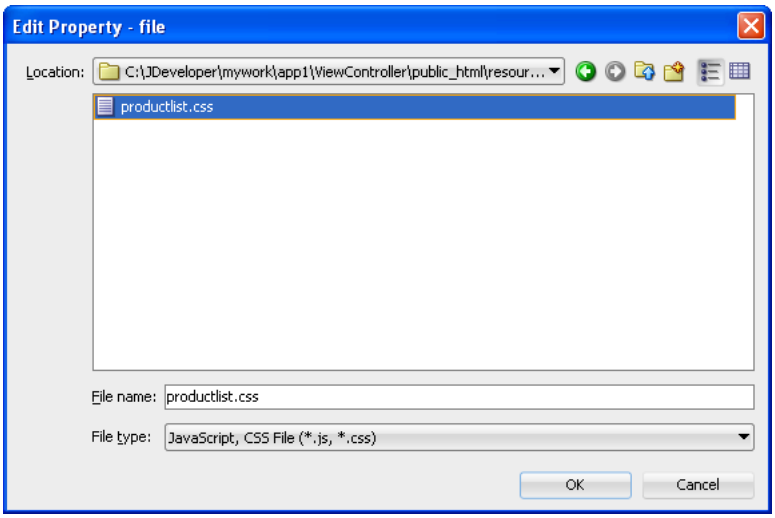
Figure 5–45 Selecting the StyleSheet Option



3. In the File column, click **Edit**, as shown in Figure 5–45.

4. Select the CSS from the Edit Property dialog, shown in Figure 5–46, and then click **OK**.

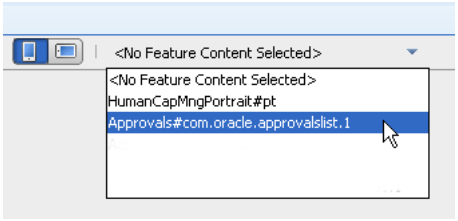
Figure 5–46 Selecting the CSS for the Application Feature



5.11.4.2 What Happens When You Apply a Skin to a Feature Application

After you add a CSS (or JavaScript file) to the Includes table, the CSS and JavaScript pages added to the application feature can be applied to an ADF Mobile AMX page by selecting the application feature from the Feature Content dropdown menu in the preview pane of the ADF Mobile AMX editor, as shown in Figure 5–47.

Figure 5–47 The Feature Content Dropdown Menu



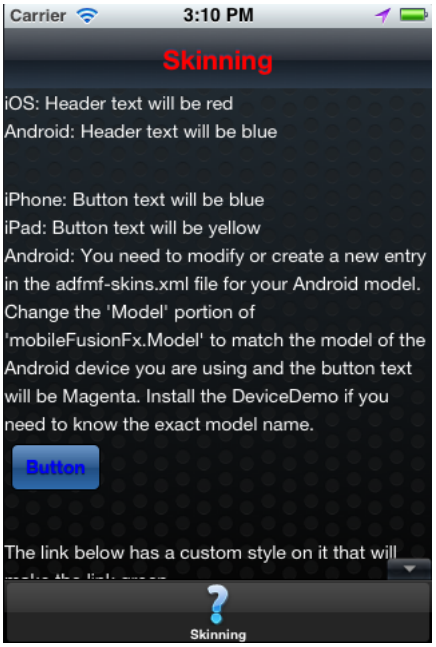
5.11.5 What You May Need to Know About Skinning

The CSS files defined in the `adfmf-skins.xml` file, illustrated in Example 5–24, show how to extend a skin to accommodate the different display requirements of the Apple iPhone and iPad. These styles are applied in a descending fashion, as illustrated in Figure 5–39. The SkinningDemo sample application provides a demonstration of how customized styles can be applied when the application is deployed to different devices. This sample application is in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

jdev_install`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples`

For example, at the iOS level, the stylesheet (`mobileFusionFx` in Example 5–24) is applied to both an iPhone or an iPad. For device-specific styling, you would define the `<skin-id>` elements for the iPhone and iPad skins. The skinning demo application illustrates the use of custom skins defined through this element. Figure 5–48 shows how a custom style is applied to an iPhone.

Figure 5–48 iPhone-Specific Rendering



Example 5–24 Skinning Levels Defined in the adfmf-skins.xml File

adfmf-skins.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf-skins xmlns="http://xmlns.oracle.com/adf/mf/skins">
    <skin>
        <id>mobileFusionFx.iPhone</id>
        <family>mobileFusionFx</family>
        <extends>mobileFusionFx.iOS</extends>
        <style-sheet-name>skins/mobilefusionfx.iphone.css</style-sheet-name>
    </skin>
    <skin>
        <id>mobileFusionFx.iPad</id>
        <family>mobileFusionFx</family>
        <extends>mobileFusionFx.iOS</extends>
        <style-sheet-name>skins/mobilefusionfx.ipad.css</style-sheet-name>
    </skin>
    <skin>
        <id>mobileFusionFx.iPod</id>
        <family>mobileFusionFx</family>
        <extends>mobileFusionFx.iOS</extends>
        <style-sheet-name>skins/mobilefusionfx.ipod.css</style-sheet-name>
    </skin>
    <!--  Skin Additions -->
    <skin-addition>
        <skin-id>mobileFusionFx.iPhone</skin-id>
        <style-sheet-name>skins/mystyles.iphone.addition1.css</style-sheet-name>
    </skin-addition>
    <skin-addition>
        <skin-id>mobileFusionFx.iPhone</skin-id>
        <style-sheet-name>skins/mystyles.iphone.addition2.css</style-sheet-name>
    </skin-addition>
    <skin-addition>
        <skin-id>mobileFusionFx.iOS</skin-id>
        <style-sheet-name>skins/mystyles.ios.addition2.css</style-sheet-name>
    </skin-addition>
```

```
</adfmf-skins>
```

5.12 Working with Feature Archive Files

The `adfmf-application.xml` file references at least one application feature. These application features, when packaged into a JAR file known as a Feature Archive file (FAR), provide the reusable content that can be consumed by other ADF Mobile applications. A FAR is essentially a self-contained collection of everything that an application feature requires, such as icon images, resource bundles, HTML, JavaScript, or other implementation-specific files. As described in Section 16.5, "Deploying Feature Archive Files (FARs)," the contents of a FAR includes a single `adfmf-feature.xml` file, which identifies each of the packaged application features by a unique ID. You can edit this file, as described in Section 5.7, "About the Mobile Feature Application Configuration File," to update such feature properties as content implementation (local or remote HTML files or ADF Mobile AMX pages) and display based on such factors as user roles and privileges or device properties. A mobile application can reference one FAR, several of them, or none at all.

5.12.1 How to Use FAR Content in an ADF Mobile Application

You make an application feature available to an ADF Mobile application by adding it to the consuming application's class path.

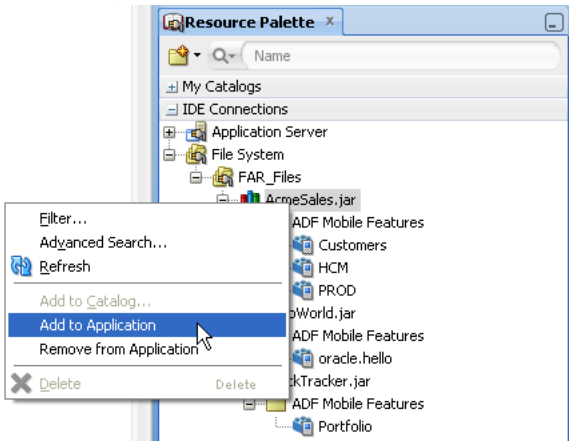
Before you begin:

Deploy the application feature as a Feature Archive file, as described in Section 16.5.2, "How to Deploy the Feature Archive Deployment Profile."

How to add application feature content to an ADF Mobile application:

1. Open the Resource Palette and then choose **New File Systems Connection**.
2. Complete the File Systems Connection dialog to create a file connection to the directory that contains the Feature Archive JAR file. For more information, refer to the Oracle JDeveloper Online help.
3. Right-click the Feature Archive file (which is noted as a JAR file) in the Resource Palette.
4. Choose **Add to Application** to add consuming application's classpath, as shown in Figure 5–49.

Figure 5–49 Adding a FAR to an ADF Mobile Application



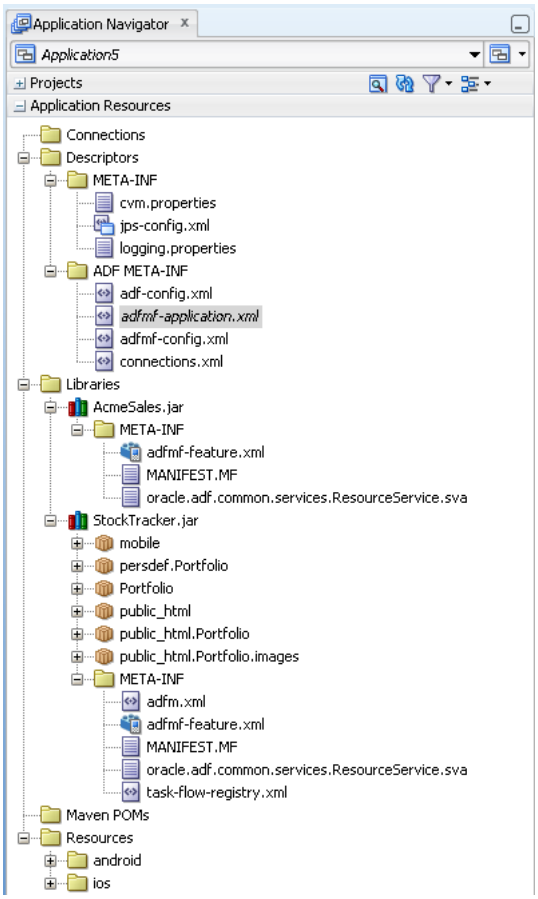
Note: Choose **Remove from Application** to remove the feature archive JAR from the consuming application's classpath.

5.12.2 What Happens When You Add a Feature Archive JAR to a Classpath

After a FAR is added to the application's classpath:

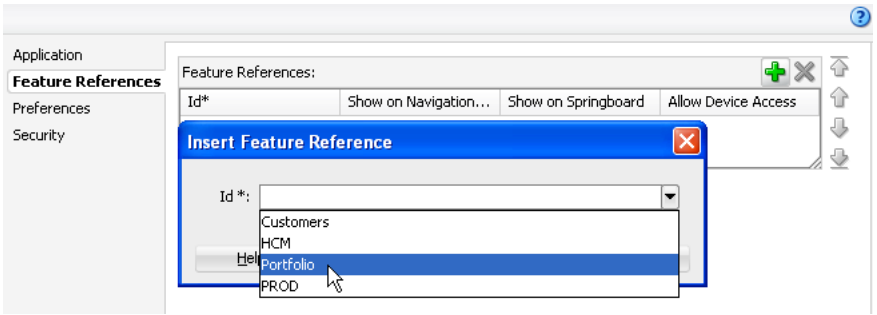
- The contents of the FAR display in the Application Resources under the **Libraries** node, as shown in Figure 5–50.

Figure 5–50 The Feature Archive File in the Application Resources of the Consuming Application



- Every application feature declared in the `adfmf-feature.xml` files included in the JARs becomes available to the consuming application, as illustrated by Figure 5–51 where the dropdown lists IDs of the available application features belonging to `AcmeSales.jar` (HCM, PROD, and Customers) as well as one called Portfolio that is defined in another Feature Archive that has been added to the application, `StockTracker.jar`. See also Section 5.5.1, "How to Designate the Content for a Mobile Application."

Figure 5–51 Referencing the Application Features Defined in Various adfmf-feature.xml Files

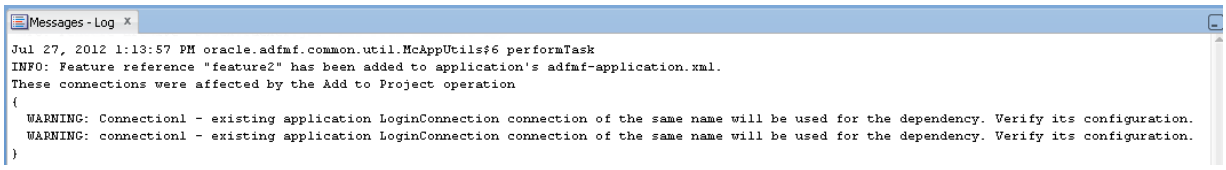


Tip: Manually adding the Feature Archive JAR to the application classpath also results in the application features displaying in the Insert Feature Reference dialog.

- The information in the `connections.xml` file located in the Feature Archive JAR is merged into the consuming application's `connections.xml` file. The Log window, shown in Figure 5–52, displays naming conflicts.

Note: You must verify that the connections are valid in the consuming application.

Figure 5–52 The Messages Log Window Showing Name Conflicts for Connections



5.12.3 What You May Need to Know About Enabling the Reuse of Feature Archive Resources

To ensure that the resources of a FAR can be used by an application, both the name of the FAR and its feature reference ID must be globally unique. Within the FAR itself, the `DataControl.dcx` file must be in a unique package directory. Rather than accepting the default names for these package directories, you should instead create a unique package hierarchy for the project. You should likewise use a similar package naming system for the feature reference IDs.

6

Controlling the Display of Application Features

This chapter describes the ADF Mobile JavaScript API extensions, the ADF Mobile Container Utilities API, and how to use the `adfmfjavaUtilities` API for custom springboard applications.

This chapter includes the following sections:

6.1 Using ADF Mobile APIs to Create a Custom Springboard Application Feature

Using JavaScript to call the JavaScript API extensions enables you to add the navigation functions to a custom springboard authored in HTML. As stated in Section 5.4.4, "What You May Need to Know About Custom Springboard Application Features with HTML Content," you can enable callbacks and leverage PhoneGap by including methods in the JavaScript `<script>` tag. Example 6–1 illustrates using this tag to call PhoneGap.

Example 6–1 Embedding the <script> Tag in an HTML Springboard Page

```
...
<script type="text/javascript" charset="utf-8" src=../../../www/js/phonegap-1.0.0.js"></script>
<script type="text/javascript" src="../../../www/js/adf.el.js"></script>
...
```

> **Note:** You can use any of the JavaScript methods located in `adf.el.js` that begin with `adf.mf.api`. For information on usage, see Section 6.2, "The ADF Mobile Container Utilities API."

ADF Mobile's use of JavaScript leverages PhoneGap's infrastructure to initialize the browser's DOM (Document Object Model) with the appropriate JavaScript constructors. It also avoids race conditions by using the PhoneGap object constructor infrastructure. ADF Mobile also posts PhoneGap messages to the native device code to initiate the ADF Mobile PhoneGap commands.

The ADF Mobile extension to the PhoneGap API enables the mobile device's API to access the configuration set in the `adfmf-feature.xml` and `adfmf-application.xml` files, which in turn results in communication between the mobile device and ADF Mobile's infrastructure. These extensions also direct the display behavior of the application features.

6.2 The ADF Mobile Container Utilities API

The methods of the ADF Mobile Container Utilities API provide ADF Mobile applications with such functionality as displaying or hiding the navigation bar, displaying a springboard, or invoking application features. You can use these methods at the Java and JavaScript layers of ADF Mobile.

In JavaScript, the Container Utilities API is located on the `adf.mf.api` JavaScript object, as shown in Example 6–2, which illustrates calling the `gotoSpringboard` method.

Example 6–2 Calling the Container Utilities API in JavaScript

```
<script type="text/javascript" charset="utf-8" src="../../../www/js/phonegap-1.0.0.js"></script>
...
adf.mf.api.gotoSpringboard();
...
```

Note: The relative path to the location of the `www/js` directory always reflects the location of the HTML springboard page, which can be located at the root of the view controller's `public_html` directory, or within a subdirectory of it. In Example 6–2, the path defined by the `src` attribute (`../../../www/js/phonegap-1.0.0.js`) is relative to the location of the HTML springboard file when it is located at the root of the `public_html` directory, as follows:

`JDeveloper\mywork\`*`Mobile_Application`*`\ViewController\public_html\customspringboard.html`

To enable the springboard files located within the subdirectories of `public_html` to access the hosted JavaScript files, you must adjust the relative path definition accordingly by adding `../` for each subdirectory location.

Because the path does not exist during design time, JDeveloper notes the JavaScript Includes in the source editor as an error by highlighting it with a red, wavy underline. This path is resolved at runtime.

In Java, the Container Utilities API is implemented as static methods on the `AdfmfContainerUtilities` class, which is located in the `oracle.adfmf.framework.api` package. Example 6–3 illustrates calling the `gotoSpringboard` method. For more information on `oracle.adfmf.framework.api.AdfmfContainerUtilities`, see *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile*.

Example 6–3 Calling the Container Utilities API in Java

```
import oracle.adfmf.framework.api.AdfmfContainerUtilities;
...
AdfmfContainerUtilities.gotoSpringboard();
```

```
...
```

6.2.1 Using the JavaScript Callbacks

The signatures of Java and JavaScript both match. In Java, they are synchronous and return results directly. Because JavaScript is asynchronous, there are two callback functions added for every function: a `success` callback that returns the results and a `failed` callback that returns any exception that is thrown. Within a Java method, the `success` value is returned from the function or method and the exception is thrown directly from the method. For example, a call with no arguments, such as `public static (, , ...) throws` is executed within Java as follows:

```
try {
      result = AdfmfContainerUtilities.(, , ...);
   }
      catch() {
       ...
      }
```

Because JavaScript calls are asynchronous, the return is required through the callback mechanism on the completion of the execution of the function. The signature for the JavaScript is `adf.mf.api.(, , ..., , )` and is invoked as follows:

```
adf.mf.api.(, , ..., , )
```

This call is defined as `function(request, response)`. The value of the `request` argument is the actual request. The `response` is defined as `function(request, response)` and its value is the actual request. The response is thrown during the execution of the function.

A call with one or more arguments, such as `public static <return value> <function name>(<arg0>, <arg1>, ...) throws <exceptions>`, is executed within Java as follows:

```
try {
      result = AdfmfContainerUtilities.<function_name>(<arg0>, <arg1>, ...);
   }
      catch(<exception>) {
       ...
      }
```

JavaScript calls cannot return a result because they are asynchronous. They instead require a callback mechanism when the execution of the function has completed. The signature for both the success and failed callbacks is `function(request, response)`, where the `request` argument is a JSON representation for the actual request and the `response` is the JSON representation of what was returned by the method (in the case of `success` callback functions) or, for `failed` callback functions, a JSON representation of the thrown exception.

6.2.2 Using the Container Utilities API

The Container Utilities API provides the following methods:

- checkforNewConfiguration—Checks for the changes in a hosted `connections.xml` file.
- getApplicationInformation—Retrieves the metadata for the ADF Mobile application.

- gotoDefaultFeature—Displays the default application feature.
- getFeatures—Retrieves the application features.
- gotoFeature—Displays a specific application feature.
- getFeatureByName—Retrieves information about the application feature using the application feature's name.
- getFeatureById—Retrieves an application feature using its ID.
- resetFeature—Resets the application feature to the same state as when it was loaded.
- gotoSpringboard—Displays the springboard.
- hideNavigationBar—Hides the navigation bar.
- showNavigationBar—Displays the navigation bar.
- "invokeMethod"—Invokes a Java method.
- invokeContainerJavaScriptFunction—Invokes a JavaScript method.

6.2.3 checkforNewConfiguration

When the Configuration Service is used, this method requests that ADF Mobile check a server that hosts the `connections.xml` file for any changes to the configured endpoints. To check for changes, ADF Mobile creates a thread that compares the connection-related content hosted on the server to the configuration on the device. If ADF Mobile finds that the device configuration is outdated, then the `checkforNewConfiguration` method issues a non-blocking call that notifies the user that a new configuration has been detected and then closes the application. ADF Mobile reminds the user to restart the application.

In Java, the method is as follows:

```
public static void checkForNewConfiguration()
                  throws oracle.adfmf.framework.exception.AdfException
```

Example 6–4 illustrates using this method.

Example 6–4 Retrieving Configuration Information Using Java

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        AdfmfContainerUtilities.checkForNewConfiguration();
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void checkForNewConfiguration(success, failed)
```

The `success` callback must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the associated `AdfmfContainerUtilities` method's return value (`void`). The `failed` callback must be in the form of `function(request,`

`response)` where the `request` argument contains the original request and the `response` argument contains the error (`AdfException`).

Example 6–5 illustrates using these callback functions to retrieve the configuration information.

Example 6–5 Using Callback Functions in JavaScript to Return Configuration Information

```
adf.mf.api.checkForNewConfiguration(
          function(req, res) { alert("checkForNewConfiguration complete"); },
          function(req, res) { alert("checkForNewConfiguration failed with " +
                                     adf.mf.util.stringify(res); }
                             );
```

6.2.4 getApplicationInformation

This method returns an `ApplicationInformation` object that contains information about the application. This method returns such metadata as the application ID, application name, version, and the vendor of a custom springboard application.

Within Java, this method is called as follows:

```
public static oracle.adfmf.framework.ApplicationInformation
              getApplicationInformation()
              throws oracle.adfmf.framework.exception.AdfException
```

Example 6–6 illustrates calling this method.

Example 6–6 Retrieving Application Information Using Java

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        ApplicationInformation[] ai = AdfmfContainerUtilities.getApplicationInformation();
        ...
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void getApplicationInformation(success, failed)
```

The `success` callback must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the associated `AdfmfContainerUtilities` method's return value, which is the `ApplicatiaonInformation` object containing application-level metadata. This includes application name, vendor, version, and application ID.

The `failed` callback must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Example 6–7 illustrates using these callback functions to retrieve the application information.

Example 6–7 Using Callback Functions in JavaScript to Application Information

```
adf.mf.api.getApplicationInformation(
```

```
 function(req, res) { alert("getApplicationInformation complete"); },
function(req, res) { alert("getApplicationInformation failed with " +
                           adf.mf.util.stringify(res); }
                         );
```

6.2.5 gotoDefaultFeature

This method requests that ADF Mobile display the default application feature. The default application feature is the one that is displayed when the ADF Mobile application is started.

> **Note:** This method may not be able to display an application feature if it has authentication- or authorization-related problems.

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void gotoDefaultFeature(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the associated `AdfmfContainerUtilities` method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the error.

Example 6–8 illustrates using these callbacks to call the default application feature.

Example 6–8 Using JavaScript Callback Functions to Call the Default Application Feature

```
adf.mf.api.gotoDefaultFeature(
          function(req, res) { alert("gotoDefaultFeature complete"); },
          function(req, res) { alert("gotoDefaultFeature failed with " +
                                     adf.mf.util.stringify(res); }
                           );
```

6.2.6 getFeatures

This method returns an array of `FeatureInformation` objects that represent the available application features. The returned metadata includes the feature ID, the application feature name, and the file locations for the image files used for the application icons. This call enables a custom springboard implementation to access the list of application features that are available after constraints have been applied. (These application features would also display within the default springboard.)

Within Java, this method is called as follows:

```
public static oracle.adfmf.framework.FeatureInformation[] getFeatures()
              throws oracle.adfmf.framework.exception.AdfException
```

Example 6–9 illustrates using this method.

Example 6–9 Retrieving the Application Feature Information Using Java

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        FeatureInformation[] fia = null;
        fia = AdfmfContainerUtilities.getFeatures();

        for(int f = 0; f < fia.length; ++f) {
           FeatureInformation fi = fia[i];
           ...
        }
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned values and the exceptions to be passed back to the JavaScript calling code as follows:

```
public void getFeatures(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the associated `AdfmfContainerUtilities` method's return value (the array of `FeatureInformation` objects).

The `failed` callback function must be in the form of `function(request, response)`, where the `request` argument contains the original request and the `response` argument contains the error (`AdfException`).

Example 6–10 Using JavaScript Callback Functions to Retrieve Application Feature Information

```
adf.mf.api.getFeatures(
          function(req, res) { alert("getFeatures complete"); },
          function(req, res) { alert("getFeatures failed with " +
adf.mf.util.stringify(res); }
                         );
```

6.2.7 gotoFeature

This method requests that ADF Mobile display the application feature identified by its ID.

> **Note:** This method may not be able to display an application feature if it has authentication- or authorization-related problems.

Within Java, this method is called as follows:

```
public static void gotoFeature(java.lang.String featureId)
                  throws oracle.adfmf.framework.exception.AdfException
```

This method's parameter, as shown in Example 6–11, is the ID of the application feature.

Example 6–11 Activating an Application Feature

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        AdfmfContainerUtilities.gotoFeature("feature.id");
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void gotoFeature(featureId, success, failed)
```

The `featureId` parameter is the application feature ID. This parameter activates the `success` callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated `AdfmfContainerUtilities` method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Example 6–12 illustrates using these callback functions to call an application feature.

Example 6–12 Activating an Application Feature Using JavaScript Callback Functions

```
adf.mf.api.gotoFeature("feature0",
          function(req, res) { alert("gotoFeature complete"); },
function(req, res) { alert("gotoFeature failed with " +
           adf.mf.util.stringify(res); }
   );
```

6.2.8 getFeatureByName

This method returns information about the application feature using the passed-in name of the application feature.

Within Java, this method is called as follows:

```
public static oracle.adfmf.framework.FeatureInformation getFeatureByName(java.lang.String
                                                                          featureName)
                                  throws oracle.adfmf.framework.exception.AdfException
```

This method's parameter, as shown in Example 6–13, is the name of the application feature.

Example 6–13 Retrieving the Application Feature Information Using the Application Feature Name

```
 ...
     try {
        FeatureInformation fi = AdfmfContainerUtilities.getFeatureByName("feature.name");
     }catch(AdfException e) {
        // handle the exception
     }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void getFeatureByName(featureName, success, failed)
```

The `featureName` parameter is the name of the application feature. The `success` callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated `AdfmfContainerUtilities` method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Example 6–14 illustrates using these callback functions.

Example 6–14 Using JavaScript Callback Functions to Retrieve the Application Feature Information Using the Application Feature Name

```
adf.mf.api.getFeatureByName("feature.name",
          function(req, res) { alert("getFeatureByName complete"); },
          function(req, res) { alert("getFeatureByName failed with " +
                                     adf.mf.util.stringify(res); }
);
```

6.2.9 getFeatureById

This method retrieves an application feature using its application ID.

Within Java, this method is called as follows:

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;
```

This method's parameter, as shown in Example 6–15, is the ID of the application feature.

Example 6–15 Retrieving an Application Feature Using its ID in Java

```
try {
     FeatureInformation fi =AdfmfContainerUtilities.getFeatureById("feature.id");
}catch(AdfException e) {
     // handle the exception
}
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void getFeatureById(featureId, success, failed)
```

The `featureId` parameter is the ID of the application feature. The `success` callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated `AdfmfContainerUtilities` method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Example 6–16 illustrates using these callback functions to retrieve an application feature.

Example 6–16 Using JavaScript Callback Functions to Retrieve an Application Feature by its ID

```
adf.mf.api.getFeatureById("feature.id",
          function(req, res) { alert("getFeatureById complete"); },
          function(req, res) { alert("getFeatureById failed with " +
adf.mf.util.stringify(res); }
   );
```

6.2.10 resetFeature

This method resets the state of the application feature. It resets the Java-side model for the application feature and then restarts the user interface presentation as if the ADF Mobile application had just been loaded and displayed the application feature for the first time.

Within Java, this method is called as follows:

```
public static void resetFeature(java.lang.String featureId)
                  throws oracle.adfmf.framework.exception.AdfException
```

The method's parameter, as shown in Example 6–17, is the ID of the application feature that is to be reset.

Example 6–17 Resetting an Application Feature in Java

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

   ...
     try {
        AdfmfContainerUtilities.resetFeature("feature.id");
     }catch(AdfException e) {
        // handle the exception
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and exception to be passed back to the JavaScript calling code as follows:

```
public void resetFeature(featureId, success, failed)
```

The `success` callback function and must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (The ID of the application feature).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Example 6–18 illustrates using these callback functions to call an application feature.

Example 6–18 Using JavaScript Callback Functions to Reset an Application Feature

```
adf.mf.api.resetFeature("feature0",
          function(req, res) { alert("resetFeature complete"); },
          function(req, res) { alert("resetFeature failed with " +
                                     adf.mf.util.stringify(res); }
                       );
```

6.2.11 gotoSpringboard

This method requests that ADF Mobile display the springboard.

> **Note:** This method may not be able to display the springboard if it has not been designated as a feature reference in the `adfmf-application.xml` file, or if it has authentication or authorization-related problems. See also Section 5.4, "Configuring the Springboard and Navigation Bar Behavior."

Within Java, this method is called as follows:

```
public static void gotoSpringboard()
```

Example 6–19 illustrates using this method

Example 6–19 Activating the Springboard in Java

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

...
  try {
     AdfmfContainerUtilities.gotoSpringboard();
  }catch(AdfException e) {
        // handle the exception
  }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void gotoSpringboard(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Example 6–20 illustrates using these callback functions.

Example 6–20 Using JavaScript Callback Functions to Activate the Springboard

```
adf.mf.api.gotoSpringboard(
    function(req, res) { alert("gotoSpringboard complete"); },
    function(req, res) { alert("gotoSpringboard failed with " +
adf.mf.util.stringify(res); }
);
```

6.2.12 hideNavigationBar

This method requests that ADF Mobile hide the navigation bar.

Within Java, this method is called as follows:

```
public static void hideNavigationBar()
```

Example 6–21 illustrates using this method.

Example 6–21 Hiding the Navigation Bar in Java

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;
```

```
 ...
try {
   AdfmfContainerUtilities.hideNavigationBar();
}catch(Exception e) {
   // handle the exception
}
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void hideNavigationBar(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Example 6–22 illustrates using these callback functions.

Example 6–22 Using JavaScript Callback Functions to Hide the Navigation Bar

```
adf.mf.api.hideNavigationBar(
      function(req, res) { alert("hideNavigationBar complete"); },
      function(req, res) { alert("hideNavigationBar failed with " +
adf.mf.util.stringify(res); }
);
```

6.2.13 showNavigationBar

This method requests that ADF Mobile display the navigation bar.

Within Java, this method is called as follows:

```
public static void showNavigationBar()
```

Example 6–23 illustrates using this method.

Example 6–23 Showing the Navigation Bar in Java

```
import oracle.adfmf.framework.api.AdfmfContainerUtilties;

...
  try {
     AdfmfContainerUtilities.showNavigationBar();
  }catch(Exception e) {
     // handle the exception
  }
```

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void showNavigationBar(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value (`void`).

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Example 6–24 illustrates using these callback functions.

Example 6–24 Using JavaScript Callback Functions to Show the Navigation Bar

```
adf.mf.api.showNavigationBar(
          function(req, res) { alert("showNavigationBar complete"); },
          function(req, res) { alert("showNavigationBar failed with " +
adf.mf.util.stringify(res); }
);
```

6.2.14 invokeMethod

This method invokes a Java method from any class in a classpath as follows:.

```
adf.mf.api.invokeMethod(classname, methodname, param1, param2, ... , paramN ,successCallback,
                        failedCallback);
```

Table 6–1 lists the parameters taken by this method.

Table 6–1 Parameters Passed to invokeJavaMethod

Parameter	Description
`classname`	The class name (including the package information) that ADF Mobile uses to create an instance when calling the Java method.
`methodname`	The name of the method that should be invoked on the instance of the class specified by the `classname` parameter.

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return value.

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

Examples of using this method with multiple parameters are as follows:

```
adf.mf.api.invokeMethod("TestBean", "setStringProp", "foo", success, failed);

adf.mf.api.invokeMethod("TestBean", "getStringProp", success, failed);
```

An example of using an integer parameter is as follows:

```
adf.mf.api.invokeMethod("TestBean", "testSimpleIntMethod", "101", success, failed);
```

The following illustrates using complex parameters:

```
adf.mf.api.invokeMethod("TestBean", "testComplexMethod",
           {"foo":"newfoo","baz":"newbaz",".type":"TestBeanComplexSubType"}, success, failed);
```

The following illustrates using no parameters:

```
adf.mf.api.invokeMethod("TestBean", "getComplexColl", success, failed);
```

The following illustrates using `String` parameters:

```
adf.mf.api.invokeMethod("TestBean", "testMethodStringStringString", "Hello ",
```

```
"World", success, failed);
```

6.2.15 invokeContainerJavaScriptFunction

The `invokeContainerJascriptFunction` invokes a JavaScript method. Table 6–2 lists the parameters passed by this method.

Table 6–2 Parameters Passed to invokeContainerJavaScriptFunction

Parameter	Description
`featureId`	The ID of the application feature used by ADF Mobile to determine the context for the JavaScript invocation. The ID determines the web view in which this method is called.
`method`	The name of the method that should be invoked.
`args`	An array of arguments that are passed to the method. Within this array, these arguments should be arranged in the order expected by the method.

This method returns a JSON object.

Example 6–25 The invokeContainerJavaScriptFunction Method

```
public static java.lang.Object invokeContainerJavaScriptFunction(java.lang.String featureId,
                                                                  java.lang.Object[] args)
                           throws oracle.adfmf.framework.exception.AdfException
```

Example 6–26 illustrates a JavaScript file called `appFunctions.js` that is included in the application feature, called `feature1`. The JavaScript method, `application.testFunction` which is described within this file, is called by the `invokeContainerJavaScriptFunction method`, shown in Example 6–27. Because the application includes a command button that is configured with an action listener that calls this function, a user sees the following alerts after clicking this button:

- APP ALERT 0
- APP ALERT 1
- APP ALERT 2

Example 6–26 appFunctions.js

```
(function()
    {
       if (!window.application) window.application = {};

       application.testFunction = function()
       {
          var args = arguments;

          alert("APP ALERT " + args.length + " ");
          return "application.testFunction - passed";
       };
    })();
```

Example 6–27 illustrates how the `invokeApplicationJavaScriptFunction` method calls the JavaScript method (`application.testFunction`) that is described in Example 6–26.

Example 6–27 Calling the JavaScript Function

```
invokeApplicationJavaScriptFuntions
    public void invokeApplicationJavaScriptFuntions(ActionEvent actionEvent) {
        AdfmfContainerUtilities.invokeContainerJavaScriptFunction("feature1",
                                                                  "application.testFunction",
                                                                   new Object[] {} );
        AdfmfContainerUtilities.invokeContainerJavaScriptFunction("feature1",
                                                                  "application.testFunction",
                                                                   new Object[] {"P1"} );
        AdfmfContainerUtilities.invokeContainerJavaScriptFunction("feature1",
                                                                  "application.testFunction",
                                                                   new Object[] {"P1", "P2"} );
    }
```

For more information, see *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile*.

In JavaScript, the `success` and `failed` callback functions enable the returned value and the exception to be passed back to the JavaScript calling code as follows:

```
public void invokeContainerJavaScriptFunction(success, failed)
```

The `success` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the associated method's return values, which are described in Table 6–2. The response contains the associated `AdfmfContainerUtilities` method's return value

The `failed` callback function must be in the form of `function(request, response)`, where the request contains the original request and the response contains the error.

6.3 Accessing Files Using the getDirectoryPathRoot Method

The `adfmfJavaUtilties` API includes the `getDirectoryPathRoot` method. This method, which can only be called from the Java layer, enables access to files on iOS systems. As shown in Example 6–28, this method enables access to files packaged in an iOS application bundle (an `.ipa` file), the location of temporary files, and also the location of the cache directory on the device. For more information on `oracle.adfmf.framework.api.AdfmfJavaUtilities`, see *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile*.

Example 6–28 Accessing Files on the iOS System

```
import oracle.adfmf.framework.api.AdfmfJavaUtilities;

...

public void getDirectoryPathRoot()
   {
  // returns the directory for storing temporary files

String tempDir =
AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.TemporaryDirectory);

 // returns the directory for storing application files

    String appDir =
AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.ApplicationDirectory);
```

```
 // returns the directory for storing cache files

    String deviceDir =
AdfmfJavaUtilities.getDirectoryPathRoot(AdfmfJavaUtilities.DeviceOnlyDirectory);

   }
```

Part IV

Creating ADF Mobile AMX Application Features

Describes how to create ADF Mobile AMX content for an ADF Mobile application, including how to create task flows, application pages, as well as use data controls, web services, and the local database.

Part IV contains the following chapters:

7

Creating ADF Mobile AMX Pages

This chapter describes how to create the ADF Mobile AMX application feature.

This chapter includes the following sections:

- Section 7.1, "Introduction to the ADF Mobile AMX Application Feature"
- Section 7.2, "Creating Task Flows"
- Section 7.3, "Creating Views"

7.1 Introduction to the ADF Mobile AMX Application Feature

ADF Mobile AMX is a subframework within ADF Mobile that provides a set of UI components that enable you to create an application feature whose behavior is identical on all supported platforms. While ADF Mobile AMX maintains the same development experience as ADF Faces by allowing you to drag these components into an editor from the Component palette or from the Data Control palette, these components are not equivalents to their ADF Faces counterparts: ADF Mobile AMX components do not support every property and behavior of ADF Faces components.

> **Note:** When developing interfaces for mobile devices, always be aware of the fact that the screen space is very limited. In addition, touchscreen support is not available on some mobile devices.

For more information, see the following:

- Section 2.8, "ADF Mobile AMX Application Feature"
- Section 4.2.4, "Creating an Application Workspace for an ADF Mobile AMX Application"
- Chapter 4, "Getting Started with ADF Mobile Application Development"
- Chapter 8, "Creating ADF Mobile AMX User Interface"
- Chapter 9, "Using Bindings and Creating Data Controls"

7.2 Creating Task Flows

Task flows allow you to define the navigation between ADF Mobile AMX pages. Using your application workspace in JDeveloper (see Section 4.2, "Creating an Application Workspace"), you can start creating the user interface for your ADF Mobile AMX application feature by designing task flows. As with any standard JSF application, ADF Mobile AMX uses navigation cases and rules to define the task flow. These

definitions are stored in a file with the default name of `ViewController-task-flow.xml` (see Section 7.2.3, "What You May Need to Know About the ViewController-task-flow.xml File").

An ADF Mobile sample application called Navigation (located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer) demonstrates how to use various navigation techniques, such as circular navigation, routers, and so on.

ADF Mobile enables you to create ADF Mobile AMX application features that have both bounded and unbounded task flows. As described in the "Task Flow Types" section of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*, a bounded task flow is also known as a task flow definition and represents the reusable portion of an application. In ADF Mobile, bounded task flows have a single entry point and no exit points. They have their own collections of activities and control-flow rules, as well as their own memory scope and managed-bean life span.

You use the ADF Mobile AMX Task Flow Designer to create bounded task flows for your feature. When designing an ADF Mobile AMX task flow, JDeveloper maintains the same experience as designing an ADF task flow, as described in the "Creating a Task Flow" section of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. Like the overview editor for task flows, this tool includes a diagrammer (see Section 7.2.4, "What You May Need to Know About the ADF Mobile Task Flow Diagrammer") in which you build the task flow by dragging and dropping activities and control flows from the Components editor. You then define these activities and the transitions between them using the Property Inspector.

Unless a task flow has already been created, ADF Mobile automatically generates a default unbounded task flow (`adfc-mobile-config.xml` file) when a new ADF Mobile AMX page is created. For an example of the unbounded task flow, see the `adfc-mobile-config.xml` file from a sample application called Navigation located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer.

For more information, see the "Creating ADF Task Flows" part of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

7.2.1 How to Create a Task Flow

You use the navigation diagrammer to declaratively create a task flow for your ADF Mobile AMX application feature. When you use the diagrammer, JDeveloper creates the XML metadata needed for navigation to work in your feature in the `ViewController-task-flow.xml` file (default).

Before you begin:

To design a task flow, the ADF Mobile application must include a View Controller project file (see Chapter 4, "Getting Started with ADF Mobile Application Development").

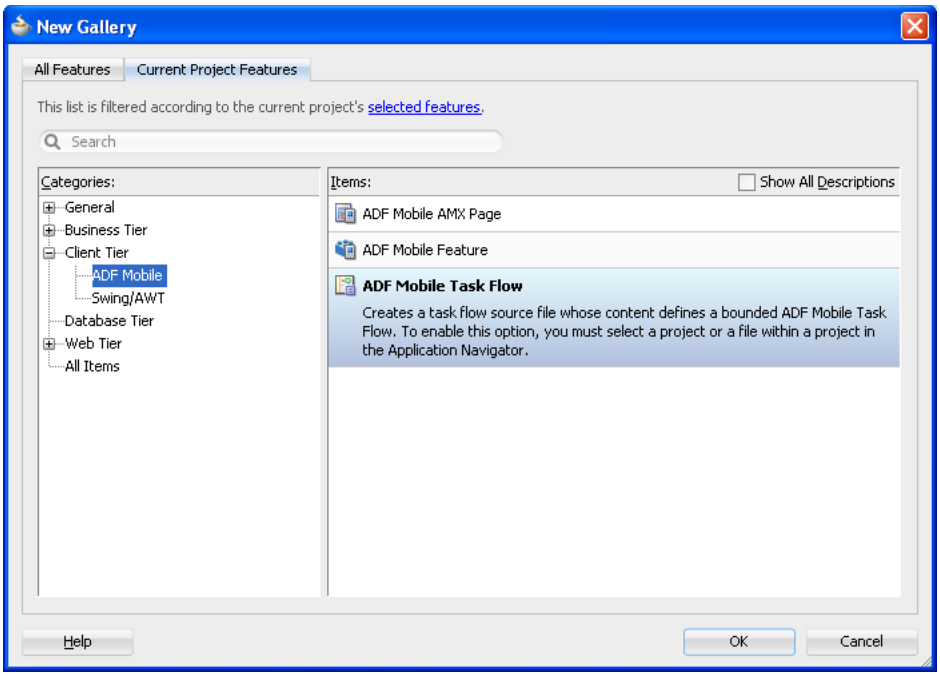
There are two ways to create a task flow in ADF Mobile:

- You can create a bounded task flow from the `admf-feature.xml` file. For more information, see Section 5.9.1, "How to Define the Application Content."
- You can use the New Gallery in JDeveloper. For more information, see "To create a task flow from the New Gallery:"

To create a task flow from the New Gallery:

1. From the top-level menu in JDeveloper, click **File**, and then select **New > From Gallery**.
2. In the **New Gallery**, expand the **Client Tier** node, select **ADF Mobile,** and then **ADF Mobile Task Flow** (see Figure 7–1). Click **OK**.

Figure 7–1 Creating New ADF Mobile Task Flow

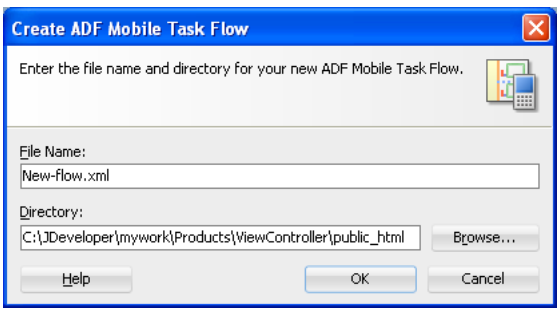


3. In the **Create ADF Mobile Task Flow** dialog (see Figure 7–2), specify the file name and location for your new task flow, and then click **OK** to open the new `<Name>-flow.xml` file in the navigation diagrammer that Figure 7–3 shows.

Note: Task flows should be created within the View Controller project of your ADF Mobile application.

Note: JDeveloper increments the number of the task flow according to the number of task flows that already exist in the same pattern.

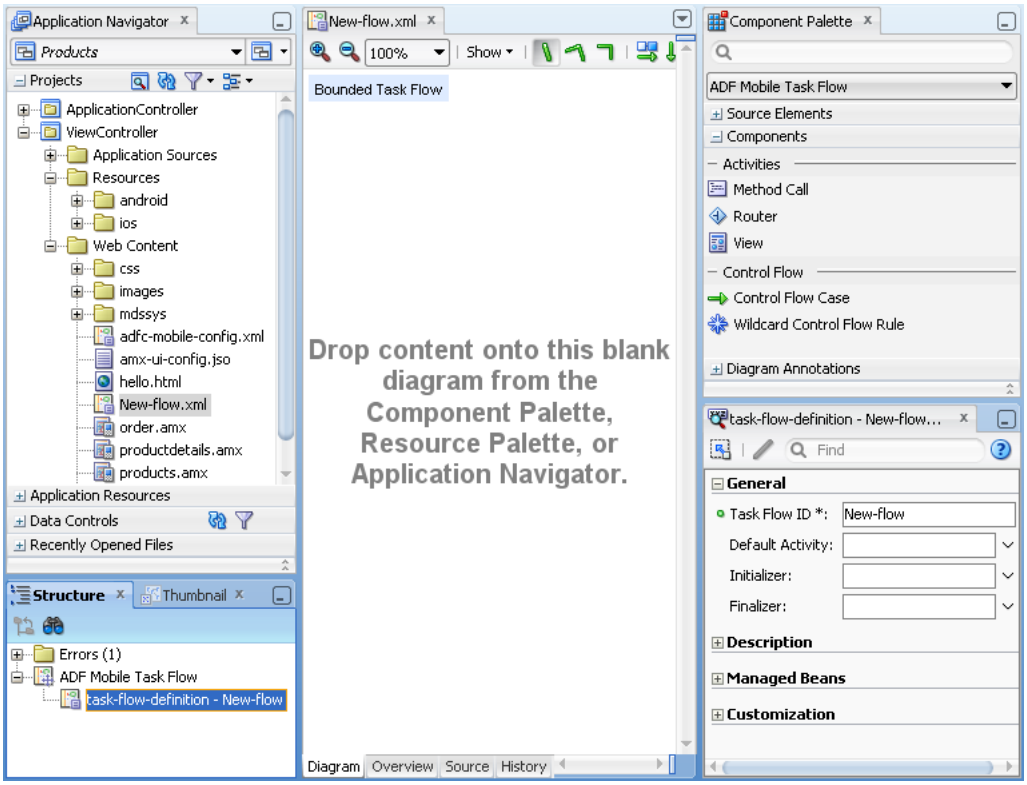
Figure 7–2 Create ADF Mobile Task Flow Dialog



4. In the **Component Palette**, select **ADF Mobile Task Flow** (see Figure 7–3), and then select the component you wish to use and drag it onto the diagram.

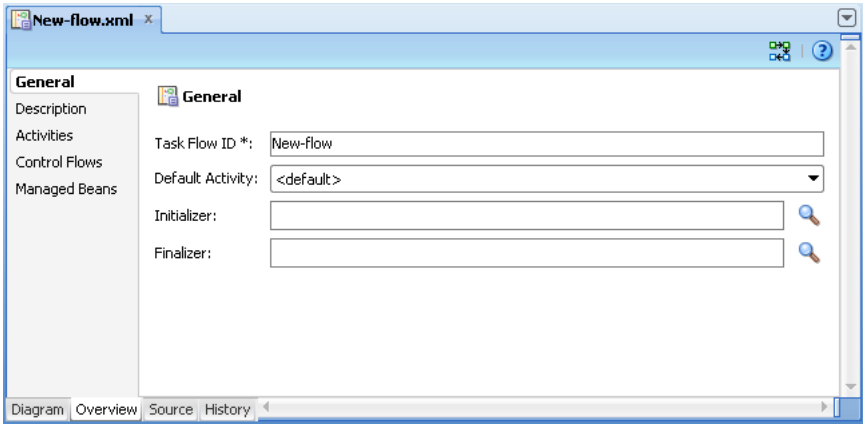
   Tip: If the Component Palette is not displayed, choose **View > Component Palette** from the main menu. By default, the Component Palette is displayed in the upper right-hand corner of JDeveloper.

Figure 7–3 New Blank Task Flow



JDeveloper creates a bounded task flow and redraws the diagram with the newly added component.

You can also open the **Overview** tab and use the overview editor to create navigation rules and navigation cases. Press F1 for details on using the overview editor to create navigation.



Additionally, you can manually add elements to the task flow file by directly editing the page in the Source editor. To open the file in the Source editor, click the **Source** tab.

Once the navigation for your ADF Mobile AMX feature is defined, you can create the pages and add the components that will execute the navigation.For more information about using navigation components on a page, see Section 7.2.8, "How to Enable Page Navigation Using Control Flow Case."

After you define the task flow for the ADF Mobile AMX application feature, you can double-click a view file to access the ADF Mobile AMX view. For more information, see Section 7.3, "Creating Views."

7.2.2 What You May Need to Know About Supported Activities and Control Flows

The ADF Mobile Task Flow designer supports a subset of ADF activities and control flows that are listed in Table 7–1.

Table 7–1 Supported Activities

Activity	Description
Method Call	Invokes a method (typically a method on a managed bean). You can place a method call activity anywhere in the control flow of an ADF Mobile AMX application feature to invoke logic based on control flow rules. For additional information, see the "Using Method Call Activities" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. You can also specify parameters that you pass into a method call that is included in a task flow. These include standard ADF parameters for a method call action in an ADF Mobile AMX task flow. When you use the designer to generate a method, it adds the required arguments and type. For more information, see the following sections of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*: ■ "Using Method Call Activities" ■ "How to Specify Method Parameters and Return Values" At run time, you can define parameters for a method call in a task flow and pass parameters into the method call itself for its usage. For more information on passing method call parameters, see the following sections of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*: ■ "Creating Complex Task Flows" ■ "Setting Parameter Values Using a Command Component" ■ "Working with Task Flow Activities" ■ "How to Specify Method Parameters and Return Values"
Router	Evaluates an Expression Language (EL) expression and returns an outcome based on the value of the expression. These outcomes can then be used to route control to other activities in the task flow. For more information, see the "Using Router Activities" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
View	Displays an ADF Mobile AMX page. For more information, see the "Using View Activities" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Table 7–1 (Cont.) Supported Activities

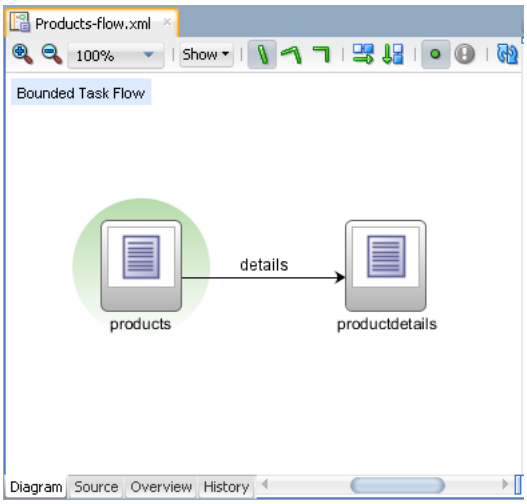
Activity	Description
Task flow call	Calls a bounded task flow from either an unbounded task flow or from a bounded task flow. While a task flow call activity allows you to call a bounded task flow located within the same ADF Mobile AMX application feature, you can also call a bounded task flow from a different ADF Mobile AMX application feature or from a Feature Archive file (FAR) that has been added an Oracle ADF library (see Section 5.12, "Working with Feature Archive Files"). The task flow call activity supports task flow input parameters and return values. For more information, see the "Using Task Flow Call Activities" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

7.2.3 What You May Need to Know About the ViewController-task-flow.xml File

The `ViewController-task-flow.xml` file is the ADF Mobile AMX counterpart to ADF's `task-flow-definition.xml` and enables you to design the interactions between views (ADF Mobile AMX pages) by dragging and dropping the ADF Mobile AMX task flow components from the Component Palette into the diagrammer.

Figure 7–4 shows a sample task flow file called `Products-flow.xml`. In this file, the control flow is directed from the `products` page to the `productdetails` page. To return to the `products` page from the `productdetails` page, the built-in `__back` navigation is used.

Figure 7–4 Task Flow File



7.2.4 What You May Need to Know About the ADF Mobile Task Flow Diagrammer

As illustrated in Figure 7–4, the task flow diagram and Component Palette display automatically after you create a task flow using the ADF Mobile Task Flow Creation utility. The task flow diagram is a visual editor onto which you can drag and drop activities and task flows from the Component Palette or from the Application navigator. For more information, see Section 7.2.5, "How to Add ADF Mobile Activities."

7.2.5 How to Add ADF Mobile Activities

As in ADF application development, you use the task flow diagrammer to drag and drop activities, views, and control flows.

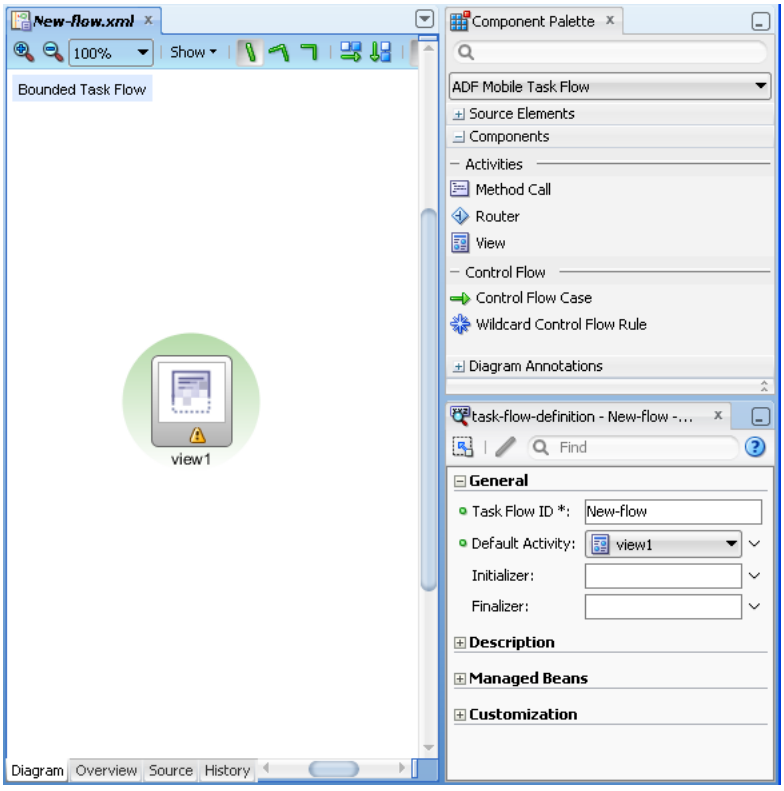
Before you begin:

You must select **ADF Mobile Task Flow** from the Component Palette, as Figure 7–5 shows.

To add an activity to an ADF Mobile task flow:

1. In the Application Navigator, double-click a task flow source file (such as `ViewController-task-flow.xml`) to display the task flow diagram and the Component Palette, as Figure 7–5 shows. The diagrammer displays the task flow editor. The Component Palette automatically displays the components available for an ADF Mobile task flow.
2. Drag an activity from the Component Palette onto the diagram. If you drag a view activity onto the diagram, you can invoke the Create ADF Mobile AMX File wizard (see Section 7.2.6, "How to Add View Activities").

Figure 7–5 The Diagrammer for the Task Flow Editor



Note: There is a default activity that is associated with each task flow.

7.2.6 How to Add View Activities

The view activity is associated in metadata with an actual ADF Mobile AMX page. You add a view activity by dragging and dropping it from the Component Palette. A

view activity displays an ADF Mobile AMX page. You can create an actual ADF Mobile AMX page by double-clicking the view activity in the Diagram window. You can also create a view activity by dragging and dropping an ADF Mobile AMX file in the Application navigator into the overview editor's Diagram tab.

7.2.7 How to Add a Wildcard Control Flow Rule

ADF Mobile task flows support the wildcard control flow rule, which represents a control flow `from-activity-id` that contains a trailing wildcard (`foo*`) or a single wildcard character. You can add a wildcard control flow rule by dragging and dropping it from the Component Palette. To configure your wildcard control flow rule, use the Property Inspector.

7.2.8 How to Enable Page Navigation Using Control Flow Case

You can create navigation using the Control Flow Case component, which identifies how control passes from one activity to the next. To create a control flow, select **Control Flow Case** from the Component Palette. Next, connect the control flow case to the source activity, and then to the destination activity. JDeveloper creates the following after you connect a source and target activity:

- `control-flow-rule`: Identifies the source activity using a `from-activity-id`.
- `control-flow-case`: Identifies the destination activity using a `to-activity-id`.

To define a control flow case directly in the ADF Mobile task flow diagram:

1. In the Application navigator, double-click a task flow source file to display the task flow diagram.
2. Select **Control Flow Case** from the Component Palette.
3. On the diagram, click a source activity and then a destination activity. JDeveloper adds the control flow case to the diagram. Each line that JDeveloper adds between an activity represents a control flow case. The `from-outcome` contains a value that can be matched against values specified in the action attribute of the UI components.
4. To change the `from-outcome`, select the text next to the control flow in the diagram. By default, this is a wildcard character.
5. To change the `from-activity-id` (the identifier of the source activity), or the `to-activity-id` (the identifier for the destination activity), drag either end of the arrow in the diagram to a new activity.

   For more information, see the "What Happens When You Create a Control Flow Rule" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

7.2.9 How to Specify Action Outcomes Using UI Components

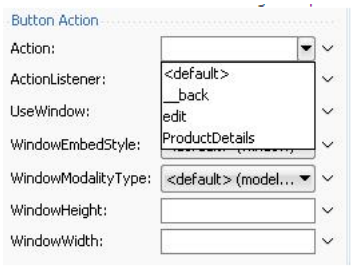
Using the Property Inspector, you can specify an action outcome by setting the `action` attribute of one of the following UI components to the corresponding control flow case `from-outcome` leading to the next task flow activity:

- Command Button (see Section 8.3.5, "How to Use Buttons")
- Command Link (see Section 8.3.6, "How to Use Links")

- List Item

You use the UI component's **Action** field (see Figure 7–6) to make a selection from a list of possible action outcomes defined in one or more task flow for a specific ADF Mobile AMX page.

Figure 7–6 Setting Actions



A **Back** action (`__back`) is automatically added to every list to enable navigation to the previously visited page.

> **Note:** An ADF Mobile AMX page can be referenced in both bounded and unbounded task flows, in which case actions outcomes from both task flows are included in the Action selection list.

7.2.10 How to Create and Reference Managed Beans

You can create and use managed beans in your ADF Mobile application to store additional data or execute custom code. You can use JDeveloper's usual editing mechanism to reference managed beans and create references to them for applicable fields. For more information, see the "Using a Managed Bean in a Fusion Web Application" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Figure 7–7 shows the **Edit** option for an `action` property in the Property Inspector. You click this option to invoke the **Edit Property** dialog that Figure 7–8 shows.

Figure 7–7 Edit Dialog

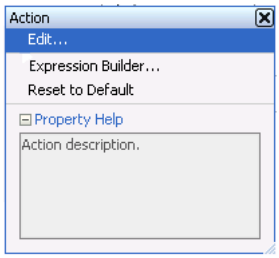


Figure 7–8 Edit Property Dialog for Action

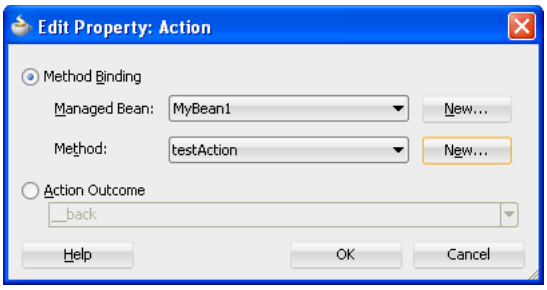


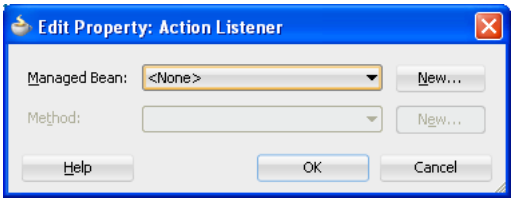
Table 7–5 lists ADF Mobile AMX attributes for which the Edit option in the Property Inspector is available.

Table 7–2 Editable Attributes

Property	Element
binding	actionListener
action	commandButton
actionListener	commandButton
action	commandLink
actionListener	commandLink
action	listItem
actionListener	listItem
valueChangeListener	inputDate
valueChangeListener	inputNumberSlider
valueChangeListener	inputText
valueChangeListener	selectBooleanCheckbox
valueChangeListener	selectBooleanSwitch
valueChangeListener	selectManyCheckbox
valueChangeListener	selectOneButton
valueChangeListener	selectOneChoice
valueChangeListener	selectOneRadio
rangeChangeListener	listView
selectionListener	listView

Clicking **Edit** for all other properties invokes a similar dialog, but without the Action Outcome option, as Figure 7–9 shows.

Figure 7–9 Edit Property Dialog for Action Listener



The preceding dialogs demonstrate that you can either create a managed bean, or select an available action outcome for the action property.

The **Action Outcome** list shown in Figure 7–8 contains the action outcomes from all task flows to which a specific ADF Mobile AMX page belongs. In addition, this list contains a `__back` navigation outcome to go back to the previously visited page (see Section 7.2.9, "How to Specify Action Outcomes Using UI Components" for more information). If a page is not part of any task flow, the only available outcome in the Action Outcome list is `__back`. When you select one of the available action outcomes and click OK, the action property value is updated with the appropriate EL expression, such as the following for a `commandButton`:

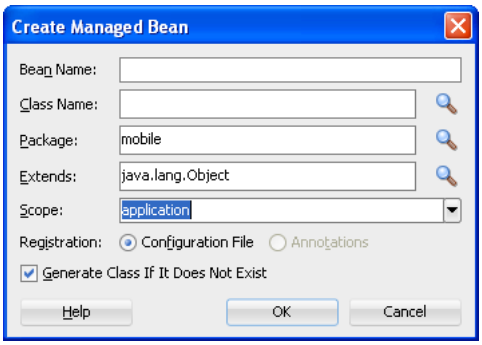
```
<amx:commandButton action="goHome"/>
```

The **Method Binding** option (see Figure 7–8) allows you to either create a new managed bean class, or select an existing one.

To create a new managed bean class:

1. Click **New** next to the Managed Bean field to open the Create Managed Bean dialog that Figure 7–10 shows.

Figure 7–10 Create Managed Bean Dialog



   ADF Mobile supports the following scopes:

   - `application`
   - `view`
   - `pageFlow`

   When you declare a managed bean to an ADF Mobile application or the ADF Mobile AMX application feature, the managed bean is instantiated and identified in the proper scope, and the bean's properties are resolved and its methods are called through EL. For more information, see Section 9.2, "Understanding EL Support."

2. Provide the managed bean and class names, and then click **OK**.

   Example 7–1 shows the newly created managed bean class. The task flow that this ADF Mobile AMX page is part of is updated to reference the bean.

Example 7–1 New Managed Bean Class

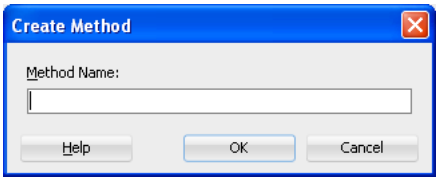
```
<managed-bean id="__3">
   <managed-bean-name>MyBean</managed-bean-name>
   <managed-bean-class>mobile.MyBean</managed-bean-class>
   <managed-bean-scope>application</managed-bean-scope>
```

```
</managed-bean>
```

> **Note:** If a given ADF Mobile AMX page is part of bounded as well as unbounded task flows, both of these task flows are updated with the managed bean entry.

3. Click **New** next to the **Method** field (see Figure 7–8 and Figure 7–9) to open the **Create Method** dialog that Figure 7–11 shows. Use this dialog to provide the managed bean method name.

Figure 7–11 Create Method Dialog



Upon completion, the selected property value is updated with the appropriate EL expression, such as the following for a `commandButton`.:

```
<amx:commandButton action="#{MyBean.getMeHome}"/>
```

The managed bean class is also updated to contain the newly created method, as Example 7–2 shows.

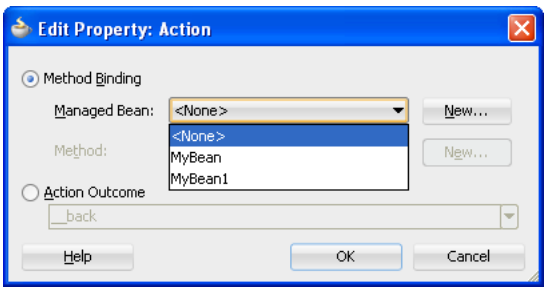
Example 7–2 New Method in Managed Bean Class

```
package mobile;

public class MyBean {
   public MyBean() {
   }

   public String getMeHome() {
      // Add event code here...
      return null;
   }
}
```

To select an existing managed bean:

1. Make a selection from the **Managed Bean** list that Figure 7–12 shows.

Figure 7–12 Selecting Managed Bean



Similar to the action outcomes, the Managed Bean list is populated with managed beans from all task flows that this ADF Mobile AMX page is part of.

Note: If the ADF Mobile AMX page is not part of any task flow, you can still create a managed bean.

For more information, see the following:

- Section 9.2.4.2, "ADF Managed Beans"
- JavaDemo, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer.

7.2.11 How to Specify the Page Transition Style

By defining the page transition style on the task flow, you can specify how ADF Mobile AMX pages transition from one view to another. The behavior of your ADF Mobile AMX page at transition can be as follows:

- fading in
- sliding in from left
- sliding in from right
- sliding up from bottom
- sliding down from top
- flipping up from bottom
- flipping down from top
- flipping from left
- flipping from right
- none

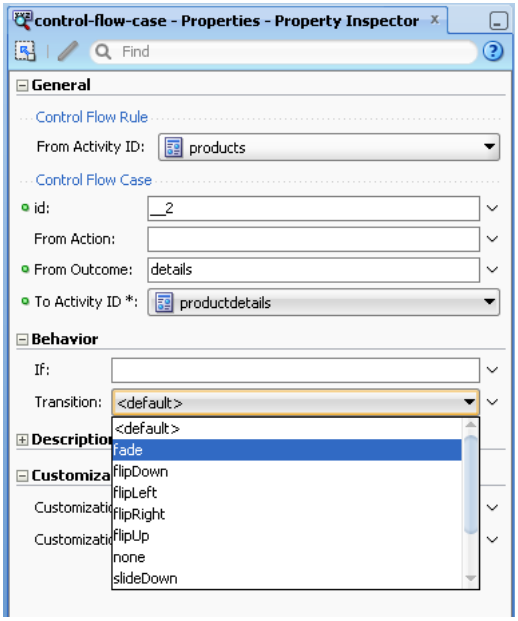
You set the transition style by modifying the `transition` attribute of the `control-flow-case` (Control Flow Case component), as Example 7–3 shows.

Example 7–3 Setting Transition Style

```
<control-flow-rule id="__1">
   <from-activity-id>products</from-activity-id>
   <control-flow-case id="__2">
      <from-outcome>details</from-outcome>
      <to-activity-id>productdetails</to-activity-id>
      <transition>fade</transition>
   </control-flow-case>
</control-flow-rule>
```

In the Property Inspector, the `transition` attribute is located under **Behavior**, as Figure 7–13 shows. The default transition style is `slideLeft`.

Figure 7–13 Setting Transition Style in Property Inspector



7.3 Creating Views

You can start creating the ADF Mobile AMX view by doing the following:

- Getting familiar with the ADF Mobile AMX page structure
- Using the preview
- Dragging and dropping components into the ADF Mobile AMX page
- Adding data controls to a view

7.3.1 How to Work with ADF Mobile AMX Pages

An ADF Mobile AMX page is represented by an XML file similar to a JSPX file in Oracle ADF Faces.

7.3.1.1 Interpreting the ADF Mobile AMX Page Structure

The following is a basic structure of the ADF Mobile AMX file:

```
<amx:view>
   <amx:panelPage id="pp1">
      <amx:facet name="header">
         <amx:outputText id="ot1" value="Welcome"/>
      …
      </amx:facet>
   </amx:panelPage>
</amx:view>
```

With the exception of data visualization components (see Section 8.5, "Providing Data Visualization"), UI elements are declared under the `<amx>` namespace.

For more information, see Section 7.3.1.3, "What Happens When You Create an ADF Mobile AMX Page."

7.3.1.2 Creating ADF Mobile AMX Pages

ADF Mobile AMX files are contained in the View Controller project of the ADF Mobile application. You create these files using the Create ADF Mobile AMX Page dialog.

ADF Mobile offers two alternative ways of creating an ADF Mobile AMX page:

- From the New Gallery
- From an existing task flow

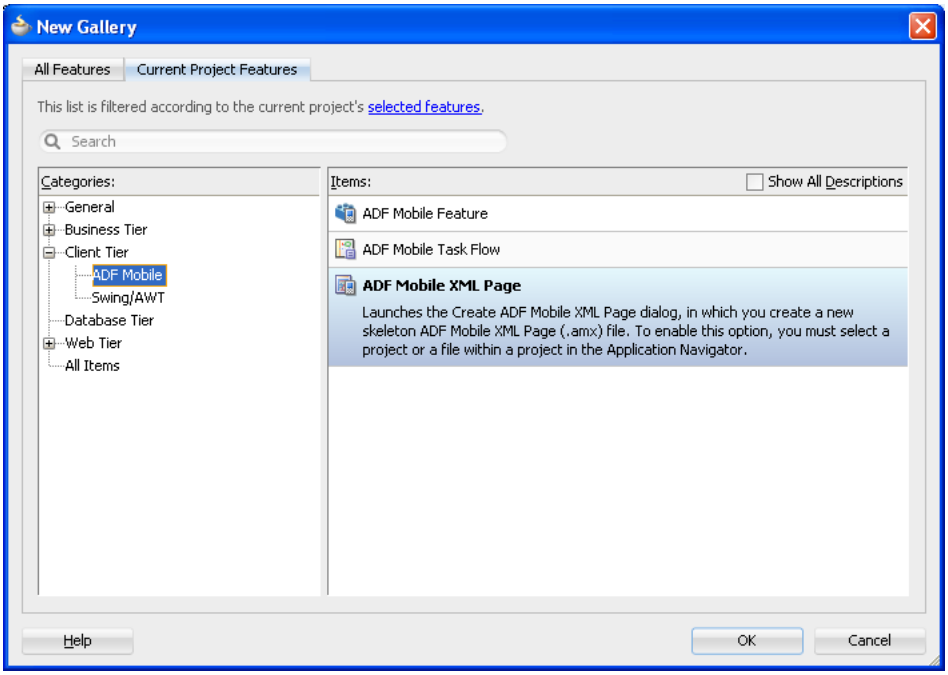
Before you begin:

To create an ADF Mobile AMX page, the ADF Mobile application must include a View Controller project file (see Chapter 4, "Getting Started with ADF Mobile Application Development").

To create an ADF Mobile AMX page from the New Gallery:

1. From the top-level menu in JDeveloper, click **File**, and then select **New > From Gallery**.
2. In the **New Gallery**, expand the **Client Tier** node, select **ADF Mobile,** and then **ADF Mobile AMX Page** (see Figure 7–14). Click **OK**.

Figure 7–14 Creating ADF Mobile AMX Page

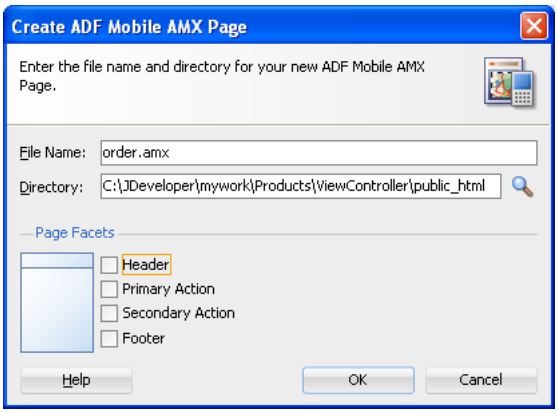


3. In the Create ADF Mobile AMX Page dialog, enter a name and, if needed, a location for your new file, as Figure 7–15 shows.
4. Optionally, you may select which facets your new ADF Mobile AMX page will include as a part of the page layout:
   - Header
   - Primary
   - Secondary
   - Footer

For more information, see Section 7.3.1.3, "What Happens When You Create an ADF Mobile AMX Page" and Section 8.2.6, "How to Use a Facet Component."

Note that when you select or deselect a facet, the image representing the page changes dynamically to reflect the changing appearance of the page.

Figure 7–15 Create ADF Mobile AMX Page Dialog



Note: ADF Mobile persists your facet selection and applies it to each subsequent invocation of the Create ADF Mobile AMX Page dialog.

5. Click **OK** on the Create ADF Mobile AMX Page dialog.

To create an ADF Mobile AMX page from a View Component of the Task Flow:

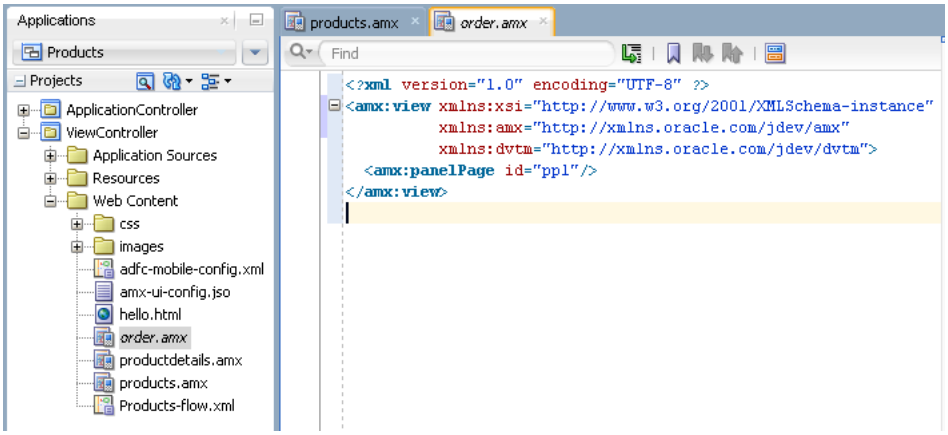
1. Open a task flow file in the diagrammer (see Figure 7–5, "The Diagrammer for the Task Flow Editor", Section 7.2.1, "How to Create a Task Flow" and Section 7.2.4, "What You May Need to Know About the ADF Mobile Task Flow Diagrammer")
2. Double-click a View component of the task flow to open the Create ADF Mobile AMX Page dialog that Figure 7–15 shows, and then enter a name and, if needed, a location for your new file. Click **OK**.

7.3.1.3 What Happens When You Create an ADF Mobile AMX Page

When you use the Create ADF Mobile AMX Page dialog to create an ADF Mobile AMX page, JDeveloper creates the physical file and adds it to the `Web Content` directory of the View Controller project.

In the Application navigator that Figure 7–16 shows, the `Web Content` node contains a newly created ADF Mobile AMX file called `order.amx`.

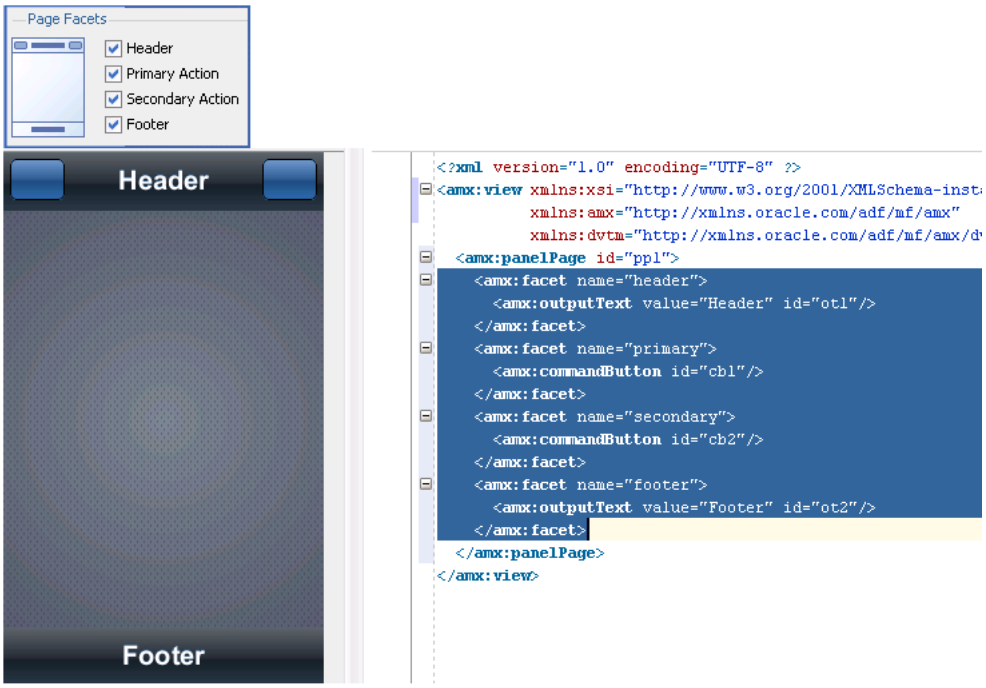
Figure 7–16 ADF Mobile AMX File in Application Navigator



JDeveloper also adds the code necessary to import the component libraries and display a page. This code is illustrated in the Source editor shown in Figure 7–16.

Figure 7–17 shows how the Preview pane and the generated ADF Mobile AMX code would look like if you selected all facet types listed in the Page Facet section of the Create ADF Mobile AMX Page dialog when creating the page.

Figure 7–17 ADF Mobile AMX Page With All Facets



In the page created with all the facets selected (see Figure 7–17), note the following:

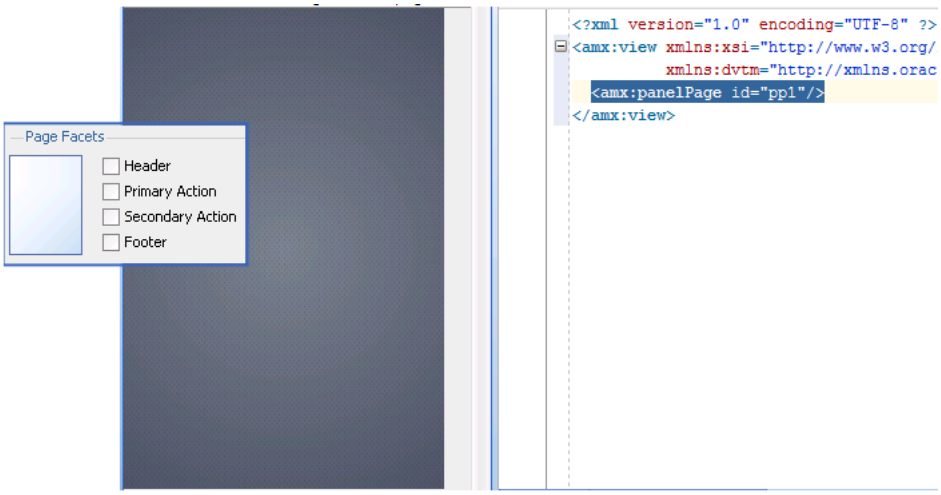
- The header is created with an Output Text component because this component is typically used for the page title.
- The primary and secondary actions are created with Button components because it is a typical pattern.
- Since there is no single dominant pattern for the footer, it is created with an Output Text component by default because that component is used in some

patterns and it prevents JDeveloper from generating the initial code with audit violation.

- Adding either the primary or secondary action without adding the header facet still causes the header section to appear in the Page Facets section of Create ADF Mobile AMX Page dialog.

Figure 7–18 shows the Page Facet section of the Create ADF Mobile AMX Page dialog without any facets selected, the Preview pane, and the generated ADF Mobile AMX code (which is also shown in Figure 7–16).

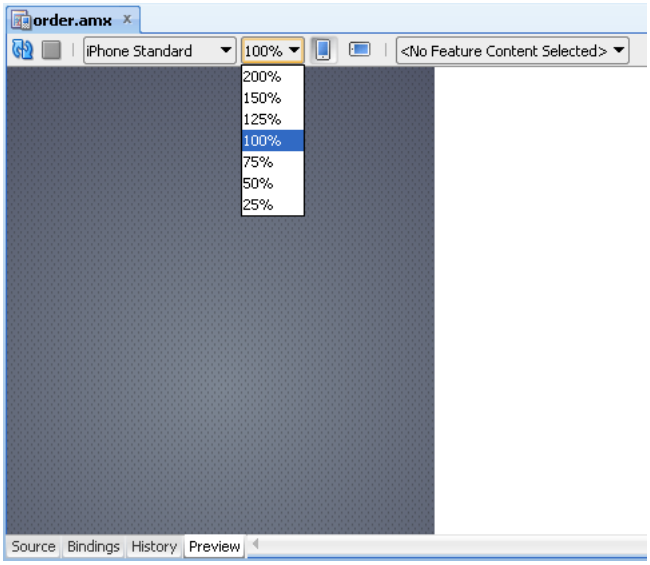
Figure 7–18 ADF Mobile AMX Page Without Facets



7.3.1.4 Using UI Editors

When the page is first displayed in JDeveloper, it is displayed in the Source editor. To view the page in a WYSIWYG environment, use the Preview pane (accessed by clicking the **Preview** tab). Figure 7–19 shows the Preview tab selected for a newly created ADF Mobile AMX page called `order.amx`. This page is blank because it has not yet been populated with ADF Mobile AMX UI components or data controls.

Figure 7–19 The Preview Pane for Newly Created Page



Using the Preview pane's toolbar that Figure 7–19 shows, you can do the following:

- Refresh the display of the ADF Mobile AMX page by clicking **Refresh Page**.
- Stop loading of the page by clicking **Stop Loading Page**.
- Modify the form factor for the page by selecting a different form factor from the dropdown list. For more information on form factors, see Section 3.3.1.1, "Configuring the Environment for Form Factors."
- Modify the scaling of the display by selecting a different percentage value from the dropdown list. Since mobile device displays can be of various sizes and densities, the Preview pane allows you to see the effect of scaling on your ADF Mobile AMX pages.

> **Note:** Scaling is available for both Portrait and Landscape mode.

- Change orientation for the display to portrait and landscape by selecting **Show Portrait Orientation** or **Show Landscape Orientation** respectively.
- Select the feature content for your ADF Mobile AMX page form the dropdown list of available application features. By default, <No Feature Content Selected> is displayed.

To view the source for the page in the Source editor, click the **Source** tab that Figure 7–16, "ADF Mobile AMX File in Application Navigator" shows. The Structure window, located in the lower left-hand corner of JDeveloper (shown in Figure 7–16 and Figure 7–19), provides a hierarchical view of the page. For more information, see Section 7.3.2.2, "Using the Preview."

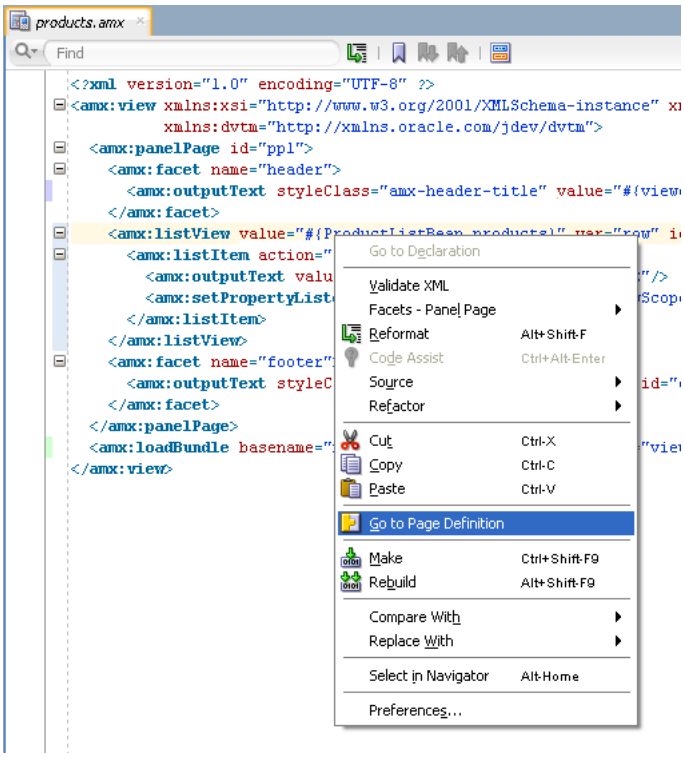
7.3.1.5 Accessing the Page Definition

ADF Mobile AMX supports JDeveloper's Go to Page Definition functionality that enables you to navigate to the page definition of an ADF Mobile AMX page by using a context menu that allows you to locate and edit the binding information quickly.

You can invoke the context menu that contains the Go to Page Definition option from the following:

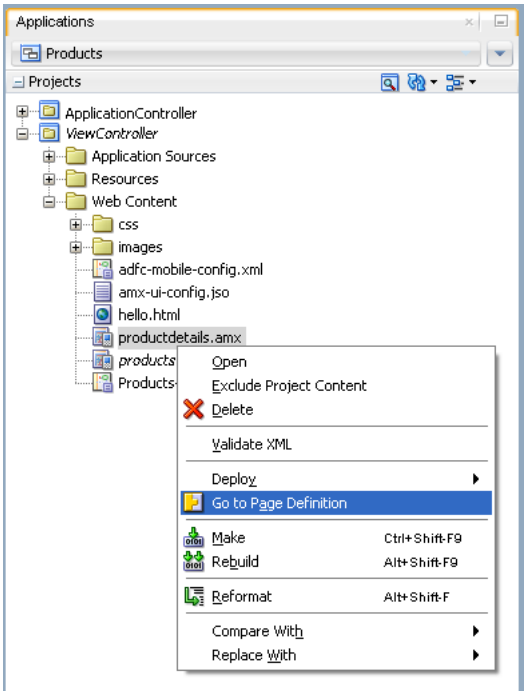
- Source editor, as Figure 7–20 shows.

Figure 7–20 Go to Page Definition from Source Editor



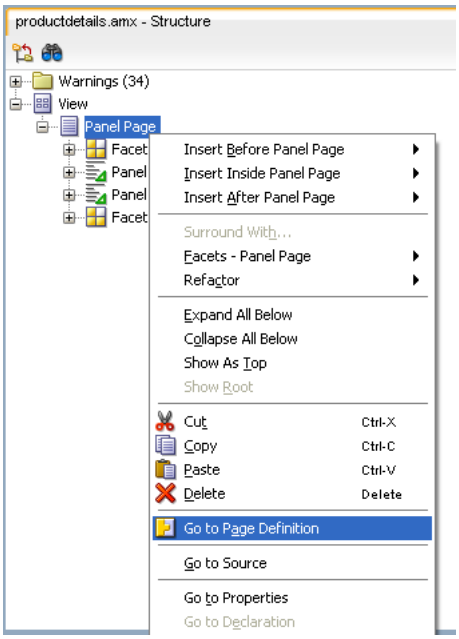
- Application navigator, as Figure 7–21 shows.

Figure 7–21 Go to Page Definition from Application Navigator



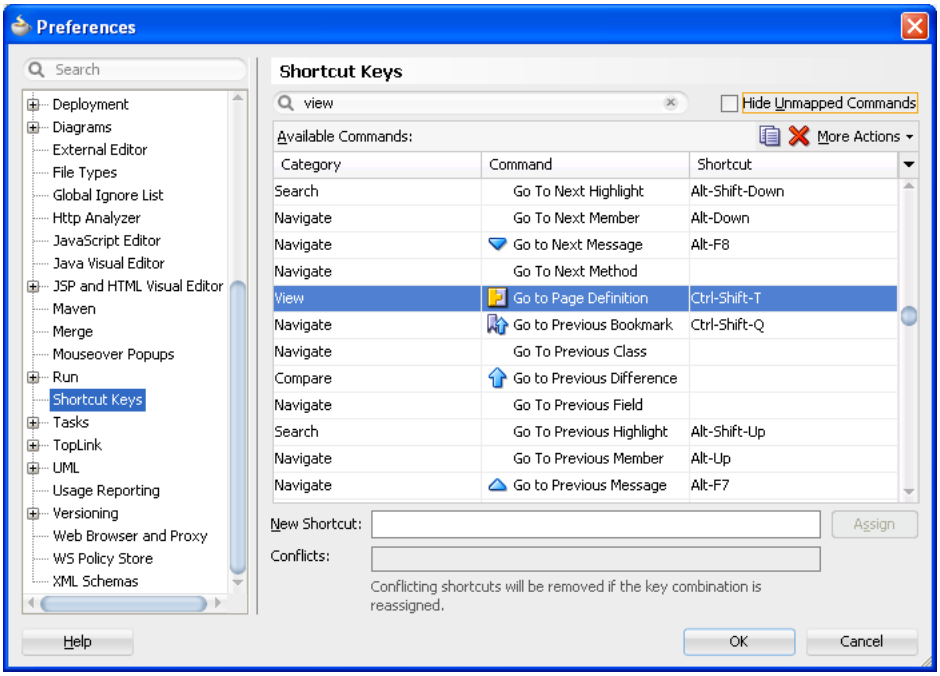
- Structure pane, as Figure 7–22 shows.

Figure 7–22 Go to Page Definition from Structure Pane



In addition, you can open the Page Definition file using the **Go to Page Definition** shortcut key defined under **Tools > Preferences** on the main menu, as Figure 7–23 shows.

Figure 7–23 Opening Page Definition from Preferences



7.3.2 How to Add UI Components and Data Controls to an ADF Mobile AMX Page

After you create an ADF Mobile AMX page, you can start adding ADF Mobile AMX UI components and data controls to your page.

7.3.2.1 Adding UI Components

You can use the Component Palette to drag and drop ADF Mobile AMX components onto the page. JDeveloper then adds the necessary declarative page code and sets certain values for component attributes.

The Component Palette displays ADF Mobile AMX components by categories (see Figure 7–24):

- General Controls
- Text and Selection
- Data Views
- Layout, with the following subcategories:
  - Interactive Containers and Headers
  - Secondary Windows
  - Core Structure
- Operations, with the following subcategories:
  - Behavior
  - Listeners
  - Validators and Converters

For information on adding and using specific components, see Section 8.3, "Creating and Using UI Components."

Before you begin:

The ADF Mobile application must include a View Controller project, which may or may not contain an ADF Mobile AMX page or ADF Mobile AMX page task flow from which to create a page.

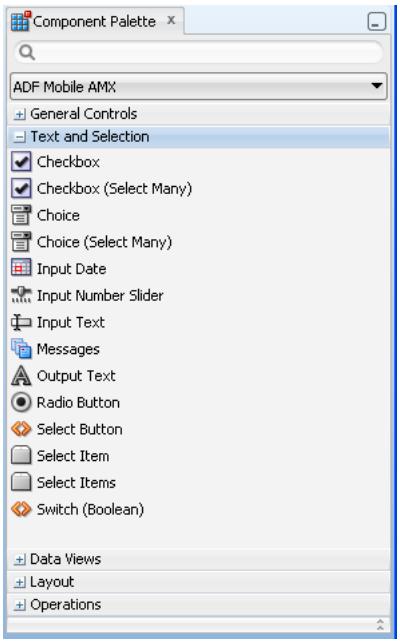
As described in Section 7.3.1.2, "Creating ADF Mobile AMX Pages," you can invoke the Create ADF Mobile AMX Page dialog by double-clicking a view icon in a task flow diagram or by selecting **Client Tier > ADF Mobile > ADF Mobile AMX Page** from the New Gallery.

To add UI components to a page:

1. Open an ADF Mobile AMX page in the Source editor (default).
2. In the Component Palette, use the menu to choose **ADF Mobile AMX**, as Figure 7–24 shows.

   Tip: If the Component Palette is not displayed, choose **View > Component Palette** from the main JDeveloper menu. By default, the Component Palette is displayed in the upper right-hand corner of JDeveloper.

Figure 7–24 ADF Mobile AMX Component Palette



3. Select the component you wish to use, and then drag and drop it onto the Source editor or Structure window. You cannot drop components onto the Preview pane.

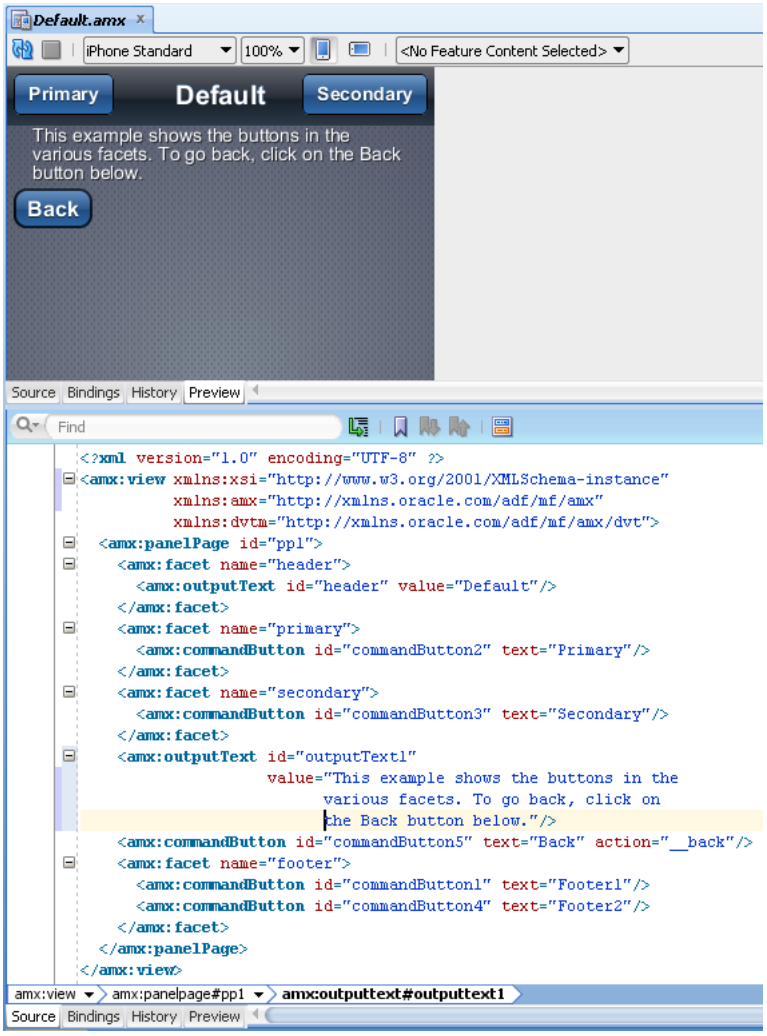
> **Note:** When building an ADF Mobile AMX page, you can only drop UI components into UI containers such as, for example, a Panel Group Layout.

JDeveloper redraws the page in the Preview pane with the newly added component.

7.3.2.2 Using the Preview

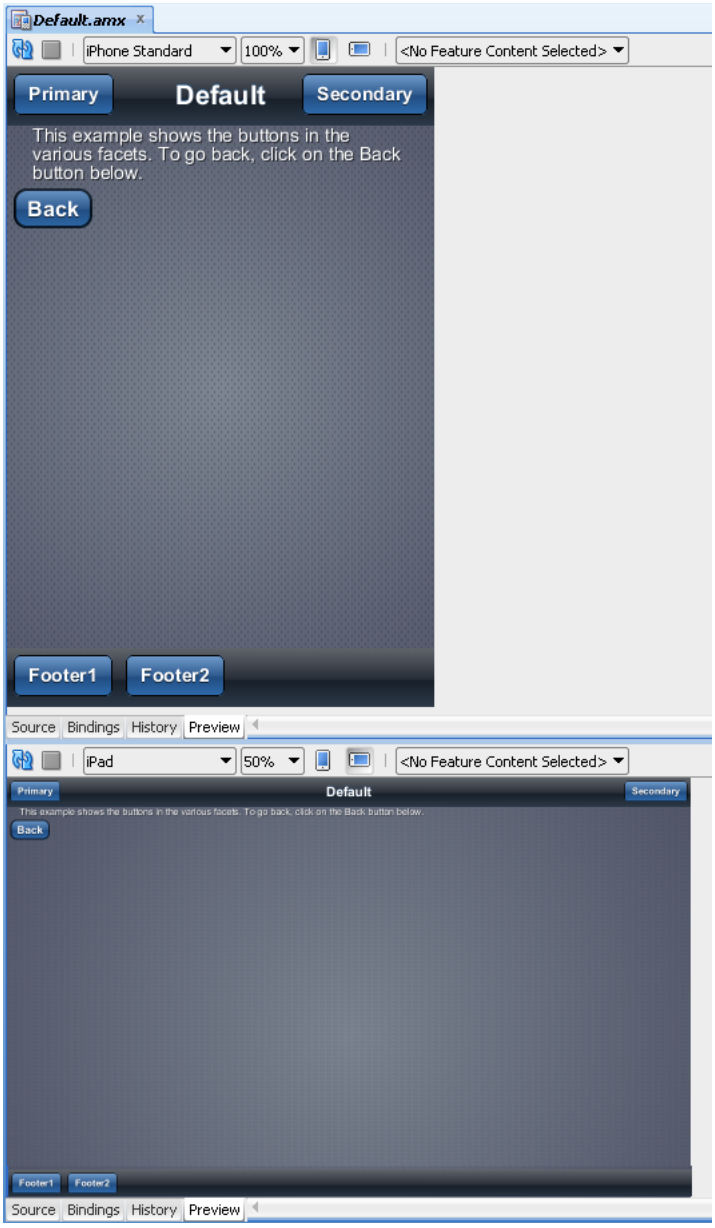
JDeveloper's preview provides WYSIWYG support for both the iOS and Android platforms when you build the user interface using ADF Mobile AMX files. As illustrated in Figure 7–25, splitting a view while adding the ADF Mobile AMX components to the ADF Mobile AMX file enables you to see both the code view through the Source editor and a UI view through the Preview pane. As a result, you can modify the source and get instant feedback in terms of the look and feel of that application on both the iOS and Android platforms.

Figure 7–25 Splitting Design and Source Views



In addition to being able to see the design and source views simultaneously, you can also open and work with multiple design views at the same time, as well as set each one to a different platform and screen size. By opening a combination of design views for different devices, you can develop applications simultaneously for different platforms and form factors using different orientation. Figure 7–26 shows a split screen with iPhone Standard on the top and iPad with 50% scaling on the bottom. You can split the Preview pane using the default split functionality of JDeveloper.

Figure 7–26 Multiple Design Views



Note: An ADF Mobile AMX page is rendered even for an invalid ADF Mobile AMX file. Errors are indicated by the error icon on a component. By moving the mouse over the error icon, you can view the error details.

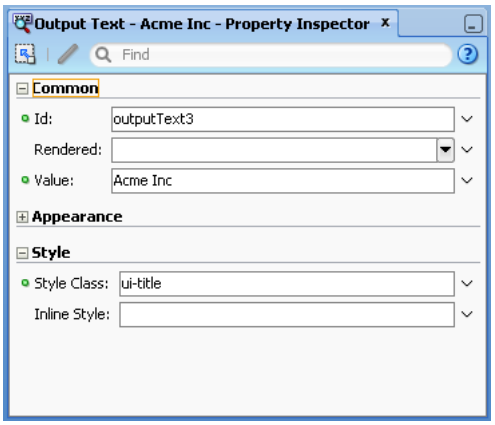
7.3.2.3 Configuring UI Components

Once you drop UI components onto a page, you can use the Property Inspector (displayed by default at the bottom right of JDeveloper) to set attribute values for each component.

Tip: If the Property Inspector is not displayed, choose **Window > Reset to Default Factory Settings** from JDeveloper's main menu.

Figure 7–27 shows the Property Inspector displaying the attributes for an Output Text component.

Figure 7–27 The Property Inspector



To set component attributes:

1. Select the component for which you want to set attributes. You can select the component in the Structure window or you can select its tag directly in the Source editor.
2. In the Property Inspector, expand the section that contains the attribute you wish to set.

   Tip: Some attributes are displayed in more than one section. Entering or changing the value in one section will also change it in any other sections. You can search for an attribute by entering the attribute name in the search field at the top of the Property Inspector.

3. In the Property Inspector, either enter values directly into the fields, or if the field contains a list, use that list to select a value. You can also use the list to the right of the field, which launches a popup containing tools you can use to set the value. These tools are either specific property editors (opened by choosing **Edit**) or the Expression Builder, which you can use to create EL expressions for the value (opened by choosing **Expression Builder**). For more information about using the Expression Builder, see the "Creating EL Expressions" section in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

When you use the Property Inspector to set or change attribute values, JDeveloper automatically changes the page source for the attribute to match the entered value.

Tip: You can always change attribute values by directly editing the page in the Source editor. To view the page in the Source editor, click the **Source** tab at the bottom of the page.

7.3.2.4 Adding Data Controls to the View

You can create databound UI components in an ADF Mobile AMX view by dragging data control elements from the Data Controls panel and dropping them into either the Structure window or the Source editor. When you drag an item from the Data Controls panel to either of these places, JDeveloper invokes a context menu of default UI components available for the item that you dropped. When you select the desired

UI component, JDeveloper inserts it into an ADF Mobile AMX page. In addition, JDeveloper creates the binding information in the associated page definition file. If such file does not exist, then JDeveloper creates one. ADF Mobile provides a visual indicator for dropping data controls to inform you of the location of the new data control.

> **Note:** A data control can only be dropped at a location allowed by the underlying XML schema.

Depending on the approach you take, you can insert different types of data controls into the Structure window of an ADF Mobile AMX page.

Dropping an attribute of a collection lets you create various input and output components. You can also create Buttons and Links by dropping a data control operation on a page.

The respective action listener is added in the ADF Mobile AMX Button for each of these operations. The EL expression in the `actionListener` is the same as the one created on the drop of an operation into an Oracle ADF Faces application.

The data control attributes and operations can be dropped as one or more of the following ADF Mobile AMX UI components (see Section 8.3, "Creating and Using UI Components"):

- Button
- Link
- Input Date
- Input Date with Label
- Input Text
- Input Text with Label
- Output Text
- Output Text with Label
- Iterator
- List Item
- List View
- Select Boolean Checkbox
- Select Boolean Switch
- Select One Button
- Select One Choice
- Select One Radio
- Select Many Checkbox
- Select Many Choice
- Convert Date Time
- Convert Number
- Form

- Read Only Form
- Parameter Form

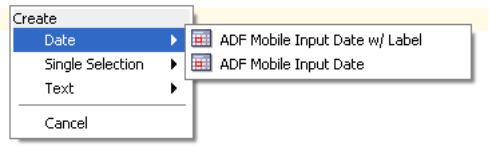
The following Date and Number types are supported:

- `java.util.Date`
- `java.sql.Timestamp`
- `java.sql.Date`
- `java.lang.Number`
- `java.lang.Integer`
- `java.lang.Long`
- `java.lang.Float`
- `java.lang.Double`

7.3.2.4.1 Dragging and Dropping Attributes If your ADF Mobile AMX page already contains a Panel Form Layout component or does not require to have all the fields added, you can drop individual attributes from a data control. Depending on the attributes types, different data binding menu options are provided, as follows:

- Date: this category provides options for creating ADF Mobile Input Date and ADF Mobile Input Date with Label controls. Figure 7–28 shows the context menu for adding date controls that appears when you drag an attribute from the Data Controls panel into the Structure window of an ADF Mobile AMX page.

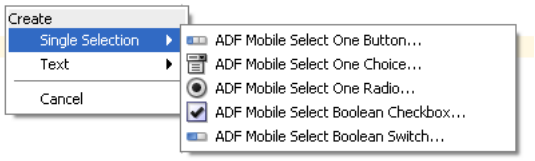
Figure 7–28 Context Menu for Date Controls



- Single Selection: this category provides options for creating the following controls:.
  - ADF Mobile Select One Button
  - ADF Mobile Select One Choice
  - ADF Mobile Select One Radio
  - ADF Mobile Select Boolean Checkbox
  - ADF Mobile Select Boolean Switch

  Figure 7–29 shows the context menu for adding selection controls that appears when you drag an attribute from the Data Controls panel into the Structure window of an ADF Mobile AMX page.

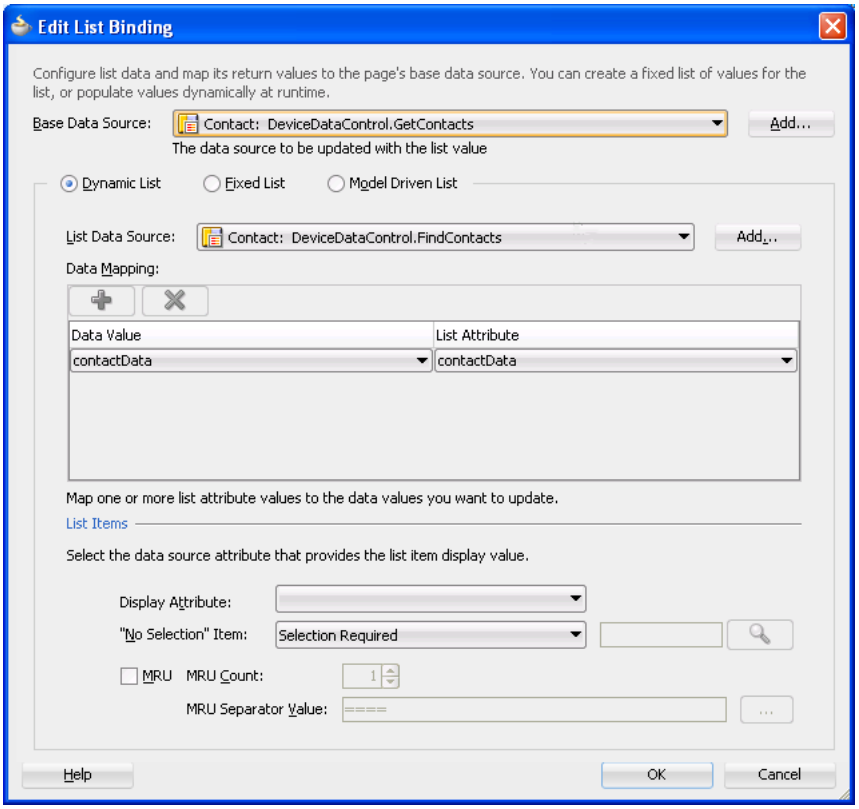
Figure 7–29 Context Menu for Selection Controls



If you are working with an existing ADF Mobile AMX page and you select **ADF Mobile Select One Button** or **ADF Mobile Select One Choice** option, an appropriate version of the Edit List Binding dialog is displayed (see Figure 7–30). If you drop a control onto a completely new ADF Mobile AMX page, the Edit Action Binding dialog opens instead. After you click OK, the Edit List Binding dialog opens.

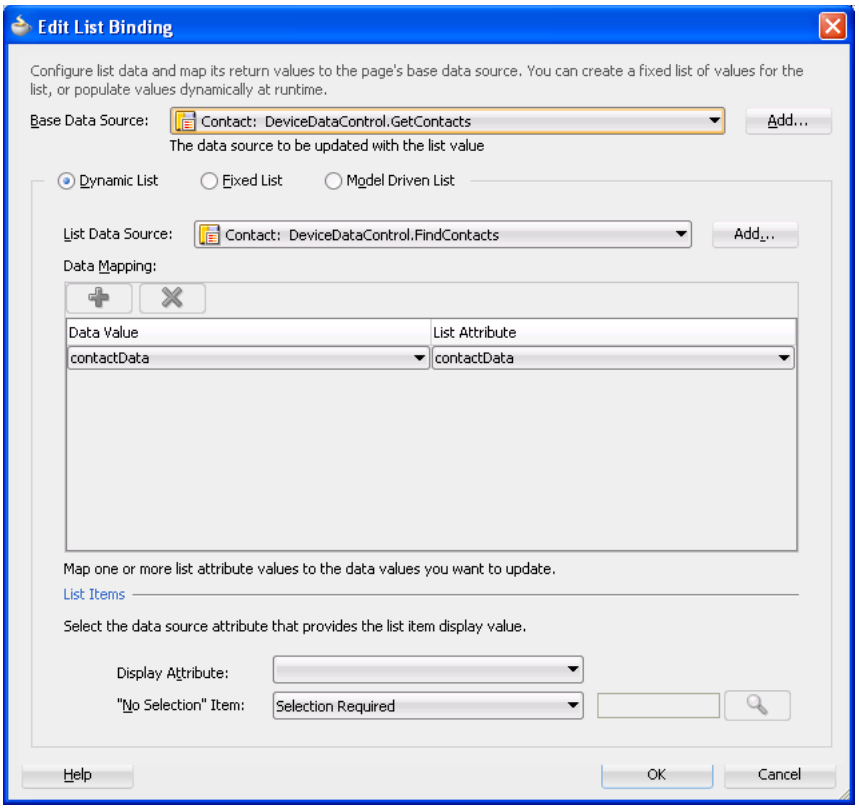
Note: The Edit List Binding or Edit Boolean Binding dialog appears every time you drop any data control attributes as any of the single selection or boolean selection components, respectively.

Figure 7–30 Edit List Binding Dialog for Select One Button and Choice Controls



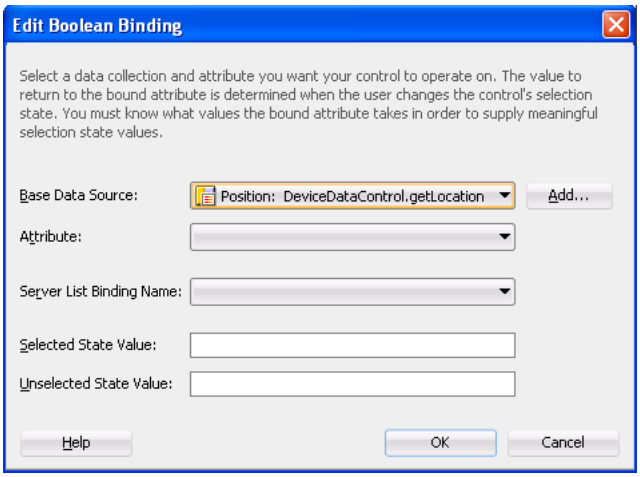
If you select **ADF Mobile Select One Radio** option, another version of the Edit List Binding dialog is displayed, as shown in Figure 7–31.

Figure 7–31 Edit List Binding Dialog for Select One Radio Control



If you select **ADF Mobile Select Boolean Checkbox** or **ADF Mobile Select Boolean Switch** option, another version of the Edit List Binding dialog is displayed, as shown in Figure 7–32.

Figure 7–32 Edit List Binding Dialog for Select Boolean Checkbox and Switch Controls

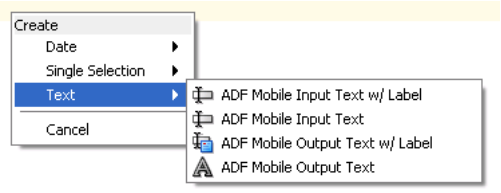


- Text: this category provides options for creating the following controls:
  - ADF Mobile Input Text
  - ADF Mobile Input Text with Label
  - ADF Mobile Output Text

– ADF Mobile Output Text with Label

Figure 7–29 shows the context menu for adding selection controls that appears when you drag an attribute from the Data Controls panel into the Structure window of a ADF Mobile AMX page.

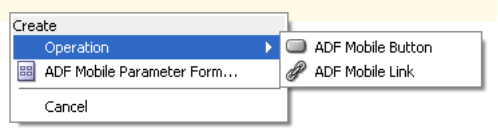
Figure 7–33 Context Menu for Text Controls



7.3.2.4.2 Dragging and Dropping Operations In addition to attributes, you can drag and drop operations or custom methods. Depending on the type of operation or method, different data binding menu options are provided, as follows:

- Operation: this category is for data control operations. It provides the following options (see Figure 7–34):
  - ADF Mobile Button
  - ADF Mobile Link
  - ADF Mobile Parameter Form

Figure 7–34 Context Menu for Operations

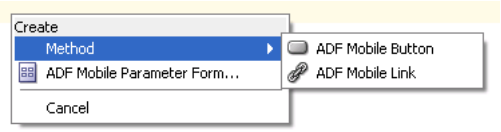


Note: If you drop an operation or a method as a child of the List View control, the context menu does not appear and the List Item is created automatically because no other valid control can be dropped as a direct child of the List View control. JDeveloper creates a binding similar to the following for the generated List Item:

```
<amx:listItem actionListener="#{bindings.getLocation.execute}"/>
```

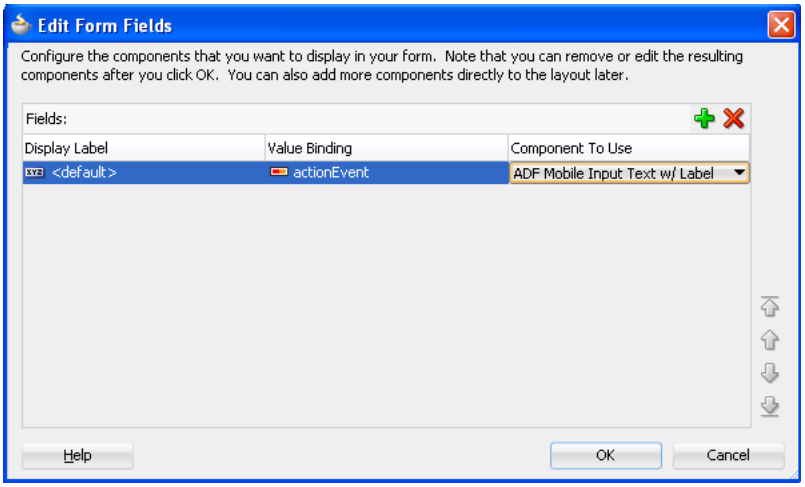
- Method: this category is for custom methods. It provides the following options (see Figure 7–34):
  - ADF Mobile Button
  - ADF Mobile Link
  - ADF Mobile Parameter Form

Figure 7–35 Context Menu for Methods



The ADF Mobile Parameter Form option allows you to choose the method or operation arguments to be inserted in the form, as well as the respective controls for each of the arguments (see Figure 7–36).

Figure 7–36 Edit Form Fields Dialog



The following data bindings are generated after you select the appropriate options in the Edit Form Fields dialog:

```
<amx:panelFormLayout id="pfl1">
   <amx:inputText id="it1"
                  value="#{bindings.actionEvent.inputValue}"
                  label="#{bindings.actionEvent.hints.label}"
                  required="#{bindings.actionEvent.hints.mandatory}"
                  maximumLength="#{bindings.actionEvent.hints.precision}"/>
</amx:panelFormLayout>
<amx:commandButton id="cb1"
                   actionListener="#{bindings.cleanInputParameter.execute}"
                   text="cleanInputParameter"
                   disabled="#{!bindings.cleanInputParameter.enabled}"/>
```

For more information about generated bindings, see Section 7.3.2.4.4, "What You May Need to Know About Generated Bindings."

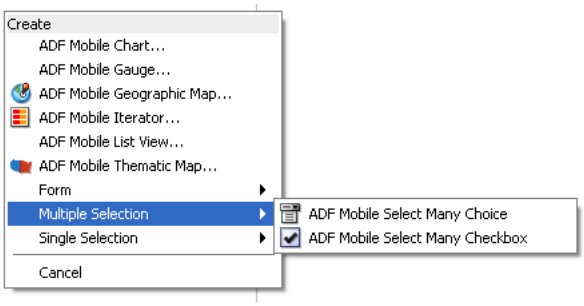
The following are supported control types for the ADF Mobile Parameter Form:

- ADF Mobile Input Date
- ADF Mobile Input Date with Label
- ADF Mobile Input Text
- ADF Mobile Input Text with Label
- ADF Mobile Output Text with Label

7.3.2.4.3 Dragging and Dropping Collections You can drag and drop collections. Depending on the type of collection, different data binding menu options are provided, as follows:

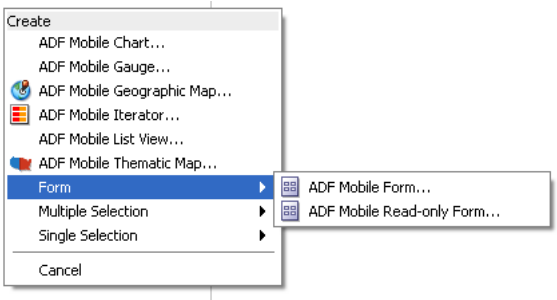
- **Multiple Selection**: this category provides options for creating multiple selection controls. The following controls can be created under this category (see Figure 7–37):
  - ADF Mobile Select Many Checkbox
  - ADF Mobile Select Many Choice

Figure 7–37 Context Menu for Multiple Selection Controls



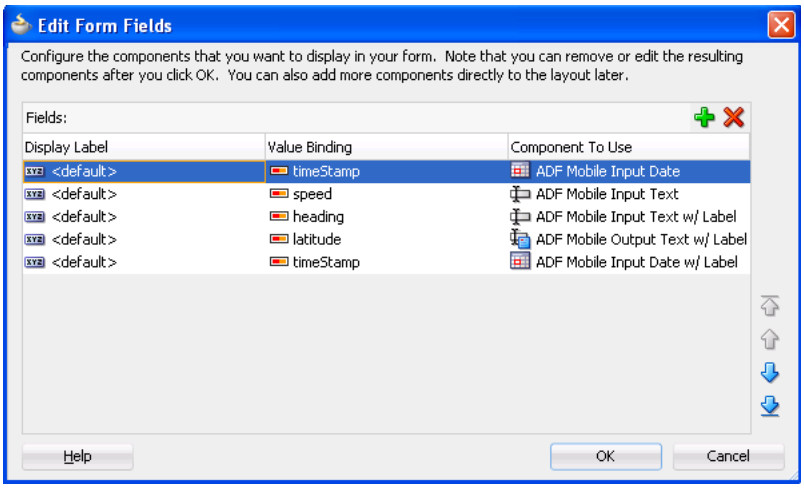
- **Form**: this category is used to create the ADF Mobile AMX Panel Form controls. The following controls can be created under this category (see Figure 7–38):
  - ADF Mobile Form
  - ADF Mobile Read-only Form

Figure 7–38 Context Menu for Form Controls



If you select **ADF Mobile Form** as the type of the form you want to create, a JDeveloper wizard is invoked that lets you choose the fields to be inserted in the form, as well as the respective controls for each of the fields (see Figure 7–39).

Figure 7–39 Edit Form Fields Dialog for ADF Mobile Form



The following data bindings are generated after you select the appropriate options in the Edit Form Fields dialog:

```
<amx:panelFormLayout id="pfl1">
   <amx:inputDate id="id1"
                  value="#{bindings.timeStamp.inputValue}"
                  required="#{bindings.timeStamp.hints.mandatory}">
      <amx:convertDateTime pattern="#{bindings.timeStamp.format}"/>
   </amx:inputDate>
   <amx:inputText id="it1"
                  value="#{bindings.speed.inputValue}"
                  simple="true"
                  required="#{bindings.speed.hints.mandatory}"
                  maximumLength="#{bindings.speed.hints.precision}"/>
   <amx:inputText id="it2"
                  value="#{bindings.heading.inputValue}"
                  label="#{bindings.heading.hints.label}"
                  required="#{bindings.heading.hints.mandatory}"
                  maximumLength="#{bindings.heading.hints.precision}"/>
   <amx:panelLabelAndMessage id="plm1"
                             label="#{bindings.latitude.hints.label}">
      <amx:outputText id="ot1" value="#{bindings.latitude.inputValue}"/>
   </amx:panelLabelAndMessage>
   <amx:inputDate id="it3"
                  value="#{bindings.timeStamp.inputValue}"
                  label="#{bindings.timeStamp.hints.label}"
                  required="#{bindings.timeStamp.hints.mandatory}">
      <amx:convertDateTime pattern="#{bindings.timeStamp.format}"/>
   </amx:inputDate>
</amx:panelFormLayout>
```

For more information about generated bindings, see Section 7.3.2.4.4, "What You May Need to Know About Generated Bindings."

The following are supported controls for ADF Mobile Form:

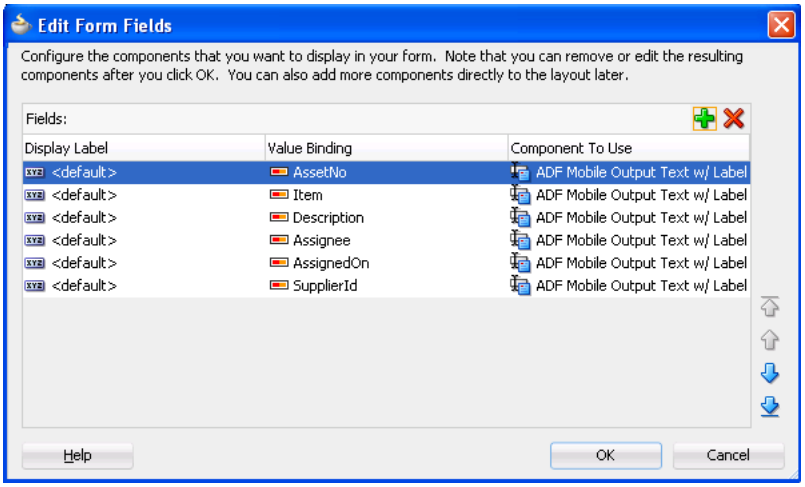
- ADF Mobile Input Date
- ADF Mobile Input Date with Label
- ADF Mobile Input Text
- ADF Mobile Input Text with Label

– ADF Mobile Output Text with Label

Note: Since ADF Mobile Output Text is not a valid Panel Form Layout child element as defined by the ADF Mobile schema, it is not supported.

If you select **ADF Mobile Read-only Form** as the type of the form you want to create, a JDeveloper wizard is invoked that lets you choose the fields to be inserted in the form, as well as the respective controls for each of the fields (see Figure 7–40).

Figure 7–40 Edit Form Fields Dialog for ADF Mobile Read-only Form



The following data bindings are generated after you select the appropriate options in the Edit Form Fields dialog:

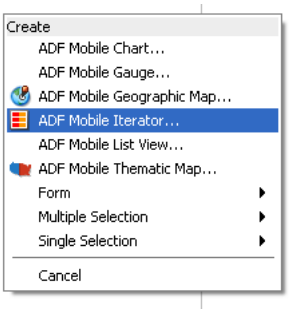
```
<amx:panelFormLayout id="pfl1">
   <amx:panelLabelAndMessage id="plm1" label="#{bindings.AssetNo.hints.label}">
      <amx:outputText id="ot1" value="#{bindings.AssetNo.inputValue}"/>
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage id="plm2" label="#{bindings.Item.hints.label}">
      <amx:outputText id="ot2" value="#{bindings.Item.inputValue}"/>
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage id="plm3"
                             label="#{bindings.Description.hints.label}">
      <amx:outputText id="ot3" value="#{bindings.Description.inputValue}"/>
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage id="plm4"
                             label="#{bindings.Assignee.hints.label}">
      <amx:outputText id="ot4" value="#{bindings.Assignee.inputValue}"/>
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage id="plm5"
                             label="#{bindings.AssignedOn.hints.label}">
      <amx:outputText id="ot5" value="#{bindings.AssignedOn.inputValue}"/>
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage id="plm6"
                             label="#{bindings.SupplierId.hints.label}">
      <amx:outputText id="ot6" value="#{bindings.SupplierId.inputValue}"/>
   </amx:panelLabelAndMessage>
</amx:panelFormLayout>
```

For more information about generated bindings, see Section 7.3.2.4.4, "What You May Need to Know About Generated Bindings."

The ADF Mobile Read-only Form only supports the ADF Mobile Output Text with Label control.

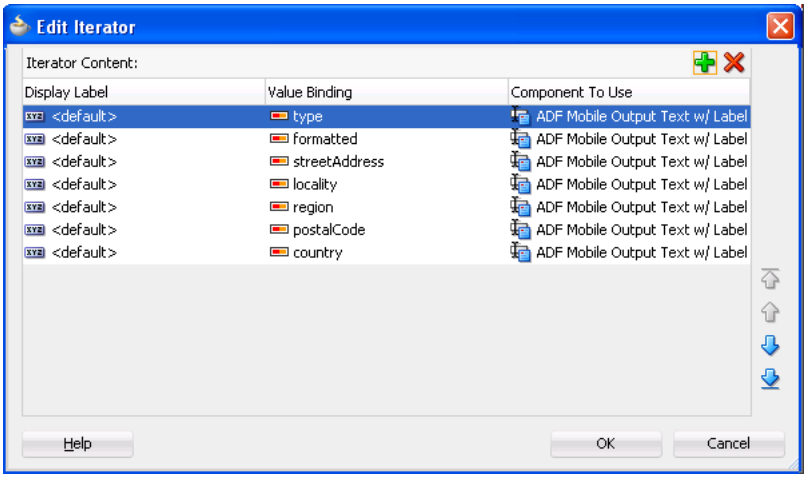
- **Iterator**: this provides an option for creating the ADF Mobile AMX Iterator with child components (see Figure 7–41).

Figure 7–41 Context Menu for Iterator Control



If you select **ADF Mobile Iterator** as the type of the control to create, a JDeveloper wizard is invoked that lets you choose the fields to be inserted in the iterator, as well as the respective controls for each of the fields, with **ADF Mobile Output Text w/Label** being a default selection (see Figure 7–42).

Figure 7–42 Edit Dialog for ADF Mobile Iterator



The following data bindings are generated after you select the appropriate options in the Edit Iterator dialog:

```
<amx:iterator var="row"
              value="#{bindings.addresses.collectionModel}"
              id="iterator1">
   <amx:panelLabelAndMessage label="#{bindings.addresses.hints.type.label}"
                             id="panelLabelAndMessage6">
      <amx:outputText value="#{row.type}" id="outputText6"/>
   </amx:panelLabelAndMessage>
   <amx:panelLabelAndMessage label=
                             "#{bindings.addresses.hints.formatted.label}"
                             id="panelLabelAndMessage2">
```

```
      <amx:outputText value="#{row.formatted}"/>
    </amx:panelLabelAndMessage>
    <amx:panelLabelAndMessage label=
                              "#{bindings.addresses.hints.streetAddress.label}"
                              id="panelLabelAndMessage5">
      <amx:outputText value="#{row.streetAddress}" id="outputText5"/>
    </amx:panelLabelAndMessage>
    <amx:panelLabelAndMessage label="#{bindings.addresses.hints.locality.label}"
                              id="panelLabelAndMessage4">
      <amx:outputText value="#{row.locality}" id="outputText4"/>
    </amx:panelLabelAndMessage>
    <amx:panelLabelAndMessage label="#{bindings.addresses.hints.region.label}"
                              id="panelLabelAndMessage7">
      <amx:outputText value="#{row.region}" id="outputText7"/>
    </amx:panelLabelAndMessage>
    <amx:panelLabelAndMessage label=
                              "#{bindings.addresses.hints.postalCode.label}"
                              id="panelLabelAndMessage1">
      <amx:outputText value="#{row.postalCode}" id="outputText1"/>
    </amx:panelLabelAndMessage>
    <amx:panelLabelAndMessage label="#{bindings.addresses.hints.country.label}"
                              id="panelLabelAndMessage3">
      <amx:outputText value="#{row.country}" id="outputText3"/>
    </amx:panelLabelAndMessage>
  </amx:iterator>
```

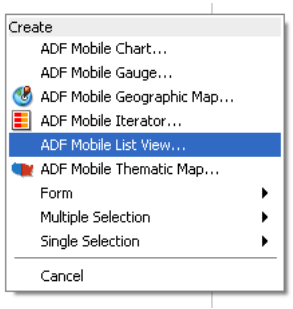
For more information about generated bindings, see Section 7.3.2.4.4, "What You May Need to Know About Generated Bindings."

The following are supported controls for ADF Mobile Iterator:

- ADF Mobile Input Text
- ADF Mobile Input Text with Label
- ADF Mobile Output Text
- ADF Mobile Output Text with Label

- **List View**: this provides an option for creating the ADF Mobile AMX List View with child components (see Figure 7–43).

Figure 7–43 Context Menu for List View Control



If you select **ADF Mobile List View** as the type of the control to create, the **ListView Gallery** opens that allows you to choose a specific layout for the List View, as Figure 7–44 shows.

Figure 7–44 ListView Gallery

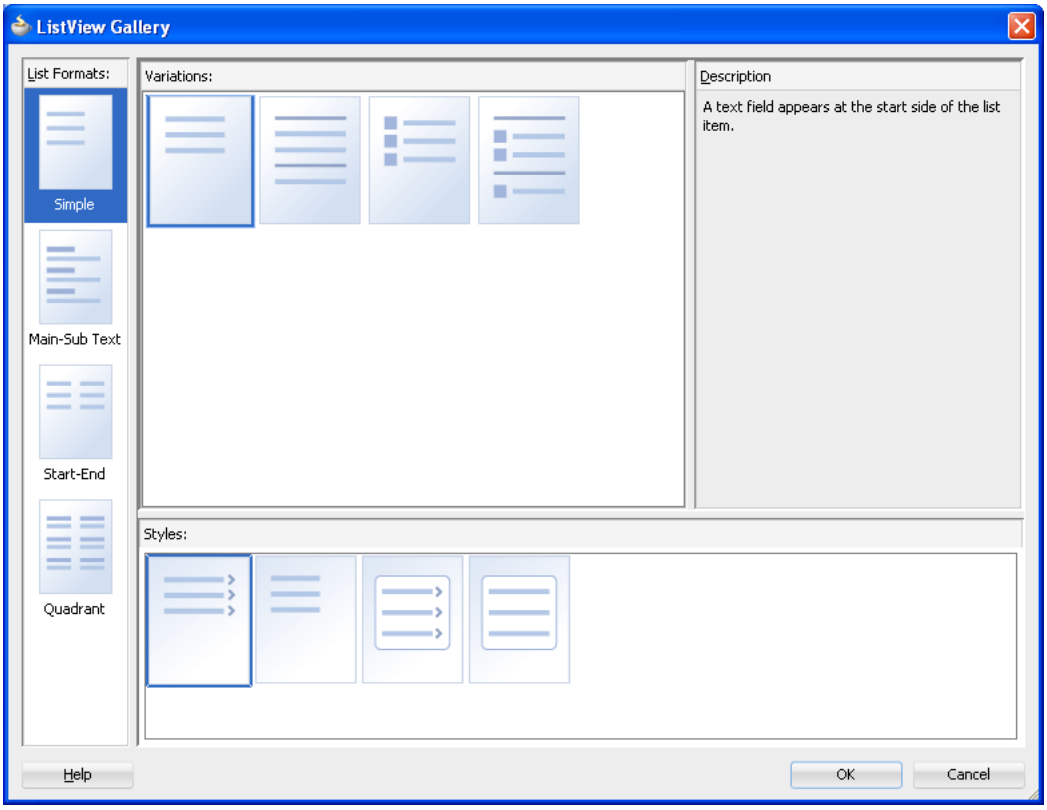


Table 7–3 lists the supported **List Formats** that are displayed in the ListView Gallery.

Table 7–3 List Formats

Format	Element Row Values
Simple	▪ Text
Main-Sub Text	▪ Main Text ▪ Subordinate Text
Start-End	▪ Start Text ▪ End Text
Quadrant	▪ Upper Start Text ▪ Upper End Text ▪ Lower Start Text ▪ Lower End Text

The **Variations** presented in the ListView Gallery (see Figure 7–44) for a selected list format consist of options to add either dividers, a leading image, or both:

– Selecting a variation with a leading image adds an Image row to the List Item Content table (see Figure 7–45).

– Selecting a variation with a divider defaults the Divider Attribute field to the first attribute in its list rather than the default No Divider value, and populates the Divider Mode field with its default value of All.

The **Styles** options presented in the ListView Gallery (see Figure 7–44) allow you to suppress chevrons, use an inset style list, or both:

- The selections do not modify any state in the Edit List View dialog (see Figure 7–45). They only affect the generated ADF Mobile AMX markup.
- Selecting a style with the inset list sets the `adfmf-listView-insetList` style class on the `listView` element in the generated ADF Mobile AMX markup.
- Selecting a style without chevrons sets the `showLinkIcon` attribute on the `listItem` element to `false` in the generated ADF Mobile AMX markup.

The following is an example of the Simple format with the inset list and chevrons:

```
<amx:listView var="row"
              value="#{bindings.employees.collectionModel}"
              fetchSize="#{bindings.employees.rangeSize}"
              styleClass="adfmf-listView-insetList"
              id="listView2">
   <amx:listItem id="li2">
      <amx:outputText value="#{row.employeename}" id="ot3"/>
   </amx:listItem>
</amx:listView>
```

The ListView Gallery's **Description** pane is updated based on the currently selected Variation. The format will include a brief description of the main style, followed by the details of the selected variation. Four main styles with four variations on each provide sixteen unique descriptions detailed in Table 7–4.

Table 7–4 List View Variations and Styles

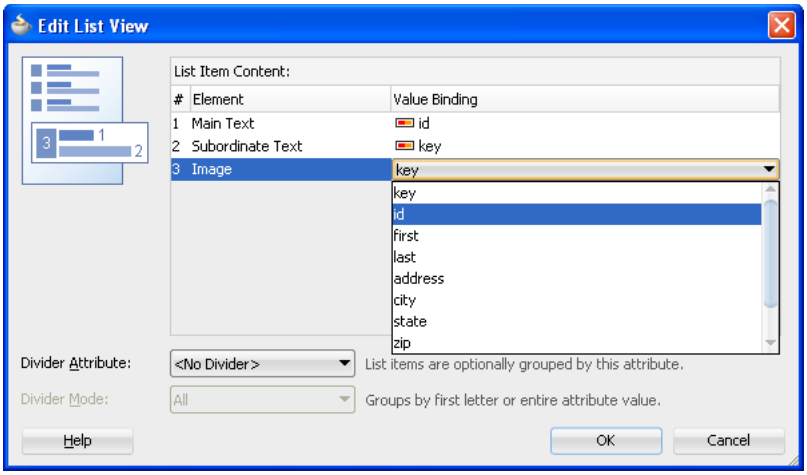
List Format	Variation	Description
Simple	Basic	A text field appears at the start side of the list item."
Simple	Dividers	A text field appears at the start side of the list item, with items grouped by dividers."
Simple	Images	A text field appears at the start side of the list item, following a leading image.
Simple	Dividers and Images	A text field appears at the start side of the list item, following a leading image. The list items are grouped by dividers.
Main-Sub Text	Basic	A prominent main text field appears at the start side of the list item with subordinate text below.
Main-Sub Text	Dividers	A prominent main text field appears at the start side of the list item with subordinate text below. The list items are grouped by dividers.
Main-Sub Text	Images	A prominent main text field appears at the start side of the list item with subordinate text below, following a leading image.
Main-Sub Text	Dividers and Images	A prominent main text field appears at the start side of the list item with subordinate text below, following a leading image. The list items are grouped by dividers.
Start-End	Basic	Text fields appear on each side of the list item
Start-End	Dividers	Text fields appear on each side of the list item, with the items grouped by dividers.
Start-End	Images	Text fields appear on each side of the list item, following a leading image.

Table 7–4 (Cont.) List View Variations and Styles

List Format	Variation	Description
Start-End	Dividers and Images	Text fields appear on each side of the list item, following a leading image. The list items are grouped by dividers.
Quadrant	Basic	Text fields appear in the four corners of the list item.
Quadrant	Dividers	Text fields appear in the four corners of the list item, with items grouped by dividers.
Quadrant	Images	Text fields appear in the four corners of the list item, following a leading image.
Quadrant	Dividers and Images	Text fields appear in the four corners of the list item, following a leading image. The list items are grouped by dividers.

After you make your selection from the ListView Gallery and click **OK**, the **Edit List View** wizard is invoked that lets you create the contents of a List Item by mapping binding attributes to the elements of the selected List View format, as Figure 7–45 shows.

Figure 7–45 Edit Dialog for ADF Mobile List View



When completing the dialog that Figure 7–45 shows, consider the following:

- The image at the start reflects the main content elements from the selected List View format and provides a mapping from the schematic representation to the named elements in the underlying table.
- The read-only cells in the Element column derive from the selected List View format.
- The editable cells in the Value Binding column are based on the data control node that was dropped.
- The List Item is generated as either an Output Text or Image component, depending on whichever is appropriate for the particular element.
- Since the number of elements (rows) is predetermined by the selected List View format, rows cannot be added or removed.
- The order of elements cannot be modified.

– The default value of the Divider Attribute field is No Divider, in which case the Divider Mode field is disabled. If you select value other than the default, then you need to specify Divider Mode parameters, as Figure 7–46 and Figure 7–47 show.

Figure 7–46 Specifying Divider Attribute

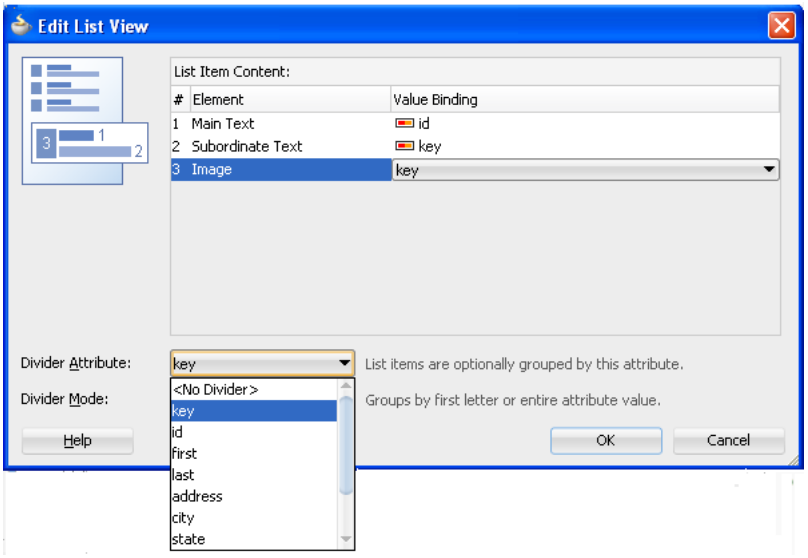
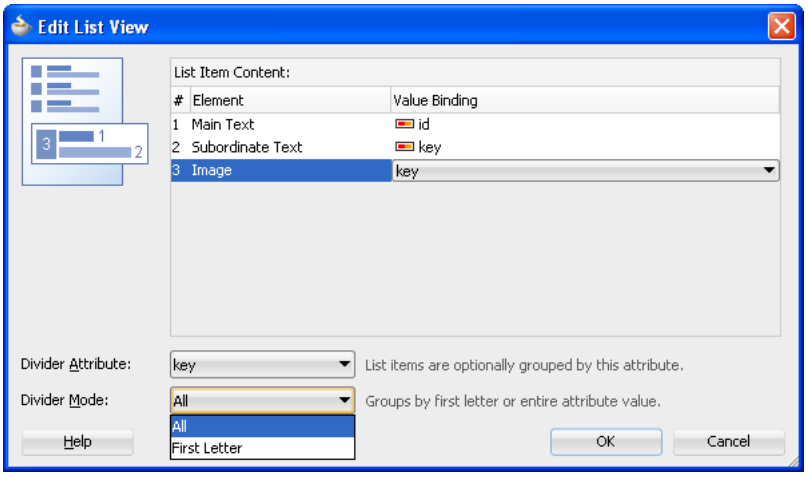


Figure 7–47 Specifying Divider Mode



For more information on List View dividers, see Section 8.2.7, "How to Use List View and List Item Components."

The following ADF Mobile AMX markup and data bindings are generated after you select the appropriate options in the Edit List View dialog:

```
<amx:listView var="row"
              value="#{bindings.contacts.collectionModel}"
              fetchSize="#{bindings.contacts.rangeSize}"
              dividerAttribute="key"
              dividerMode="all"
              id="listView1">
   <amx:listItem showLinkIcon="false" id="listItem1" >
      <amx:panelGroupLayout layout="horizontal"
```

```
                        id="panelGroupLayout1">
            <amx:image source="#{row.key}" id="image1"/>
            <amx:panelGroupLayout id="panelGroupLayout2">
               <amx:outputText value="#{row.id}"
                               styleClass="adfmf-listItem-maintext"
                               id="outputText2"/>
               <amx:outputText value="#{row.key}"
                               styleClass="adfmf-listItem-subtext"
                               id="outputText1"/>
            </amx:panelGroupLayout>
         </amx:panelGroupLayout>
      </amx:listItem>
   </amx:listView>
```

For more information about generated bindings, see Section 7.3.2.4.4, "What You May Need to Know About Generated Bindings."

The following are supported controls for ADF Mobile List View:

- ADF Mobile Output Text
- ADF Mobile Image

7.3.2.4.4 What You May Need to Know About Generated Bindings

Table 7–5 shows sample bindings that are added to an ADF Mobile AMX page when components are dropped.

Table 7–5 Sample Data Bindings

Component	Data Bindings
Button	`actionListener="#{bindings.FindContacts.execute}" text="FindContacts" disabled="#{!bindings.FindContacts.enabled}"`
Link	`actionListener="#{bindings.FindContacts.execute}" text="FindContacts" disabled="#{!bindings.FindContacts.enabled}"`
Input Date with Label	`<amx:inputDate id="inputDate1" value="#{bindings.timeStamp.inputValue}" label="#{bindings.timeStamp.hints.label}" required="#{bindings.timeStamp.hints.mandatory}"> </amx:inputDate>`
Input Date	`<amx:inputDate id="inputDate1" value="#{bindings.timeStamp.inputValue}" required="#{bindings.timeStamp.hints.mandatory}"> </amx:inputDate>`
Input Text with Label	`value="#{bindings.contactData.inputValue}" label="#{bindings.contactData.hints.label}" required="#{bindings.contactData.hints.mandatory}" maximumLength="#{bindings.contactData.hints.precision}"`
Input Text	`value="#{bindings.contactData.inputValue}" simple="true" required="#{bindings.contactData.hints.mandatory}" maximumLength="#{bindings.contactData.hints.precision}"`
Output Text	`value="#{bindings.contactData.inputValue}"`
Output Text with Label	`<amx:panelLabelAndMessage id="panelLabelAndMessage1" label="#{bindings.contactData.hints.label}"> <amx:outputText value="#{bindings.contactData.inputValue}"/> </amx:panelLabelAndMessage>`

Table 7–5 (Cont.) Sample Data Bindings

Component	Data Bindings
Select Boolean Checkbox	`value="#{bindings.contactData.inputValue}"` `label="#{bindings.contactData.label}"`
Select Boolean Switch	`value="#{bindings.contactData.inputValue}"` `label="#{bindings.contactData.label}"`

Select One Button	`<amx:selectOneButton id="selectOneButton1"`
	`value="#{bindings.contactData.inputValue}"`
	`label="#{bindings.contactData.label}"`
	`required="#{bindings.contactData.hints.mandatory}">`
	`<amx:selectItems value="#{bindings.contactData.items}"/>`
	`</amx:selectOneButton>`

Select One Choice	`<amx:selectOneChoice id="selectOneChoice1"`
	`value="#{bindings.contactData.inputValue}"`
	`label="#{bindings.contactData.label}"`
	`required="#{bindings.contactData.hints.mandatory}">`
	`<amx:selectItems value="#{bindings.contactData.items}"/>`
	`</amx:selectOneChoice>`

Select Many Checkbox	`<amx:selectManyCheckbox id="selectManyCheckbox1"`
	`value="#{bindings.AssetView.inputValue}"`
	`label="#{bindings.AssetView.label}">`
	`<amx:selectItems value="#{bindings.AssetView.items}"/>`
	`</amx:selectManyCheckbox>`

Select One Radio	`<amx:selectOneRadio id="selectOneRadio1"`
	`value="#{bindings.contactData.inputValue}"`
	`label="#{bindings.contactData.label}"`
	`required="#{bindings.contactData.hints.mandatory}">`
	`<amx:selectItems value="#{bindings.contactData.items}"/>`
	`</amx:selectOneRadio>`

Select Many Choice	`<amx:selectManyChoice id="selectManyChoice1"`
	`value="#{bindings.AssetView.inputValue}"`
	`label="#{bindings.AssetView.label}">`
	`<amx:selectItems value="#{bindings.AssetView.items}"/>`
	`</amx:selectManyChoice>`

7.3.2.4.5 What You May Need to Know About Generated Drag and Drop Artifacts The first drag and drop event generates the following directories and files:

- \Application\ViewController\adfmsrc\
  - \mobile
    - \pageDefs
      - **view1PageDef.xml**
      - **view2PageDef.xml**
      - **...**
    - **DataBindings.cpx**

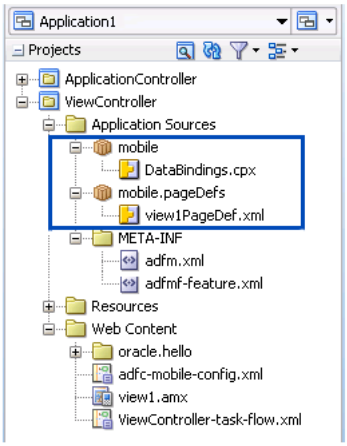


Figure 7–48 shows a sample `DataBindings.cpx` file generated upon drag and drop.

Figure 7–48 DataBindings.cpx File

```
<?xml version="1.0" encoding="UTF-8" ?>
<Application xmlns="http://xmlns.oracle.com/adfm/application"
             version="12.1.1.60.73" id="DataBindings"
             SeparateXMLFiles="false" Package="mobile" ClientType="Generic">
  <pageMap>
    <page path="/view1.amx" usageId="mobile_view1PageDef"/>
  </pageMap>
  <pageDefinitionUsages>
    <page id="mobile_view1PageDef" path="mobile.pageDefs.view1PageDef"/>
  </pageDefinitionUsages>
  <dataControlUsages>
    <dc id="DeviceDataControl" path="model.DeviceDataControl"/>
  </dataControlUsages>
</Application>
```

For more information, see the "Working with the DataBindings.cpx File" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Figure 7–49 shows a sample `PageDef` file generated upon drag and drop.

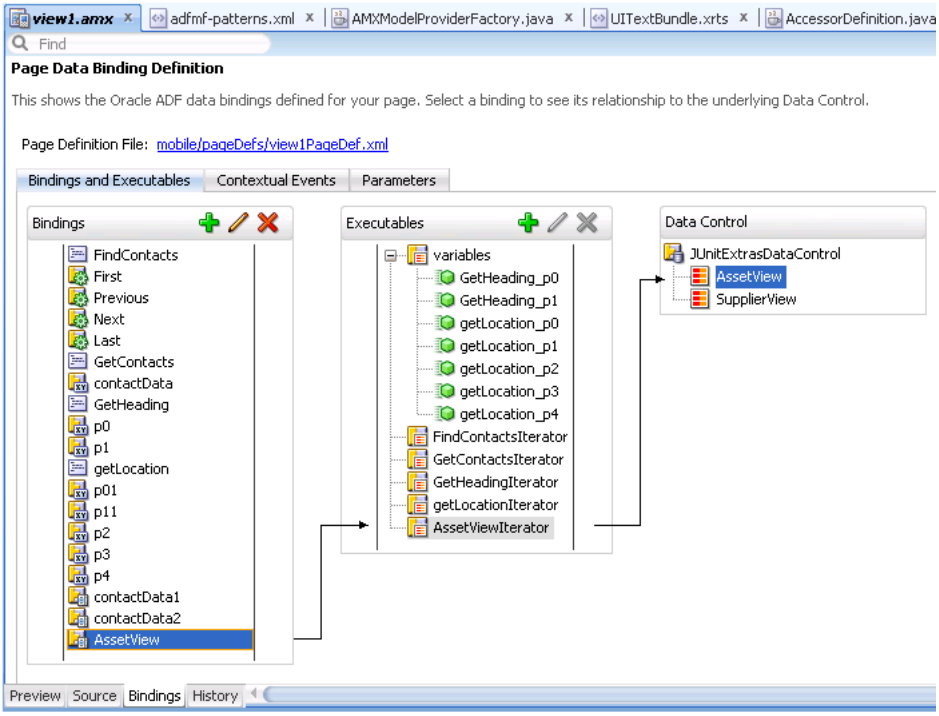
Figure 7–49 PageDef File

```
<?xml version="1.0" encoding="UTF-8" ?>
<pageDefinition xmlns="http://xmlns.oracle.com/adfm/uimodel" version="12.1.1.60.73" id="view1PageDef"
                Package="mobile.pageDefs">
  <parameters/>
  <executables>
    <variableIterator id="variables"/>
    <methodIterator Binds="GetContacts.result" DataControl="DeviceDataControl" RangeSize="25"
                    BeanClass="oracle.adfmf.model.datacontrols.device.Contact" id="GetContactsIterator"/>
  </executables>
  <bindings>
    <methodAction id="GetContacts" RequiresUpdateModel="true" Action="invokeMethod" MethodName="GetContacts"
                  IsViewObjectMethod="false" DataControl="DeviceDataControl"
                  InstanceName="data.DeviceDataControl.dataProvider"
                  ReturnName="data.DeviceDataControl.methodResults.
                              GetContacts_DeviceDataControl_dataProvider_GetContacts_result"/>
    <attributeValues IterBinding="GetContactsIterator" id="contactData">
      <AttrNames>
        <Item Value="contactData"/>
      </AttrNames>
    </attributeValues>
  </bindings>
</pageDefinition>
```

For more information, see the "Working with Page Definition Files" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

7.3.2.4.6 Using the ADF Mobile AMX Editor Bindings Tab JDeveloper's Bindings tab (see Figure 7–50) is available in the ADF Mobile AMX Editor. It displays the data bindings defined for a specific ADF Mobile AMX page. If you select a binding, its relationship to the underlying Data Control are shown and the link to the `PageDef` file is provided.

Figure 7–50 Bindings Tab



7.3.2.4.7 What You May Need to Know About Removal of Unused Bindings When you delete or cut an ADF Mobile AMX component from the Structure pane, unused bindings are automatically removed from your page.

> **Note:** Deleting a component from the Source editor does not trigger the removal of bindings.

Figure 7–51 demonstrates the deletion of a List View component that references bindings. Upon deletion, the related binding entry is automatically removed from the corresponding `PageDef.xml` file.

Figure 7–51 Deleting Bound Components from Page

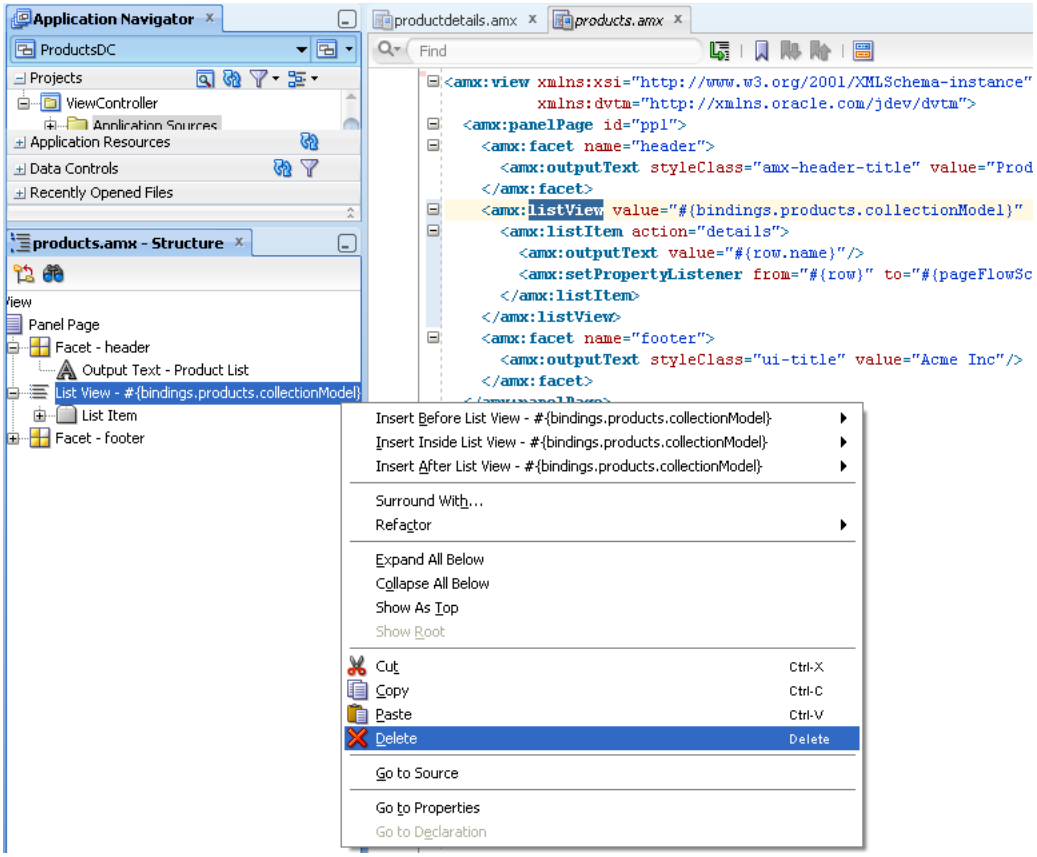
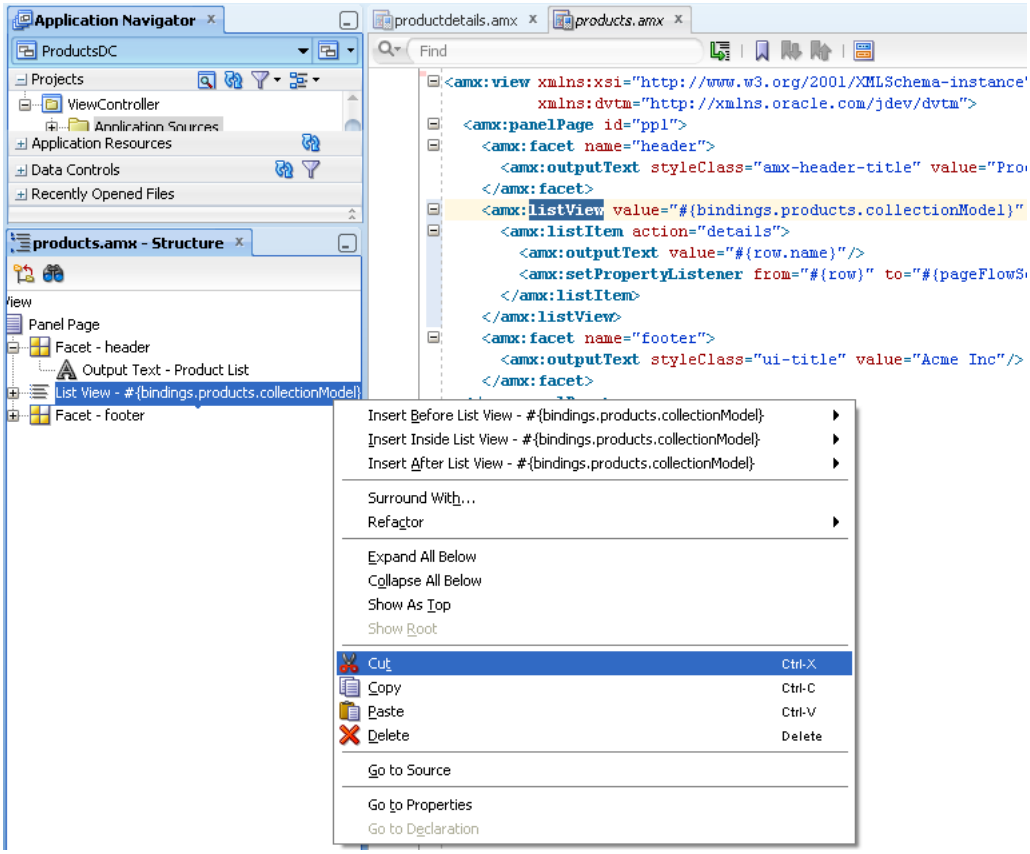


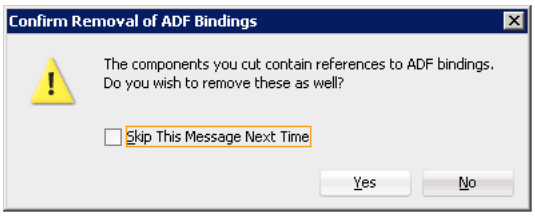
Figure 7–52 demonstrates the removal of the List View component by cutting it from the page.

Figure 7–52 Cutting Bound Components from Page



After clicking **Cut**, you are presented with the Confirm Removal of Bindings dialog that prompts you to choose whether or not to delete the corresponding bindings, as shown in Figure 7–53.

Figure 7–53 Confirm Removal of Bindings Dialog



7.3.2.5 What You May Need to Know About Element Identifiers and Their Audit

ADF Mobile generates a unique element identifier (`id`) and automatically inserts it into the ADF Mobile AMX page when an element is added by dropping a component from the Component palette, or by dragging and dropping a data control. This results in a valid identifier in the ADF Mobile AMX page that differentiates each component from others, possibly similar components within the same page.

ADF Mobile provides an identifier audit utility that does the following:

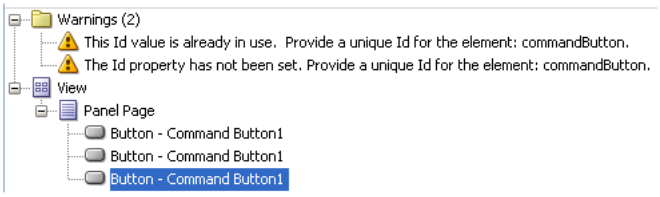
- Checks the presence and uniqueness of identifiers in an ADF Mobile AMX page.
- If the identifier is not present or not unique:
  - for a required `id` attribute of an element, an error is reported;

  - for an optional `id` attribute of an element, a warning is reported.
- Provides an automatic fix to generate a unique `id` for the element when a problem with the identifier is reported.

Figure 7–54 and Figure 7–55 show the identifier error reporting in the Source editor and Structure pane respectively.

Figure 7–54 Element Identifier Audit in Source Editor

Figure 7–55 Element Identifier Audit in Structure Pane



In addition to the `id`, the audit utility checks the `popupId` and `alignId` attributes of the Show Popup Behavior operation (see Section 8.2.8, "How to Use a Popup Component").

Figure 7–56 and Figure 7–57 show the Show Popup Behavior's Popup Id and Align Id attributes error reporting in the Source Editor respectively.

Figure 7–56 Popup Id Attribute Audit in Source Editor

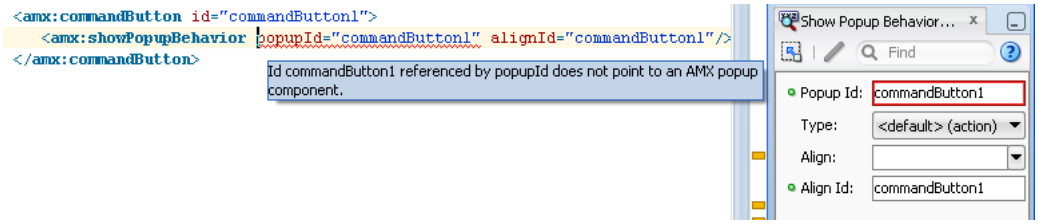
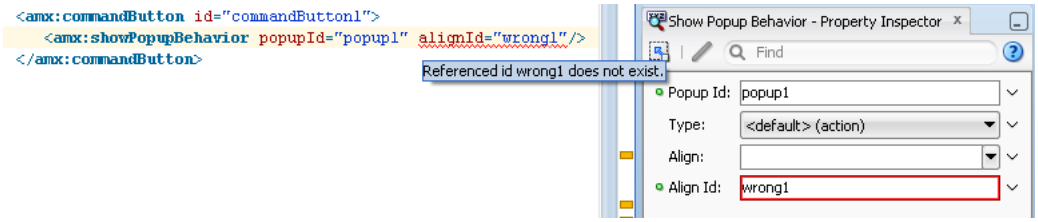


Figure 7–57 Align Id Attribute Audit in Source Editor



For more information, see the "Auditing and Profiling Applications" and "Understanding Auditing" in *Oracle Fusion Middleware User's Guide for Oracle JDeveloper*.

8

Creating ADF Mobile AMX User Interface

This chapter describes how to create the user interface for ADF Mobile AMX pages.

This chapter includes the following sections:

8.1 Introduction to Creating UI for ADF Mobile AMX Pages

ADF Mobile provides a set of layout and field components that enable you to create ADF Mobile AMX pages that behave appropriately for both the iOS and Android user experience. While ADF Mobile AMX maintains the same development experience as ADF Faces by allowing you to drag these components into an editor from the Component palette or from the Data Control palette, these components are not identical to their ADF Faces counterparts: ADF Mobile AMX components do not support every property and behavior of ADF Faces components. In essence, ADF Mobile AMX UI components render HTML equivalents of the native components on the iOS and Android platforms, with their design-time behavior being very similar to the ADF Faces components. In addition, the UI components are integrated with ADF Mobile's controller and model for declarative navigation and data binding.

> **Note:** When developing interfaces for mobile devices, always be aware of the fact that screen space is very limited. In addition, touchscreen support is not available on some mobile devices.

For more information, see the following:

- Chapter 7, "Creating ADF Mobile AMX Pages"
- Chapter 9, "Using Bindings and Creating Data Controls"

8.2 Designing the Page Layout

ADF Mobile AMX provides layout components (listed in Table 8–1) that let you arrange UI components in a page. Usually, you begin building pages with these components, and then add other components that provide other functionality either inside these containers, or as child components to the layout components. Some of these components provide geometry management functionality, such as the capability to stretch when placed inside a component that stretches.

Table 8–1 ADF Mobile AMX Page Management, Layout, and Spacing Components

Component	Type	Description
View	Core Page Structure Component	Creates a `view` element in an ADF Mobile AMX file. Automatically inserted into the file when the file is created. For more information, see Section 8.2.1, "How to Use a View Component."
Panel Page	Core Page Structure Component	Creates a `panelPage` element in an ADF Mobile AMX file. Defines the central area in a page that scrolls vertically between the header and footer areas. For more information, see Section 8.2.2, "How to Use a Panel Page Component." For more information about ADF Mobile AMX files, see Section 7.3.1.2, "Creating ADF Mobile AMX Pages."
Facet	Core Page Structure Component	Creates a `facet` element in an ADF Mobile AMX file. Defines an arbitrarily named facet on the parent component. For more information, see Section 8.2.6, "How to Use a Facet Component."
Panel Group Layout	Page Layout Container	Creates a `panelGroupLayout` element in an ADF Mobile AMX file. Groups child components either vertically or horizontally. For more information, see Section 8.2.3, "How to Use a Panel Group Layout Component."
Panel Form Layout	Page Layout Container	Creates a `panelFormLayout` element in an ADF Mobile AMX file. Positions components, such as Input Text, so that their labels and fields line up horizontally or above each component. For more information, see Section 8.2.4, "How to Use a Panel Form Layout Component."
Panel Label And Message	Page Layout Container	Creates a `panelLabelAndMessage` element in an ADF Mobile AMX file. Lays out a label and its children. For more information, see Section 8.2.5, "How to Use a Panel Label And Message Component."
List View	Page Layout Container	Creates a `listView` element in an ADF Mobile AMX file. Represents an iteration component that provides the look and feel of a list of rows with selection capabilities. For more information, see Section 8.2.7, "How to Use List View and List Item Components."
List Item	Page Layout Component	Creates a `listItem` element in an ADF Mobile AMX file. For more information, see Section 8.2.7, "How to Use List View and List Item Components."
Popup	Secondary Window	Creates a `popup` element in an ADF Mobile AMX file. For more information, see Section 8.2.8, "How to Use a Popup Component."

Table 8–1 (Cont.) ADF Mobile AMX Page Management, Layout, and Spacing

Component	Type	Description
Panel Splitter	Interactive Page Layout Container	Creates a `panelSplitter` element in an ADF Mobile AMX file. For more information, see Section 8.2.9, "How to Use a Panel Splitter Component."
Panel Item	Interactive Page Layout Component	Creates a `panelItem` element in an ADF Mobile AMX file. For more information, see Section 8.2.9, "How to Use a Panel Splitter Component."
Spacer	Spacing Component	Creates an area of blank space represented by a `spacer` element in an ADF Mobile AMX file. For more information, see the "Separating Content Using Blank Space or Lines" section in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.
Table Layout	Page Layout Container	Creates a `tableLayout` element in an ADF Mobile AMX file. Represents a table consisting of rows. For more information, see Section 8.2.10, "How to Use a Table Layout Component."
Row Layout	Page Layout Container	Creates a `rowLayout` element in an ADF Mobile AMX file. Represents a row consisting of cells in a Table Layout component. For more information, see Section 8.2.10, "How to Use a Table Layout Component."
Cell Format	Page Layout Component	Creates a `cellFormat` element in an ADF Mobile AMX file. Represents a cell in a Row Layout component. For more information, see Section 8.2.10, "How to Use a Table Layout Component."

You add a layout component by dragging and dropping it onto an ADF Mobile AMX page from the Component palette (see Section 7.3.2.1, "Adding UI Components"). Then you use the Property Inspector to set the component's attributes (see Section 7.3.2.3, "Configuring UI Components"). For information on attributes of each particular component, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

Example 8–1 demonstrates several page layout elements defined in an ADF Mobile AMX file.

> **Note:** You declare the page layout elements under the `<amx>` namespace.

Example 8–1 Page Layout Components Definition

```
<amx:panelPage id="pp1">
   <amx:outputText id="outputText1"
                   value="Sub-Section Title 1"
                   styleClass="adfmf-text-sectiontitle"/>
   <amx:panelFormLayout id="panelFormLayout1" labelPosition="start">
      <amx:panelLabelAndMessage id="panelLabelAndMessage1" label="Name">
         <amx:commandLink id="commandLink1" text="Jane Doe" action="editname" />
      </amx:panelLabelAndMessage>
      <amx:panelLabelAndMessage id="panelLabelAndMessage2" label="Street Address">
         <amx:commandLink id="commandLink2"
```

```
                        text="123 Main Street"
                        action="editaddr" />
      </amx:panelLabelAndMessage>
      <amx:panelLabelAndMessage id="panelLabelAndMessage3" label="Phone">
         <amx:outputText id="outputText2" value="888-555-1212" />
      </amx:panelLabelAndMessage>
   </amx:panelFormLayout>
   <amx:outputText id="outputText3"
                   value="Sub-Section Title 2"
                   styleClass="adfmf-text-sectiontitle" />
   <amx:panelFormLayout id="panelFormLayout2" labelPosition="start">
      <amx:panelLabelAndMessage id="panelLabelAndMessage4" label="Type">
         <amx:commandLink id="commandLink3" text="Personal" action="edittype" />
      </amx:panelLabelAndMessage>
      <amx:panelLabelAndMessage label="Anniversary">
         <amx:outputText id="outputText4" value="November 22, 2005" />
      </amx:panelLabelAndMessage>
   </amx:panelFormLayout>
   <amx:panelFormLayout id="panelFormLayout3" labelPosition="start">
      <amx:panelLabelAndMessage id="panelLabelAndMessage5" label="Date Created">
         <amx:outputText id="outputText5" value="June 20, 2011" />
      </amx:panelLabelAndMessage>
   </amx:panelFormLayout>
</amx:panelPage>
```

Figure 8–1 Page Layout Components at Design Time

You use the standard Cascading Style Sheets (CSS) to manage visual presentation of your layout components. CSS are located in the `Web Content/css` directory of your ViewController project, with default CSS provided by the framework. For more information, see Section 8.6.1, "How to Use Component Attributes to Define Style."

An ADF Mobile sample application called LayoutDemo demonstrates how to use layout components in conjunction with such ADF Mobile AMX UI components as a Button, to achieve some of the common layouts that follow common patterns. In addition, this sample application shows how to work with styles to adjust the page layout to a specific pattern. The LayoutDemo application is located in the

PublicSamples.zip file within the *jdev_install*/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples directory on your development computer.

8.2.1 How to Use a View Component

A View (view element in an ADF Mobile AMX file) is a core page structure component that is automatically inserted into an ADF Mobile AMX file when the file is created. This component provides a hierarchical representation of the page and its structure and represents a single ADF Mobile AMX page.

For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.2.2 How to Use a Panel Page Component

A Panel Page (panelPage element in an ADF Mobile AMX file) is a component that allows you to define a scrollable area of the screen for laying out other components.

By default, when you create an ADF Mobile AMX page, JDeveloper automatically creates and inserts a Panel Page component into the page. When you add components to the page, they will be inserted inside the Panel Page component.

To prevent scrolling of certain areas (such as a header and footer of the page) and enable stretching when orientation changes, you can specify a Facet component for your Panel Page. The Panel Page's header Facet includes the title placed in the Navigation Bar of each page. For information about other types of Facet components that the Panel Page can contain, see Section 8.2.6, "How to Use a Facet Component."

Example 8–2 shows the panelPage element defined in an ADF Mobile AMX file. This Panel Page contains a header Facet.

Example 8–2 Panel Page Definition

```
<amx:panelPage id="pp1">
   <amx:facet name="header">
      <amx:outputText id="ot1" value="Welcome"/>
   </amx:facet>
</amx:panelPage>
```

For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.2.3 How to Use a Panel Group Layout Component

The Panel Group Layout component is a basic layout component that lays out its children horizontally or vertically. To create the Panel Group Layout component, use the Component palette.

To add the Panel Group Layout component:

1. In the **Component Palette**, drag and drop a **Panel Group Layout** to the ADF Mobile AMX page.
2. Insert the desired child components into the Panel Group Layout component.
3. To add spacing between adjacent child components, insert the Spacer (spacer) component.
4. Use the **Property Inspector** to set the component attributes. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

Example 8–3 shows the `panelGroupLayout` element defined in an ADF Mobile AMX file.

Example 8–3 Panel Group Layout Definition

```
<amx:panelGroupLayout styleClass="prod" id="pgl1">
   <amx:outputText styleClass="prod-label" value="Screen Size:" id="ot1"/>
</amx:panelGroupLayout>
```

8.2.4 How to Use a Panel Form Layout Component

The Panel Form Layout (`panelFormLayout`) component positions components so that their labels and fields align horizontally. In general, the main content of the Panel Form Layout component is comprised of input components (such as Input Text) and selection components (such as Choice). If a child component with a `label` attribute defined is placed inside the Panel Form Layout component, the child component's label and field are aligned and sized based on the Panel Form Layout definitions. Within the Panel Form Layout, the label area can either be displayed on the start side of the field area or on a separate line above the field area. Separate lines are used if the `labelPosition` attribute of the Panel Form Layout is set to `topStart`, `topCenter`, or `topEnd`. Otherwise the label area appears on the start side of the field area. Within the label area, the `labelPosition` attribute controls where the label text can be aligned:

- to the start side (`labelPosition="start"` or `labelPosition="topStart"`)
- to the center (`labelPosition="center"` or `labelPosition="topCenter"`)
- to the end side (`labelPosition="end"` or `labelPosition="topEnd"`)

Within the field area, the `fieldHalign` attribute controls where the field content can be aligned:

- to the start side (`fieldHalign="start"`)
- to the center (`fieldHalign="center"`)
- to the end side (`fieldHalign="end"`)

To add the Panel Form Layout component:

1. In the **Components** palette, drag and drop a **Panel Form Layout** component to the ADF Mobile AMX page.
2. In the **Property Inspector**, set the component's attributes. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

Example 8–4 shows the `panelFormLayout` element defined in an ADF Mobile AMX file.

Example 8–4 Panel Form Layout Definition

```
<amx:panelFormLayout styleClass="prod" id="pfl1">
   <amx:panelLabelAndMessage label="Type" id="plm1">
      <amx:commandLink text="Personal" action="edittype" id="cl1"/>
   </amx:panelLabelAndMessage>
</amx:panelFormLayout>
```

8.2.5 How to Use a Panel Label And Message Component

Use the Panel Label And Message (`panelLabelAndMessage`) component to place a component, which does not have a `label` attribute, inside a Panel Form Layout. These components usually include an Output Text, Button, or Link.

To add the Panel Label And Message component:

1. In the **Components** palette, drag and drop a **Panel Label And Message** component into a **Panel Group Layout** component.
2. In the **Property** editor, set the component's attributes. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

Example 8–4 shows the `panelLabelAndMessage` element defined in an ADF Mobile AMX file. The `label` attribute is used for the child component.

Example 8–5 Panel Label and Message Definition

```
<amx:panelLabelAndMessage label="Phone" id="plm1">
   <amx:outputText value="888-555-1212" id="ot1"/>
</amx:panelLabelAndMessage>
```

8.2.6 How to Use a Facet Component

You use the Facet (`facet`) component to define an arbitrarily named facet, such as a header or footer, on the parent layout component. The position and rendering of the Facet are determined by the parent component.

The ADF Mobile AMX page header is typically represented by the Panel Page component (see Section 8.2.2, "How to Use a Panel Page Component") in combination with the Header, Primary, and Secondary facets:

- Header facet: contains the page title.
- Primary Action facet: represents an area that appears in the left corner of the header bar and typically hosts Button or Link components, but can contain any component type.
- Secondary Action facet: represents an area that appears in the right corner of the header bar and typically hosts Button or Link components, but can contain any component type.

The ADF Mobile AMX page footer is represented by the Panel Page component (see Section 8.2.2, "How to Use a Panel Page Component") in combination with the footer facet:

- Footer facet: represents an area that appears below the content area and typically hosts Button or Link components, but can contain any component type.

Example 8–6 shows the `facet` element declared inside the Panel Page container. The type of the facet is always defined by its `name` attribute (see Table 8–2).

Example 8–6 Facet Definition

```
<amx:panelPage id="pp1">
   <amx:facet name="footer">
      <amx:commandButton id="cb2" icon="folder.png"
                         text="Move (#{myBean.mailcount})"
                         action"move"/>
   </amx:facet>
</amx:panelPage>
```

Table 8–2 lists Facet types that you can use with specific parent components.

Table 8–2 Facet Types and Parent Components

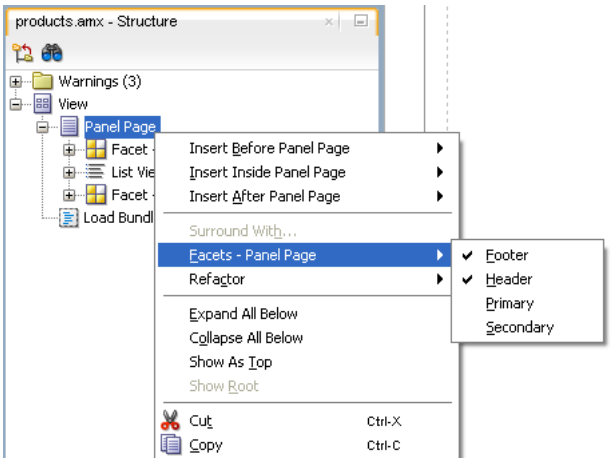
Parent Component	Facet Type (name)
Panel Page (`panelPage`)	`header`, `footer`, `primary`, `secondary`
List View (`listView`)	`header`, `footer`
Data Visualization Components. For more information, see Section 8.5, "Providing Data Visualization."	`dataStamp`, `seriesStamp`

To add the Facet component:

You can use the context menu displayed on the Structure pane or Source view to add a Facet component as a child of another component. The context menu displays only facets that are valid for your selected parent component. To add a Facet, first select and then right-click the parent component in the Structure pane or Source view, and then select one of the following:

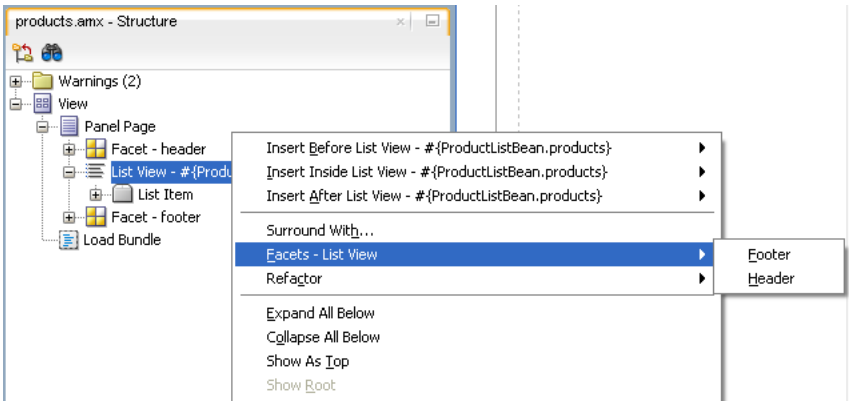
- If the parent component is a Panel Page, select **Facets > Panel Page**, and then choose the type of Facet from the list, as Figure 8–2 shows.

Figure 8–2 Using Context Menu to Add Facet to Panel Page



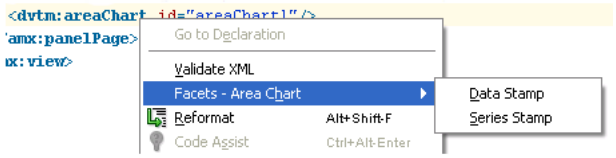
- If the parent component is a List View, select **Facets > List View**, and then choose the type of Facet from the list, as Figure 8–3 shows.

Figure 8–3 Using Context Menu to Add Facet to List View



- If the parent component is one of the data visualization components, select **Facets > *<DVTM Component Name>***, and then choose the type of Facet from the list, as Figure 8–4 shows.

Figure 8–4 Using Context Menu to Add Facet to Data Visualization Component



For more information about data visualization components and their attributes, see Section 8.5, "Providing Data Visualization."

Alternatively:

1. In the **Components Palette**, drag and drop a **Facet** component into another component listed in Table 8–2.
2. In the **Property Inspector**, set the component's attributes. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.2.7 How to Use List View and List Item Components

Use the List View (`listView`) component to display data as a list of choices where the end user can select one or more options.

The List Item (`listItem`) component represents a single row in the List View. Typically, you place a List Item component inside the List View to lay out and style a list of data items. At run time, List Item components respond to swipe gestures (see Section 8.4, "Enabling Gestures").

The List View allows you to define one of the following:

- A row that is replicated based on the number of items in the list (collection).
- A static row that is produced by adding a child List Item component without specifying the List View's `var` and `value` attributes. You can add as many of these static items as necessary, which is useful when you know the contents of the list at design time. In this case, the list behaves like a set of menu items.

You can create the following types of List View components:

- Basic List

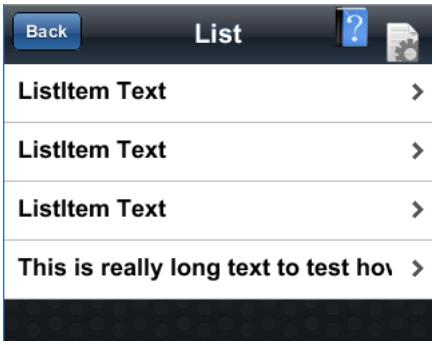
  Example 8–7 shows the `listView` element defined in an ADF Mobile AMX file. This definition corresponds to the basic component.

Example 8–7 Basic List View Definition

```
<amx:listView id="listView1">
   <amx:listItem id="listItem1">
      <amx:outputText id="outputText1" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem2">
      <amx:outputText id="outputText3" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem3">
      <amx:outputText id="outputText5" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem4">
      <amx:outputText id="outputText7"
                      value="This is really long text to test how it is handled"/>
   </amx:listItem>
</amx:listView>
```

Figure 8–5 demonstrates a basic List View component at design time.

Figure 8–5 Basic List View at Design Time



Example 8–8 shows another definition of the `listView` element in an ADF Mobile AMX file. This definition also corresponds to the basic component; however, the value of this List View is provided by a collection.

Example 8–8 Basic List View Definition

```
<amx:listView id="list1" value="#{myBean.listCollection}" var="row">
   <amx:listItem actionListener="#{myBean.selectRow}"
                 showLinkIcon="false"
                 id="listItem1">
      <amx:outputText value="#{row.name}" id="outputText1"/>
   </amx:listItem>
</amx:listView>
```

- List with icons

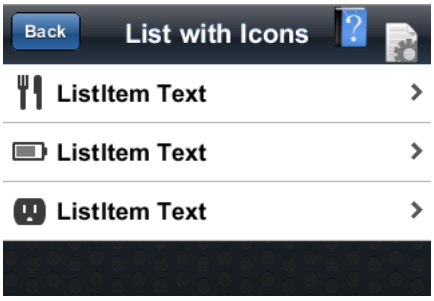
  Example 8–9 shows the `listView` element defined in an ADF Mobile AMX file. This definition corresponds to the component with icons.

Example 8–9 List View with Icons Definition

```
<amx:listView id="list1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItem1">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
            <amx:cellFormat id="cf11" width="40px" halign="center">
               <amx:image id="image1" source="#{row.image}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf12" width="100%" height="43px">
               <amx:outputText id="outputText1" value="#{row.desc}"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 8–6 demonstrates a List View component with icons and text at design time.

Figure 8–6 List View with Icons at Design Time



- List with search
- List with dividers. This type of list allows you to group data and show order. Attributes of the List View component define characteristics of each divider. For information about attributes, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

  ADF Mobile AMX provides a list divider that can do the following:

  – Collapse its contents independently.

  – Show a count of items in each divider.

  – Collapse at the same time.

  Example 8–10 shows the `listView` element defined in an ADF Mobile AMX file. This definition corresponds to the component with collapsible dividers and item counts.

Example 8–10 List View with Dividers Definition

```
<amx:listView id="list1"
          value="#{bindings.data.collectionModel}"
          var="row"
          collapsibleDividers="true"
          collapsedDividers="#{pageFlowScope.mylistDisclosedDividers}"
          dividerMode="all"
          dividerAttribute="type"
          showDividerCount="true">
```

```
   <amx:listItem>
      <amx:outputText id="ot1" value="#{row.name}">
   </amx:listItem>
</amx:listView>
```

- Inset List

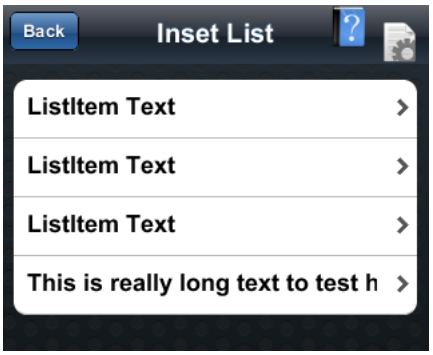
  Example 8–11 shows the `listView` element defined in an ADF Mobile AMX file. This definition corresponds to the inset component.

Example 8–11 Inset List View Definition

```
<amx:listView id="listView1" styleClass="adfmf-listView-insetList">
   <amx:listItem id="listItem1">
      <amx:outputText id="outputText1" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem2">
      <amx:outputText id="outputText3" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem3">
      <amx:outputText id="outputText5" value="ListItem Text"/>
   </amx:listItem>
   <amx:listItem id="listItem4">
      <amx:outputText id="outputText7"
                      value="This is really long text to test how it is handled"/>
   </amx:listItem>
</amx:listView>
```

Figure 8–7 demonstrates an inset List View component at design time.

Figure 8–7 Inset List View at Design Time



Example 8–12 shows another definition of the `listView` element in an ADF Mobile AMX file. This definition also corresponds to the inset component, however, the value of this List View is provided by a collection.

Example 8–12 Inset List Definition

```
<amx:listView id="list1"
              value="#{CarBean.carCollection}"
              var="row"
              styleClass="adfmf-listView-insetList">
   <amx:listItem id="li1" action="go">
      <amx:outputText id="ot1" value="#{row.name}"/>
   </amx:listItem>
</amx:listView>
```

There is a particular order in which ADF Mobile AMX processes the List Item component's child operations and attributes. For more information, see Section 8.3.5.7, "What You May Need to Know About the Order of Processing Operations and Attributes."

Unlike other ADF Mobile AMX components, when you drag and drop a List View onto an ADF Mobile AMX page, a dialog called **ListView Gallery** appears (see Figure 8–8). This dialog allows you to choose a specific layout for the List View.

Figure 8–8 ListView Gallery Dialog

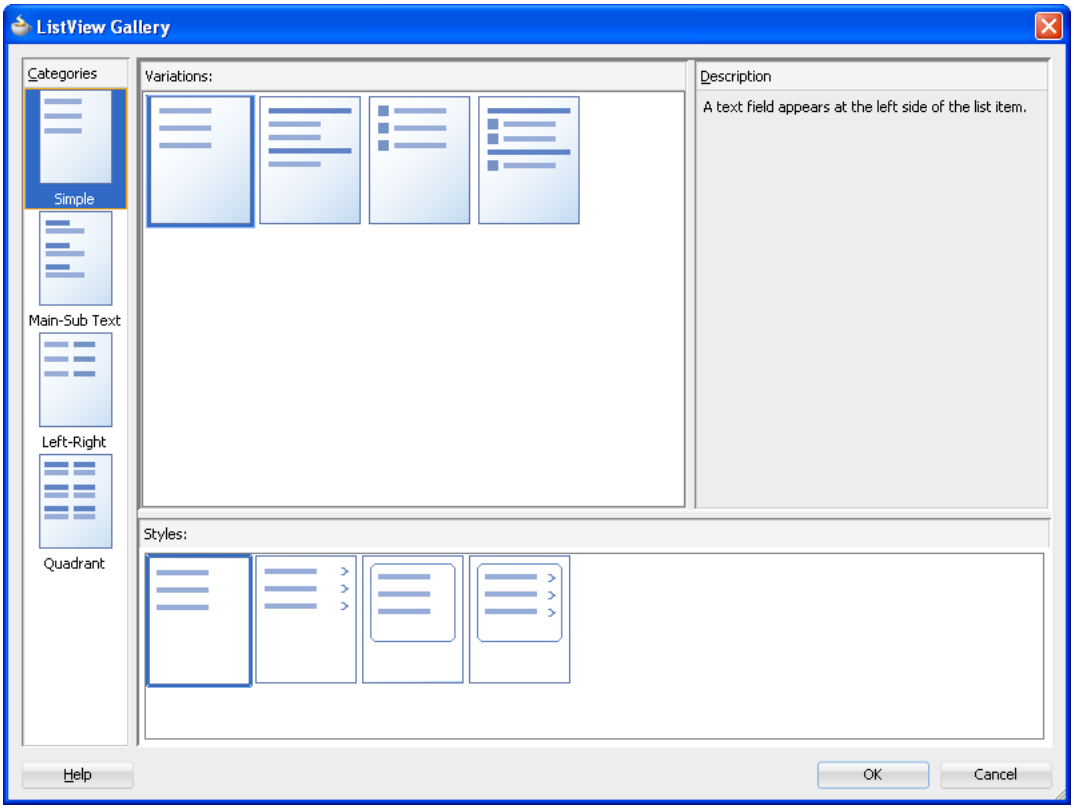


Table 8–3 lists the supported **List Formats** that are displayed in the ListView Gallery.

Table 8–3 List Formats

Format	Element Row Values
Simple	▪ Text
Main-Sub Text	▪ Main Text ▪ Subordinate Text
Start-End	▪ Start Text ▪ End Text
Quadrant	▪ Upper Start Text ▪ Upper End Text ▪ Lower Start Text ▪ Lower End Text

The **Variations** presented in the ListView Gallery (see Figure 8–8) for a selected list format consist of options to add either dividers, a leading image, or both:

- Selecting a variation with a leading image adds an Image row to the List Item Content table (see Figure 8–9).
- Selecting a variation with a divider defaults the Divider Attribute field to the first attribute in its list rather than the default No Divider value, and populates the Divider Mode field with its default value of All.

The **Styles** options presented in the ListView Gallery (see Figure 8–8) allow you to suppress chevrons, use an inset style list, or both:

- The selections do not modify any state in the Edit List View dialog (see Figure 8–9). They only affect the generated ADF Mobile AMX markup.
- Selecting a style with the inset list sets the `adfmf-listView-insetList` style class on the `listView` element in the generated ADF Mobile AMX markup.
- Selecting a style without chevrons sets the `showLinkIcon` attribute on the `listItem` element to `false` in the generated ADF Mobile AMX markup.

The following is an example of the Simple format with the inset list and chevrons:

```
<amx:listView var="row"
              value="#{bindings.employees.collectionModel}"
              fetchSize="#{bindings.employees.rangeSize}"
              styleClass="adfmf-listView-insetList"
              id="listView2">
   <amx:listItem id="li2">
      <amx:outputText value="#{row.employeename}" id="ot3"/>
   </amx:listItem>
</amx:listView>
```

The ListView Gallery's **Description** pane is updated based on the currently selected Variation. The format includes a brief description of the main style, followed by the details of the selected variation. Four main styles with four variations on each provide sixteen unique descriptions detailed in Table 8–4.

Table 8–4 List View Variations and Styles

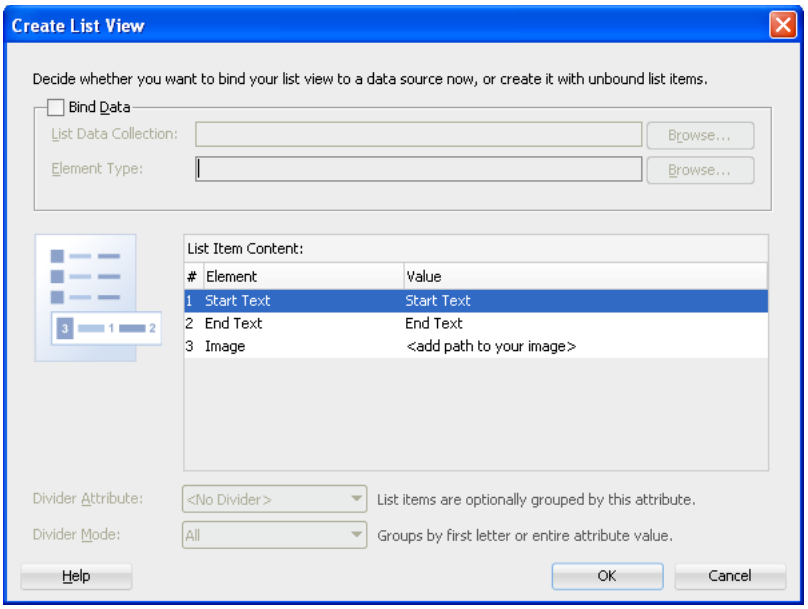
List Format	Variation	Description
Simple	Basic	A text field appears at the start side of the list item.
Simple	Dividers	A text field appears at the start side of the list item, with items grouped by dividers.
Simple	Images	A text field appears at the start side of the list item, following a leading image.
Simple	Dividers and Images	A text field appears at the start side of the list item, following a leading image. The list items are grouped by dividers.
Main-Sub Text	Basic	A prominent main text field appears at the start side of the list item with subordinate text below.
Main-Sub Text	Dividers	A prominent main text field appears at the start side of the list item with subordinate text below. The list items are grouped by dividers.
Main-Sub Text	Images	A prominent main text field appears at the start side of the list item with subordinate text below, following a leading image.

Table 8–4 (Cont.) List View Variations and Styles

List Format	Variation	Description
Main-Sub Text	Dividers and Images	A prominent main text field appears at the start side of the list item with subordinate text below, following a leading image. The list items are grouped by dividers.
Start-End	Basic	Text fields appear on each side of the list item.
Start-End	Dividers	Text fields appear on each side of the list item, with the items grouped by dividers.
Start-End	Images	Text fields appear on each side of the list item, following a leading image.
Start-End	Dividers and Images	Text fields appear on each side of the list item, following a leading image. The list items are grouped by dividers.
Quadrant	Basic	Text fields appear in the four corners of the list item.
Quadrant	Dividers	Text fields appear in the four corners of the list item, with items grouped by dividers.
Quadrant	Images	Text fields appear in the four corners of the list item, following a leading image.
Quadrant	Dividers and Images	Text fields appear in the four corners of the list item, following a leading image. The list items are grouped by dividers.

After you make your selection from the ListView Gallery and click **OK**, the **Edit List View** wizard is invoked that lets you create either an unbound List View component that displays static text in the List Item child components (see Figure 8–9), or choose a data source for the dynamic binding (see Figure 8–10).

Figure 8–9 Creating Unbound List View



When completing the dialog that Figure 8–9 shows, consider the following:

- The List Data Collection and Element Type fields are disabled when the Bind Data checkbox is in the deselected state.

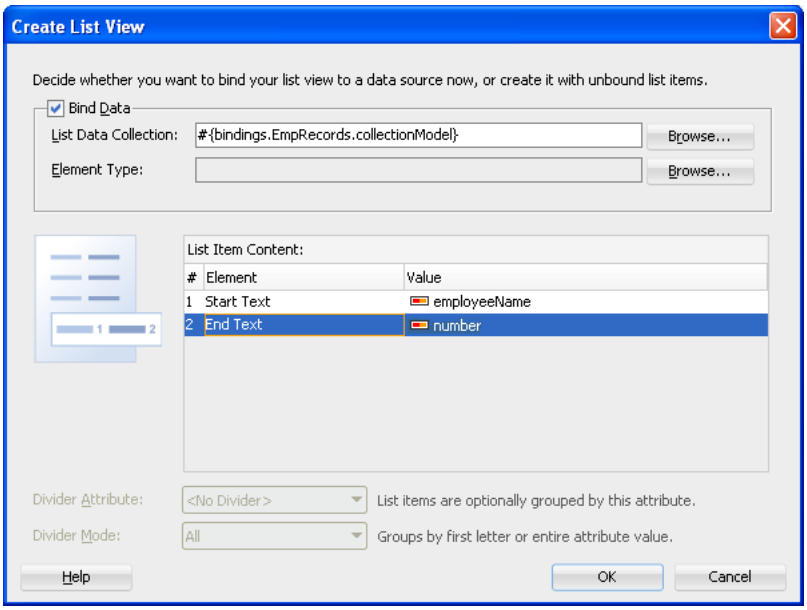
- The image on the left reflects the main content elements from the selected List View format
- The editable cells of the Value column are populated with static text strings (see Table 8–5).
- If the List Item Content cell contains an Image, the Value cell is defaulted to the `<add path to your image>` string. If this is the case, you have to replace it with the path to the image.
- Since you cannot set the divider attribute when the List View contains List Item child components, rather than being data bound, both the Divider Attribute and the Divider Mode fields are disabled.

Table 8–5 Static Text Strings for List View

List Format	Element Row Values	Values for the Output Text
Simple	▪ Text	▪ 'ListItem Text'
Main-Sub Text	▪ Main Text ▪ Subordinate Text	▪ 'Main Text' ▪ 'This is the subordinate text.'
Start-End	▪ Start Text ▪ End Text	▪ 'Start Text' ▪ 'End Text'
Quadrant	▪ Upper Start Text ▪ Upper End Text ▪ Lower Start Text ▪ Lower End Text	▪ 'Start Text' ▪ 'End Text' ▪ 'Lower Start Text' ▪ 'Lower End Text'

Figure 8–10 shows the Edit List View dialog when you choose to bind the List View component to data.

Figure 8–10 Creating Bound List View



When completing the dialog that Figure 8–10 shows, consider the following:

- When you select the Bind Data Now checkbox, the List Data Collection and Element Type fields become enabled.
- To select a data source, click Browse. This opens the Select List View Data Collection dialog that enables you to either choose a data control definition (see Figure 8–11) or to use the EL Builder to select an appropriate EL expression (see Figure 8–12).

Figure 8–11 Selecting Data Control Definition

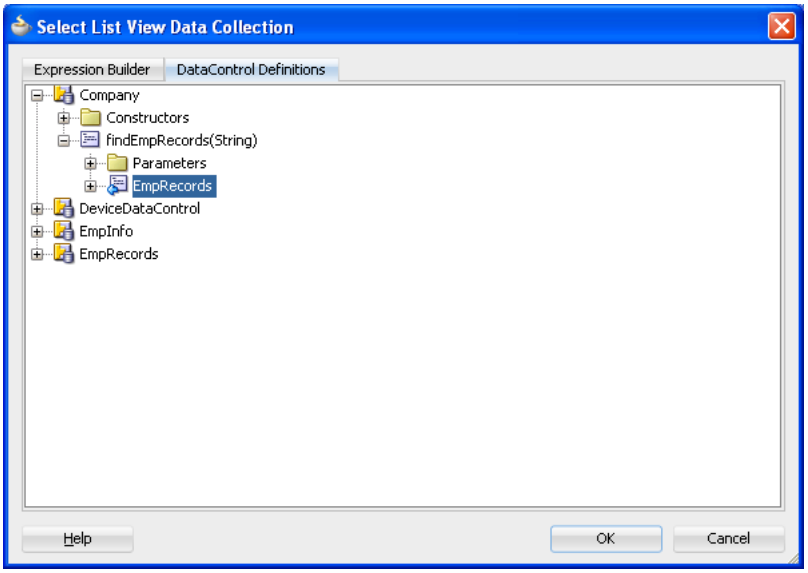
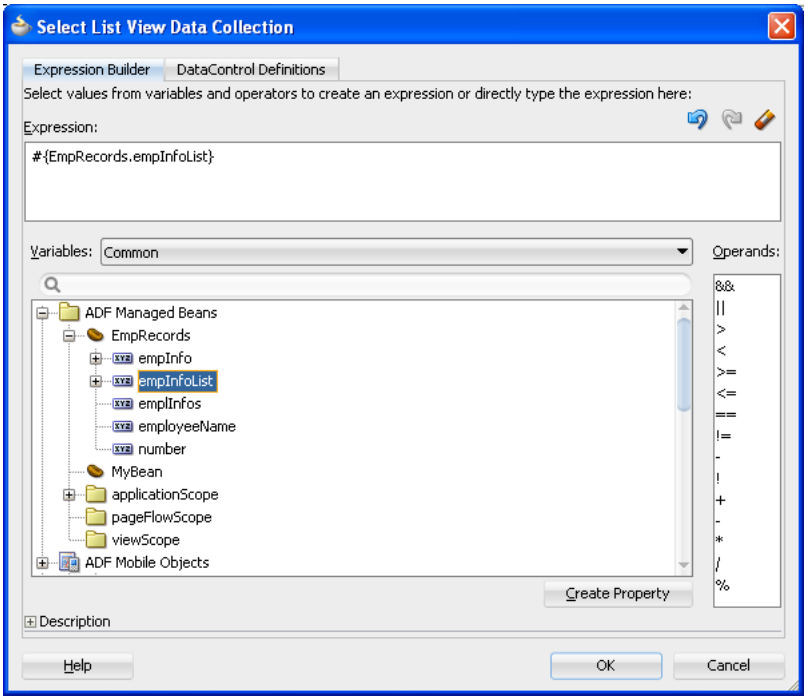


Figure 8–12 Selecting EL Expression



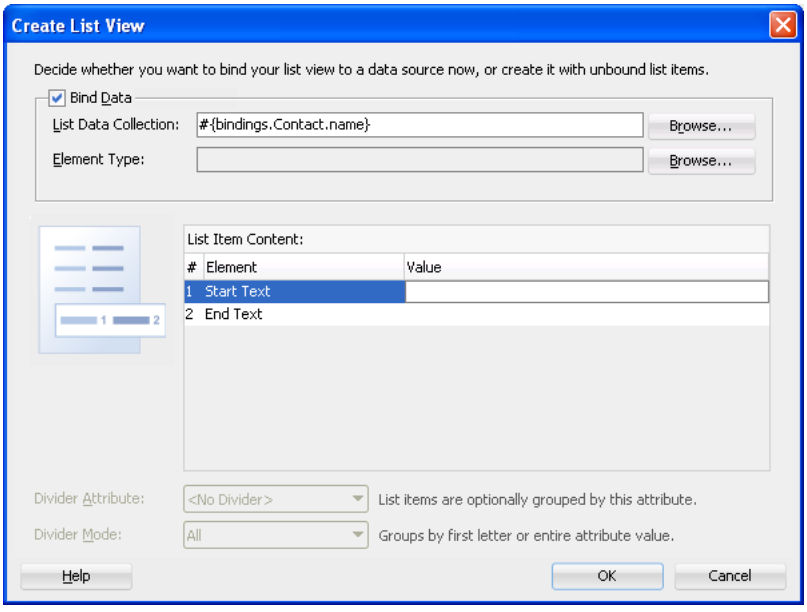
- You may declare the type of the data collection using the Element Type field (see Figure 8–10).

- After you have selected a valid data collection, the Value column in the List Item Content table changes to Value Bindings whose editable cells are populated with lists of attributes from the data collection. For a description of a special case setting, refer to Figure 8–13.
- The image on the left reflects the main content elements from the selected List View format and provides a mapping from the schematic representation to the named elements in the underlying table.
- The List Item is generated as either an Output Text or Image component, depending on whichever is appropriate for the particular element.
- Since the number of elements (rows) is predetermined by the selected List View format, rows cannot be added or removed.
- The order of elements cannot be modified.
- The default value of the Divider Attribute field is No Divider, in which case the Divider Mode field is disabled. If you select value other than the default, then you need to specify Divider Mode parameters. In addition, if you chose a Variation in the ListView Gallery that includes dividers, the default value will be set to the first attribute in the list.

The following are special cases to consider when creating a bound List View:

- If a Java bean method returns a list without generics, you should use the Element Type field to create the List Item content, as Figure 8–10 shows.
- If the list data collection value provided is not collection-based, a Value column replaces the Value Bindings column with blank values, as Figure 8–13 shows.

Figure 8–13 Providing Non-Collection-Based Values



For more information, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- Section 8.2.7.3, "Configuring Layout Using Styles"
- Section 7.3.2.4.3, "Dragging and Dropping Collections"

- LayoutDemo, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer. This sample demonstrates how to use various types of the List View component and how to apply styles to adjust the page layout to a specific pattern.
- Appendix C.5, "Limitations to the Application Usage"

8.2.7.1 Configuring List View Paging

You can configure the List View component to display data in a list that is arbitrarily long. This is done by appending data to the bottom of the list.

The `fetchSize` attribute determines how many rows the List View component should initially load. If there are more rows available than defined by the `fetchSize`, a clickable area is displayed after the last fetched row. Clicking within this area loads another batch of rows that equals the `fetchSize`. Once there are no more rows to display, the clickable area disappears.

The `fetchSize` attribute does not represent the number of loaded rows. Instead, it represents the value by which the number of rows is incremented. When the List View component is created, the `fetchSize` attribute is by default set to use an EL expression that points to the `rangeSize` of the `PageDef` iterator. For information on the `PageDef` file, see Section 7.3.2.4.5, "What You May Need to Know About Generated Drag and Drop Artifacts" and Figure 7–49, "PageDef File". Setting the `fetchSize` to the same value as the `rangeSize` improves the application performance.

Example 8–13 shows the `listView` element that was created from a collection called `testResults` of a data control (see Section 7.3.2.4, "Adding Data Controls to the View").

Example 8–13 Setting fetchSize Attribute

```
<amx:listView var="row"
              value="#{bindings.testResults.collectionModel}"
              fetchSize="#{bindings.testResults.rangeSize}">
```

In the preceding example, the `fetchSize` attribute points to the `rangeSize` on `bindings.testResults`. Example 8–14 shows a line from the `PageDef` file for this ADF Mobile AMX page. This `PageDef` file contains an `accessorIterator` element called `testResultsIterator` to which the `testResults` is bound.

Example 8–14 accessorIterator in PageDef File

```
<accessorIterator MasterBinding="Class1Iterator"
                  Binds="testResults"
                  RangeSize="25"
                  DataControl="Class1"
                  BeanClass="mobile.Test"
                  id="testResultsIterator"/>
```

For more information, see the "Working with Page Definition Files" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

If the `fetchSize` attribute is set to -1, all records are retrieved.

8.2.7.2 Rearranging List View Items

Items in a List View can be rearranged. This functionality is similar on iOS and Android platforms: both show a Rearrange icon aligned along the right margin for each list item. The Rearrange operation is initiated when the end user touches and drags a list item using the Rearrange affordance as a handle. The operation is completed when the end user lifts their finger from the display screen.

> **Note:** If the end user touches and drags anywhere else on the list item, the list scrolls up or down.

The Rearrange Drag operation "undocks" the list item and allows the end user to move the list item up or down in the list.

For its items to be rearrangeable, the List View must be in an edit mode and be able to listen to moves.

Example 8–15 shows the `listView` element defined in an ADF Mobile AMX file. This definition presents a list with an Edit and Stop Editing buttons at the top that allow switching between editable and noneditable list mode.

Example 8–15 Rearrangeable List View Definition

```
<amx:panelPage id="pp1">
   <amx:commandButton id="edit" text="Edit"
                      rendered="#{!bindings.inEditMode.inputValue}">
      <amx:setPropertyListener from="true"
                               to="#{bindings.inEditMode.inputValue}"
                               type="action"/>
   </amx:commandButton>
   <amx:commandButton id="stop"
                      text="Stop Editing"
                      rendered="#{bindings.inEditMode.inputValue}">
      <amx:setPropertyListener from="false"
                               to="#{bindings.inEditMode.inputValue}"
                               type="action"/>
   </amx:commandButton>
   <amx:listView id="lv1"
                 value="#{bindings.data.collectionModel}"
                 var="row"
                 editMode="#{bindings.inEditMode.inputValue}"
                 moveListener="#{MyBean.Reorder}">
      <amx:listItem>
         <amx:outputText id= "ot1" value="#{row.description}">
      </amx:listItem>
   </amx:listView>
</amx:panelPage>
```

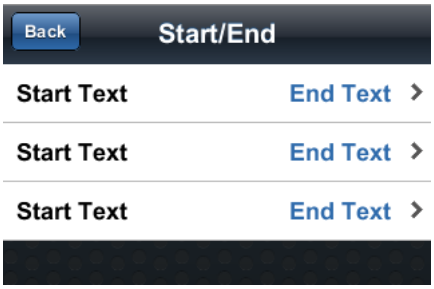
For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.2.7.3 Configuring Layout Using Styles

To adjust the ADF Mobile AMX page layout to a specific pattern, you can combine the use of the various types of List View components and styles that are defined through the `styleClass` attribute (see Section 8.6, "Styling UI Components") that uses a set of predefined styles.

Figure 8–14 shows a List View component with differently styled text added to the start (left side) and end (right side) of each row. Besides the text, rows are equipped with a right-pointing arrow representing a link that expands each list item.

Figure 8–14 List View with Start and End Text at Design Time



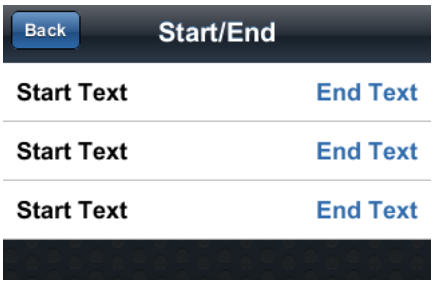
Example 8–16 shows the `listView` element and its child elements defined in an ADF Mobile AMX file. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-startText` places the Output Text component to the start (left side) of the row and applies a black font to its text; setting this attribute to `adfmf-listItem-endText` places the Output Text component to the end (right side) of the row and applies a blue font to its text. Whether or not the arrow pointing to the right is visible is configured by the `showLinkIcon` attribute of the `listItem` element. Since the default value of this attribute is `true`, the example does not contain this setting.

Example 8–16 Definition of List View with Start and End Text

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItema">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
            <amx:cellFormat id="cf1s1" width="10px"/>
            <amx:cellFormat id="cf11" width="60%" height="43px">
               <amx:outputText id="outputText11" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf1s2" width="10px"/>
            <amx:cellFormat id="cf12" halign="end" width="40%">
               <amx:outputText id="outputText12" value="#{row.status}"
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 8–15 shows a List View component with differently styled text added to the start and end of each row. The rows do not contain right-pointing arrows representing links.

Figure 8–15 List View with Start and End Text Without Expansion Links at Design Time



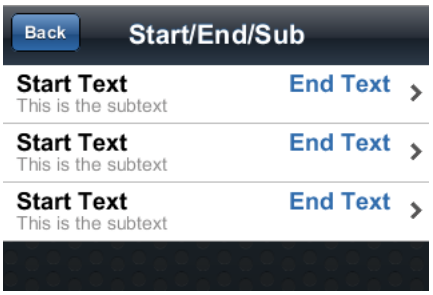
Example 8–17 shows the `listView` element and its child elements defined in an ADF Mobile AMX file. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-startText` places the Output Text component to the start of the row and applies a black font to its text; setting this attribute to `adfmf-listItem-endText` places the Output Text component to the end of the row and applies a blue font to its text. Whether or not the arrow pointing to the right is visible on each particular row is configured by the `showLinkIcon` attribute of the `listItem` element. Since in this example this attribute is set to `false` for every `listItem` element, arrows pointing to the right are not displayed.

Example 8–17 Definition of List View with Start and End Text Without Expansion Links

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItema" showLinkIcon="false">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl1">
            <amx:cellFormat id="cf1s1" width="10px"/>
            <amx:cellFormat id="cf11" width="60%" height="43px">
               <amx:outputText id="outputText11" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf1s2" width="10px"/>
            <amx:cellFormat id="cf12" halign="end" width="40%">
               <amx:outputText id="outputText12" value="#{row.status}"
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 8–16 shows a List View component with differently styled text added to the start and end of each row, and with a subtext added below the end text on the left.

Figure 8–16 List View with Start and End Text and Subtext at Design Time



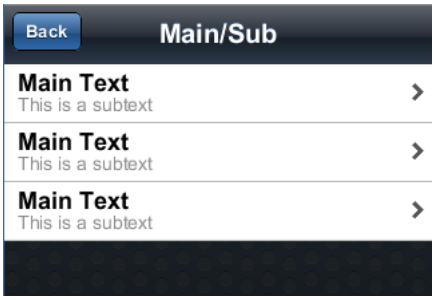
Example 8–18 shows the `listView` element and its child elements defined in an ADF Mobile AMX file. In addition to the text displayed at the start and end of each row, this List View contains subtext placed under the end text. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-upperStartText` places the Output Text component to the left side of the row and applies a black font to its text; setting this attribute to `adfmf-listItem-upperEndText` places the Output Text component to the right side of the row and applies a smaller grey font to its text; setting this attribute to `adfmf-listItem-captionText` places the Output Text component under the Output Text component whose `styleClass` attribute is set to `adfmf-listItem-upperStartText` and applies a smaller grey font to its text.

Example 8–18 Defining List View with Start and End Text and Subtext

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItema">
      <amx:tableLayout id="tl1" width="100%">
         <amx:rowLayout id="rl11">
            <amx:cellFormat id="cf1s1" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cf11" width="60%" height="28px">
               <amx:outputText id="outputTexta1" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf1s2" width="10px"/>
            <amx:cellFormat id="cf12" halign="end" width="40%">
               <amx:outputText id="outputTexta2" value="#{row.status}"
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rl12">
            <amx:cellFormat id="cf13" columnSpan="3" width="100%" height="12px">
               <amx:outputText id="outputTexta3"
                               value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 8–17 shows a List View component with differently styled text added as a main text and subtext to each row.

Figure 8–17 List View with Main Text and Subtext at Design Time



Example 8–19 shows the `listView` element and its child elements defined in an ADF Mobile AMX file. This List View is populated with rows containing a main text and

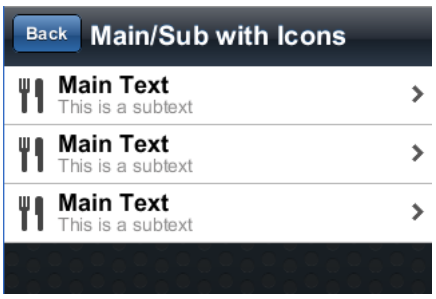
subtext. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-mainText` places the Output Text component to the start of the row and applies a large black font to its text; setting this attribute to `adfmf-listItem-captionText` places the Output Text component under the Output Text component whose `styleClass` attribute is set to `adfmf-listItem-mainText` and applies a smaller grey font to its text.

Example 8–19 Defining List View with Main Text and Subtext

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItema">
      <amx:tableLayout id="tla1" width="100%">
         <amx:rowLayout id="rla1">
            <amx:cellFormat id="cf1s1" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cfa1" width="100%" height="28px">
               <amx:outputText id="outputTexta1" value="#{row.name}"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rla2">
            <amx:cellFormat id="cfa2" width="100%" height="12px" >
               <amx:outputText id="outputTexta2" value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 8–18 shows a List View component with icons and differently styled text added as a main text and subtext to each row.

Figure 8–18 List View with Icons, Main Text and Subtext at Design Time



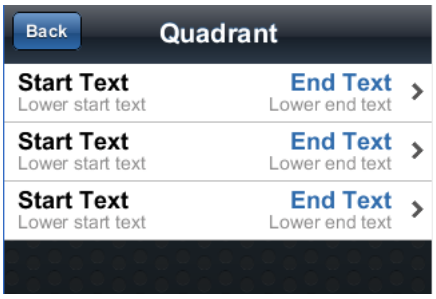
Example 8–20 shows the `listView` element and its child elements defined in an ADF Mobile AMX file. This List View is populated with cells containing an icon, main text, and subtext. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-mainText` places the Output Text component to the left side of the row and applies a large black font to its text; setting this attribute to `adfmf-listItem-captionText` places the Output Text component under the Output Text component whose `styleClass` attribute is set to `adfmf-listItem-mainText` and applies a smaller grey font to its text. The position of the `image` element is defined by its enclosing `cellFormat`.

Example 8–20 Defining List View with Icons, Main Text and Subtext

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItemd">
      <amx:tableLayout id="tld1" width="100%">
         <amx:rowLayout id="rld1">
            <amx:cellFormat id="cfdi" rowSpan="2" width="40px" halign="center">
               <amx:image id="imaged1" source="#{row.image}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cfd1" width="100%" height="28px">
               <amx:outputText id="outputTextd1" value="#{row.name}"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rld2">
            <amx:cellFormat id="cfd2" width="100%" height="12px">
               <amx:outputText id="outputTextd2" value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 8–19 shows a List View component with two types of differently styled text added to the start and end of each row. Besides the text, rows are equipped with a right-pointing arrow representing a link that expands each list item.

Figure 8–19 List View with Four Types of Text at Design Time



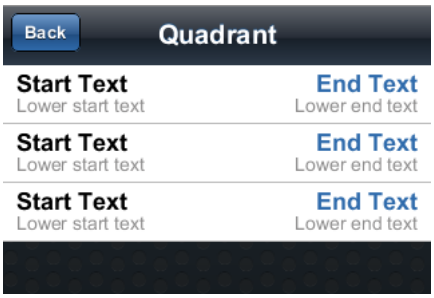
Example 8–21 shows the `listView` element and its child elements defined in an ADF Mobile AMX file. In addition to the text displayed at the start and end of each row, this List View contains two different types of text placed on each side of each row. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-upperStartText` places the Output Text component at the top left corner of the row and applies a large black font to its text; setting this attribute to `adfmf-listItem-upperEndText` places the Output Text component at the top right corner of the row and applies a large grey font to its text; setting this attribute to `adfmf-listItem-lowerStartText` places the Output Text component at the bottom left corner of the row and applies a smaller grey font to its text; setting this attribute to `adfmf-listItem-lowerEndText` places the Output Text component at the bottom right corner of the row and applies a smaller grey font to its text. Whether or not the arrow pointing to the right is visible is configured by the `showLinkIcon` attribute of the `listItem` element. Since the default value of this attribute is `true`, the example does not contain this setting.

Example 8–21 Defining List View with Four Types of Text

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItema">
      <amx:tableLayout id="tla1" width="100%">
         <amx:rowLayout id="rla1">
            <amx:cellFormat id="cf1s1" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cfa1" width="60%" height="28px">
               <amx:outputText id="outputTexta1" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf1s2" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cfa2" width="40%" halign="end">
               <amx:outputText id="outputTexta2" value=#{row.status}
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rla2">
            <amx:cellFormat id="cfa3" width="60%" height="12px">
               <amx:outputText id="outputTexta3" value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
            <amx:cellFormat id="cfa4" width="40%" halign="end">
               <amx:outputText id="outputTexta4" value="#{row.priority}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

Figure 8–20 shows a List View component with two types of differently styled text added to the start and end of each row.

Figure 8–20 List View with Four Types of Text and Without Expansion Links at Design Time



Example 8–22 shows the `listView` element and its child elements defined in an ADF Mobile AMX file. In addition to the text displayed at the start and end of each row, this List View contains two different types of text placed on each side of each row. The way each `outputText` child element is laid out in the list is specified by the `tableLayout` child element of the `listItem`. Alternatively, you may use the `styleClass` attribute to lay out and style `outputText` elements: setting this attribute to `adfmf-listItem-upperStartText` places the Output Text component at the top left corner of the row and applies a large black font to its text; setting this attribute to `adfmf-listItem-upperEndText` places the Output Text component at the top right corner of the row and applies a large grey font to its text; setting this attribute to `adfmf-listItem-lowerStartTextNoChevron` places the Output Text component at the bottom left corner of the row and applies a smaller grey font to its

text; setting this attribute to `adfmf-listItem-lowerEndTextNoChevron` places the Output Text component at the bottom right corner of the row and applies a smaller grey font to its text. Whether or not the arrow pointing to the right is visible on each particular row is configured by the `showLinkIcon` attribute of the `listItem` element. Since in this example this attribute is set to `false` for every `listItem` element, arrows pointing to the right are not displayed.

Example 8–22 Defining List View with Four Types of Text and Without Expansion Links

```
<amx:listView id="listView1" value="#{myBean.listCollection}" var="row">
   <amx:listItem id="listItema" showLinkIcon="false">
      <amx:tableLayout id="tla1" width="100%">
         <amx:rowLayout id="rla1">
            <amx:cellFormat id="cf1s1" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cfa1" width="60%" height="28px">
               <amx:outputText id="outputTexta1" value="#{row.name}"/>
            </amx:cellFormat>
            <amx:cellFormat id="cf1s2" width="10px" rowSpan="2"/>
            <amx:cellFormat id="cfa2" width="40%" halign="end">
               <amx:outputText id="outputTexta2" value=#{row.status}
                               styleClass="adfmf-listItem-highlightText"/>
            </amx:cellFormat>
         </amx:rowLayout>
         <amx:rowLayout id="rla2">
            <amx:cellFormat id="cfa3" width="60%" height="12px">
               <amx:outputText id="outputTexta3" value="#{row.desc}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
            <amx:cellFormat id="cfa4" width="40%" halign="end">
               <amx:outputText id="outputTexta4" value="#{row.priority}"
                               styleClass="adfmf-listItem-captionText"/>
            </amx:cellFormat>
         </amx:rowLayout>
      </amx:tableLayout>
   </amx:listItem>
</amx:listView>
```

8.2.7.4 What You May Need to Know About Using a Static List View

If you create a List View component that is not populated from the model but by hardcoded values, this List View becomes static resulting in some of its properties that you set at design time (for example, `dividerAttribute`, `dividerMode`, `fetchSize`, `moveListener`) ignored at run time.

8.2.8 How to Use a Popup Component

Use the Popup (`popup`) component to display a popup window. You can declare this component as a child of the View component.

You can use the following operations in conjunction with the Popup component:

- Show Popup Behavior (`showPopupBehavior`) operation represents a declarative way to show the Popup in response to a client-triggered event specified using the `type` attribute of the Show Popup Behavior.
- Close Popup Behavior (`closePopupBehavior`) operation represents a declarative way to close the Popup in response to a client-triggered event.

The `Popup Id` attribute of the Show Popup Behavior specifies the unique identifier of the Popup component relative to its parent component. The `Align Id` attribute of the

Show Popup Behavior specifies the unique identifier of the component relative to which the Popup is to be aligned. Since setting identifiers manually is tedious and can lead to invalid references, you set values for these two attributes using an editor that is integrated with the standard Property Inspector (see Figure 8–21). There is an Audit rule that is specifically defined to validate these identifiers (see Section 7.3.2.5, "What You May Need to Know About Element Identifiers and Their Audit").

Example 8–23 shows `popup` and `showPopupBehavior` elements defined in an ADF Mobile AMX file.

Example 8–23 Popup and Show Popup Behavior Definition

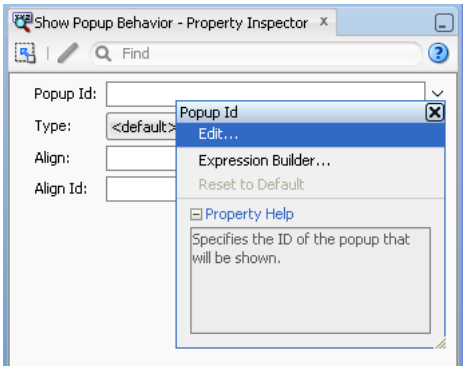
```
<amx:view>
   <amx:panelPage id="panelPage1">
      <amx:commandButton id="commandButton1">
         <amx:showPopupBehavior popupId="popup1" type="action"
                                align="before" alignId="panelPage1" />
      </amx:commandButton>
   </amx:panelPage>
   <amx:popup id="popup1"/>
</amx:view>
```

Popup components can display validation messages when the user input errors occur. For more information, see Section 8.9, "Validating Input."

To set a Popup Id attribute:

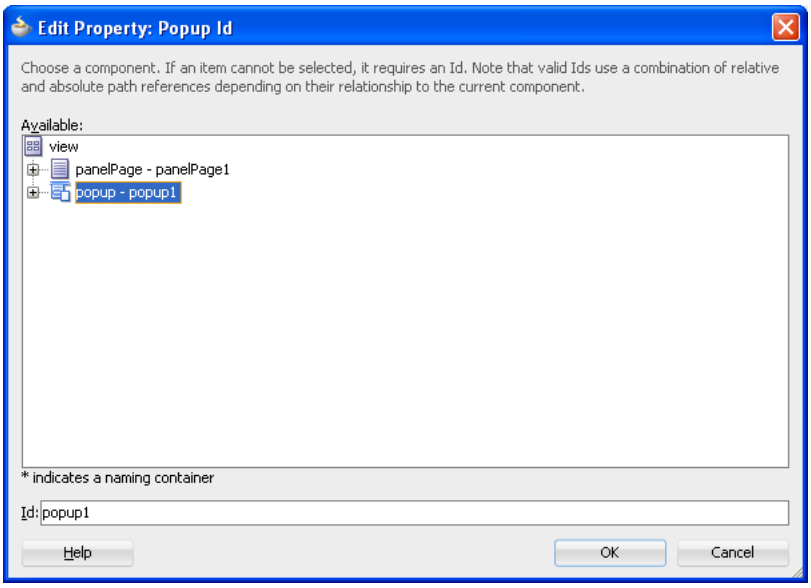
1. Select the `showPopupBehavior` element in the Source editor or Structure pane.
2. Click the down arrow to the right of the **Popup Id** field to open the **Popup Id** dialog, as Figure 8–21 shows.

Figure 8–21 Setting Popup Id Attribute



3. Select **Edit** on the **Popup Id** dialog to open the **Edit Property: Popup Id** dialog that Figure 8–22 shows.

Figure 8–22 Edit Property for Popup Id Dialog

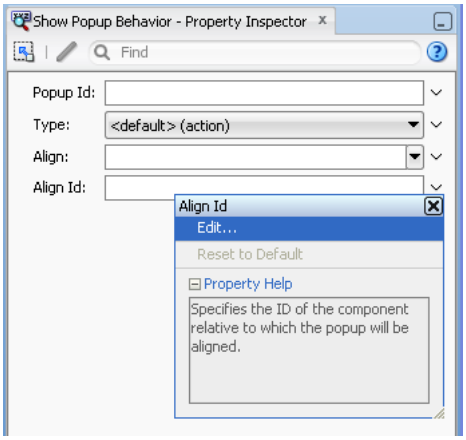


4. Select the Popup component.

To set an Align Id attribute:

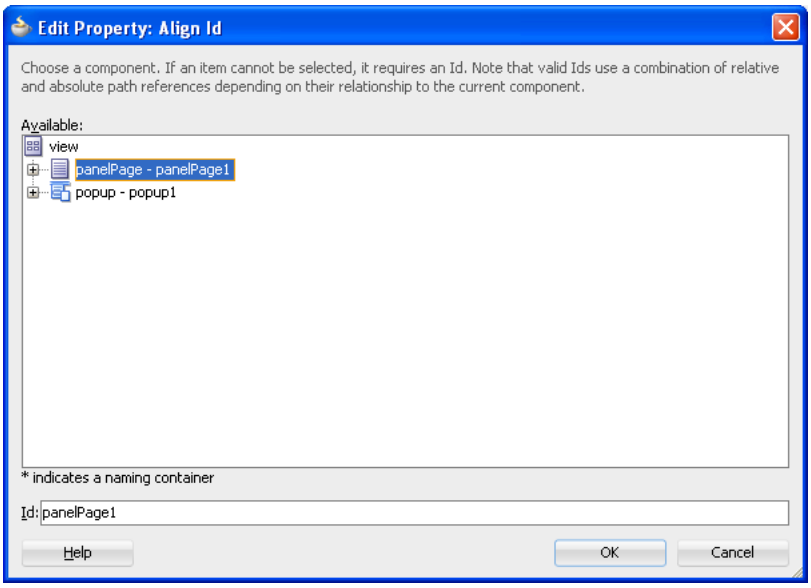
1. Select the `showPopupBehavior` element in the Source editor or Structure pane.
2. Click the down arrow to the right of the **Align Id** field to open the **Align Id** dialog, as Figure 8–23 shows.

Figure 8–23 Setting Align Id Attribute



3. Select **Edit** on the **Popup Id** dialog to open the **Edit Property: Popup Id** dialog that Figure 8–24 shows.

Figure 8–24 Edit Property for Align Id Dialog



4. Select the parent component of the Show Popup Behavior operation.

An ADF Mobile sample application called LayoutDemo demonstrates how to use the Popup component and how to apply styles to adjust the page layout to a specific pattern. The LayoutDemo application is located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer.

8.2.9 How to Use a Panel Splitter Component

Use the Panel Splitter (`panelSplitter`) component to display multiple content areas that may be controlled by a left-side navigation pane. Panel Splitter components are commonly used on tablet devices that have larger display size. These components are typically used with a list on the left and the content on the right side of the display area. Any of the following components can contain a Panel Splitter component:

- Panel Group Layout
- Panel Page
- Popup
- Iterator
- Facet
- Panel Item

A Panel Splitter can contain a Facet (see Section 8.2.6, "How to Use a Facet Component") and Panel Item components. The Panel Item (`panelItem`) component represents the content area of a Panel Splitter. Since each Panel Splitter component must have a least one Panel Item, the Panel Item is automatically added to the Panel Splitter when the Panel Splitter is created. Each Panel Item component can contain any component that a Panel Group Layout can contain (see Section 8.2.3, "How to Use a Panel Group Layout Component").

The left side of the Panel Splitter is represented by a navigator facet (`navigator`), which is optional in cases where only multiple content with animations is desired (for example, drawing a multicontent area with a Select Button that requires animation

when selecting different buttons to switch content). When in landscape mode, this facet is rendered; in portrait mode, a button is placed above the content area and when clicked, the content of the facet is launched in a popup.

Example 8–24 shows the `panelSplitter` element defined in an ADF Mobile AMX file, with the `navigator` facet used as a child component.

Example 8–24 Panel Splitter with Navigator Definition

```
<panelSplitter id="ps1"
               selectedItem="#{bindings.display.inputValue}"
               animation="flipover">
   <facet name="navigator">
      <listView value="#{bindings.data.collectionModel}" var="row">
         <listItem>
            <outputText value="#{row.description}">
            <setPropertyListener from="#{row.type}"
                                 to="#{bindings.display.inputValue}"
                                 type="action"/>
         </listItem>
      </listView>
   </facet>
   <panelItem id="x">
      <panelGroupLayout>
         ...
      </panelGroupLayout>
   </panelItem>
   <panelItem id="y">
      <panelGroupLayout>
         ...
      </panelGroupLayout>
   </panelItem>
</panelSplitter>
```

Example 8–25 shows the `panelSplitter` element defined in an ADF Mobile AMX file. The `navigator` facet is not defined for this panelSplitter. Instead, selection of the layout is enabled through the use of a `selectOneButton`.

Example 8–25 Panel Splitter with Select Button Definition

```
<selectOneButton id="sob1" value="#{bindings.listtype.inputValue}">
   <selectItem label="Contacts" value="contacts"/>
   <selectItem label="Accounts" value="accounts"/>
</selectOneButton>

<panelSplitter id="ps1"
               selectedItem="#{bindings.listtype.inputValue}"
               position="0"
               animation="flipover">
   <panelItem id="contacts" animation="fade">
      <panelGroupLayout>
         ...
      </panelGroupLayout>
   </panelItem>
   <panelItem id="accounts">
      <panelGroupLayout>
         ...
      </panelGroupLayout>
   </panelItem>
</panelSplitter>
```

Example 8–26 shows the `panelSplitter` element defined in an ADF Mobile AMX file, with a `navigator` facet that is itself a panelSplitter with selectOneButton to enable selection of the layout.

Example 8–26 Panel Splitter with Navigator and Select Button Definition

```
<panelSplitter id="ps1"
               selectedItem="#{bindings.display.inputValue}"
               animation="flipover">
   <facet name="navigator">
      <selectOneButton id="sob1" value="#{bindings.listtype.inputValue}">
         <selectItem label="Contacts" value="contacts"/>
         <selectItem label="Accounts" value="accounts"/>
      </selectOneButton>

      <panelSplitter id="sv2"
                     selectedItem="#{bindings.listtype.inputValue}"
                     navigatorWidth="0"
                     animation="flipup">
         <panelItem id="contacts">
            <panelGroupLayout>
               ...
               <commandButton ...>
                  <setPropertyListener from="x"
                                       to="#{bindings.display.inputValue}"
                                       type="action"/>
               </commandButton>
            </panelGroupLayout>
         </panelItem>
         <panelItem id="accounts">
            <panelGroupLayout>
               ...
               <commandLink ...>
                  <setPropertyListener from="y"
                                       to="#{bindings.display.inputValue}"
                                       type="action"/>
               </commandLink>
            </panelGroupLayout>
         </panelItem>
      </panelSplitter>
   </facet>
   <panelItem id="x">
      <panelGroupLayout>
         ...
      </panelGroupLayout>
   </panelItem>
   <panelItem id="y">
      <panelGroupLayout>
         ...
      </panelGroupLayout>
   </panelItem>
</panelSplitter>
```

For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.2.10 How to Use a Table Layout Component

Use the Table Layout (`tableLayout`) component to display data in a typical table format that consists of rows containing cells.

The Row Layout (`rowLayout`) component represents a single row in the Table Layout. The Table Layout component must contain either one or more Row Layout components or Iterator components that can produce Row Layout components.

The CellFormat (`cellFormat`) component represents a cell in the Row Layout. The Row Layout component must contain either one or more CellFormat components or Iterator components that can produce CellFormat components.

The Table Layout structure does not allow cell contents to use percentage heights nor can a height be assigned to the overall table structure as a whole. For details, see the description of `layout`, `width`, and `height` attributes of the Table Layout, Row Layout, and Cell Format components in the *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

To add the Table Layout component:

1. In the **Component Palette**, drag and drop a **Table Layout** to the ADF Mobile AMX page.
2. Insert the desired number of Row Layout or Iterator child components into the Table Layout component.
3. Insert Cell Format or Iterator child components into each Row Layout component.
4. Use the **Property Inspector** to set the attributes of all added components. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

Example 8–27 shows the `tableLayout` element and its children defined in an ADF Mobile AMX file.

Example 8–27 Defining Table Layout

```
<amx:tableLayout id="tableLayout1"
                 rendered="#{pageFlowScope.pRendered}"
                 styleClass="#{pageFlowScope.pStyleClass}"
                 inlineStyle="#{pageFlowScope.pInlineStyle}"
                 borderWidth="#{pageFlowScope.pBorderWidth}"
                 cellPadding="#{pageFlowScope.pCellPadding}"
                 cellSpacing="#{pageFlowScope.pCellSpacing}"
                 halign="#{pageFlowScope.pHalign}"
                 layout="#{pageFlowScope.pLayoutTL}"
                 shortDesc="#{pageFlowScope.pShortDesc}"
                 summary="#{pageFlowScope.pSummary}"
                 width="#{pageFlowScope.pWidth}">
   <amx:rowLayout id="rowLayout1">
      <amx:cellFormat id="cellFormatA" rowSpan="2" halign="center">
         <amx:outputText id="otA" value="Cell A"/>
      </amx:cellFormat>
      <amx:cellFormat id="cellFormatB" rowSpan="2" halign="center">
         <amx:outputText id="otB" value="Cell B (wide content)"/>
      </amx:cellFormat>
      <amx:cellFormat id="cellFormatC" rowSpan="2" halign="center">
         <amx:outputText id="otC" value="Cell C"/>
      </amx:cellFormat>
   </amx:rowLayout>
   <amx:rowLayout id="rowLayout2">
      <amx:cellFormat id="cellFormatD" halign="end">
         <amx:outputText id="otD" value="Cell D"/>
      </amx:cellFormat>
      <amx:cellFormat id="cellFormatE">
         <amx:outputText id="otE" value="Cell E"/>
      </amx:cellFormat>
```

```
    </amx:rowLayout>
</amx:tableLayout>
```

8.3 Creating and Using UI Components

You can use the following UI components when developing your ADF Mobile AMX feature:

- Input Text (see Section 8.3.1, "How to Use the Input Text Component")
- Input Number Slider (see Section 8.3.2, "How to Use the Input Number Slider Component")
- Input Date (see Section 8.3.3, "How to Use the Input Date Component")
- Output Text (see Section 8.3.4, "How to Use the Output Text Component")
- Button (see Section 8.3.5, "How to Use Buttons")
- Link (see Section 8.3.6, "How to Use Links")
- Image (see Section 8.3.7, "How to Display Images")
- Checkbox (see Section 8.3.8, "How to Use the Checkbox Component")
- Select Many Checkbox (see Section 8.3.9, "How to Use the Select Many Checkbox Component")
- Select Many Choice (see Section 8.3.11, "How to Use the Select Many Choice Component")
- Boolean Switch (see Section 8.3.12, "How to Use the Boolean Switch Component")
- Choice (see Section 8.3.10, "How to Use the Choice Component")
- Select Button (see Section 8.3.13, "How to Use the Select Button Component")
- Radio Button (see Section 8.3.14, "How to Use the Radio Button Component")
- Carousel (see Section 8.3.15, "How to Use Carousel Component")
- Verbatim (see Section 8.3.16, "How to Use Verbatim Component")

You can also use the following miscellaneous components that include operations, listener-type components, and converters as children of the UI components when developing your ADF Mobile AMX feature:

- Iterator (see Section 8.3.17, "How to Enable Iteration")
- Load Bundle (see Section 8.3.18, "How to Load a Resource Bundle")
- Action Listener (see Section 8.3.19, "How to Use the Action Listener")
- Set Property Listener (see Section 8.3.20, "How to Use the Set Property Listener")
- Convert Date Time (see Section 8.3.21, "How to Convert Date and Time Values")
- Convert Number (see Section 8.3.22, "How to Convert Numerical Values")

You add a UI component by dragging and dropping it onto the ADF Mobile AMX page from the Component palette (see Section 7.3.2.1, "Adding UI Components"). Then you use the Property Inspector to set the component's attributes (see Section 7.3.2.3, "Configuring UI Components"). For information on attributes of each particular component, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

Note: On an ADF Mobile AMX page, you place UI components within layout components (see Section 8.2, "Designing the Page Layout"). UI elements are declared under the `<amx>` namespace.

You can add event listeners to some UI components. For more information, see Section 8.10, "Using Event Listeners." Event listeners are applicable to components for the ADF Mobile AMX run-time description on both iOS and Android-powered devices, but the listeners do not have any effect at design time.

An ADF Mobile sample application called CompGallery demonstrates how to create and configure ADF Mobile AMX UI components. Another sample application called LayoutDemo shows how to lay out components on an ADF Mobile AMX page. The sample applications are located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer.

8.3.1 How to Use the Input Text Component

The Input Text (`inputText`) component represents an editable text field. The following types of Input Text components are available:

- Standard single-line Input Text, which is declared as an `inputText` element in an ADF Mobile AMX file:

```
<amx:inputText id="text1"
                  label="Text Input:"
                  value="#{myBean.text}" />
```

- Password Input Text:

```
<amx:inputText id="text1"
               label="Password Input:"
               value="#{myBean.text}"
               secret="true" />
```

- Multiline Input Text (also known as text area):

```
<amx:inputText id="text1"
               label="Textarea:"
               value="#{myBean.text}"
               simple="true"
               rows="4" />
```

Figure 8–25 shows the Input Text component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:inputText id="inputText1"
               label="Input Text"
               value="text"/>
```

Figure 8–25 Input Text at Design Time

The `inputType` attribute lets you define how the component interprets the user input: as a text (default), email address, number, telephone number, or URL. These input types are based on the values allowed by HTML5.

To enable conversion of numbers, as well as date and time values that are entered in the Input Text component, you use the Convert Number (see Section 8.3.22, "How to Convert Numerical Values") and Convert Date Time (see Section 8.3.21, "How to Convert Date and Time Values") components.

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer.

On some mobile devices, when the end user taps an Input Text field, the keyboard is displayed (slides up). If an Input Text is the only component on an ADF Mobile AMX page, the input focus is on this field and the keyboard is displayed by default when the page loads.

A multiline Input Text may be displayed on a secondary page where it is the only component, in which case the multiline Input Text receives focus when the page loads and the keyboard becomes visible.

Input Text components render and behave differently on iOS and Android-powered devices: on iPhone and iPad, Input Text components may be displayed with or without a border.

When creating an Input Text component, consider the following:

- If a field represented by an Input Text component is displayed with a border, the field has a fixed height and rounded corners.
- To input or edit content, the end user has to tap in the field, which triggers a blinking insertion cursor to be displayed at the point of the tap, allowing the end user to edit the content. If the field does not contain content, the insertion cursor is positioned at the start of the field.
- Fields represented by Input Text components may contain default text, typically used as a prompt. When the end user taps a key on the keyboard in such a field, the default text clears when Edit mode is entered.
- Fields represented by Input Text components do not have a selected appearance. Selection is indicated by the blinking insertion cursor within the field.
- If the end user enters more text than fits in the field, the text content shifts left one character at a time as the typing continues.

A multiline Input Text component is rendered as a rectangle of any height with rounded corners. This component supports scrolling when the content is too large to fit within the boundaries of the field: rows of text scroll up as the text area fills and new rows of text are added. The end user may flick up or down to scroll rows of text if there are more rows than can be displayed in the given display space. A scroll bar is displayed within the component to indicate the area is being scrolled.

Password field briefly echoes each typed character, and then reverts the character to a dot to protect the password.

8.3.2 How to Use the Input Number Slider Component

The Input Number Slider (`inputNumberSlider`) component enables selection of numeric values from a range of values by using a slider instead of entering the value

by using keys. The filled portion of the trough or track of the slider visually represents the current value.

The Input Number Slider may be used in conjunction with the Output or Input Text component to numerically show the value of the slider. The Input Text component also allows direct entry of a slider value: when the end user taps the Input Text field, the keyboard in numeric mode slides up; the keyboard can be dismissed by either using the slide-down button or by tapping away from the slider component.

The Input Number Slider component always shows the minimum and maximum values within the defined range of the component.

> **Note:** The Input Number Slider component should not be used in cases where a precise numeric entry is required or where there is a wide range of values (for example, 0 to 1000).

Example 8–28 demonstrates the `inputNumberSlider` element defined in an ADF Mobile AMX file.

Example 8–28 Input Number Slider Definition

```
<amx:inputNumberSlider id="slider1" value="#{myBean.count}"/>
```

Figure 8–26 shows the Input Number Slider component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:inputNumberSlider id="inputNumberSlider1"
                       label="Input Number"
                       minimum="0"
                       maximum="20"
                       stepSize="1"
                       value="10"/>
```

Figure 8–26 Input Number Slider at Design Time

To enable conversion of numbers that are entered in the Input Number Slider component, you use the Convert Number component (see Section 8.3.22, "How to Convert Numerical Values").

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer.

Similar to other ADF Mobile AMX UI components, the Input Number Slider component has a normal and selected state. The component is in its selected state at any time it is touched. To change the slider value, the end user touches, and then interacts with the slider button.

8.3.3 How to Use the Input Date Component

The Input Date (`inputDate`) component presents a popup input field for entering dates. The default date format is the short date format appropriate for the current locale. For example, the default format in American English (ENU) is `mm/dd/yy`. The `inputType` attribute defines if the component accepts date, time, or date and time as an input. The time zone depends on the time zone configured for the mobile device, and, therefore, it is relative to the device. At run time, the Input Date component has the device's native look and feel.

Example 8–29 demonstrates the `inputDate` element defined in an ADF Mobile AMX file. The `inputType` attribute of this component is set to the default value of `date`.

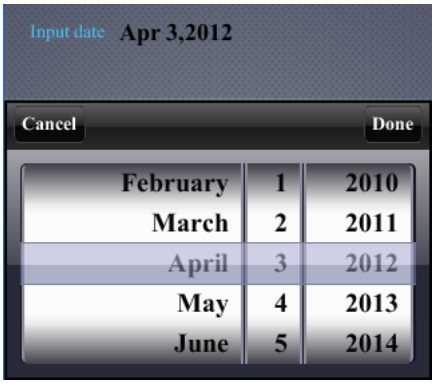
Example 8–29 Input Date Definition

```
<amx:inputDate id="inputDate1" label="Input Date" value="#{myBean.date}"/>
```

Note: The `value` attribute can only be specified as an EL expression.

Figure 8–27 shows the Input Date component displayed in the Preview pane.

Figure 8–27 Input Date at Design Time



For more information, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- *Global Dates and Times* defined by W3C
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.4 How to Use the Output Text Component

ADF Mobile AMX provides the Output Text (`outputText`) component for you to use as a label to display text.

Example 8–30 demonstrates the `outputText` element defined in an ADF Mobile AMX file.

Example 8–30 Output Text Definition

```
<amx:outputText id="ot1"
```

```
value="output"
styleClass="#{pageFlowScope.pStyleClass}"/>
```

Figure 8–28 shows the Output Text component displayed in the Preview pane.

Figure 8–28 Output Text at Design Time

You use the Convert Number (see Section 8.3.22, "How to Convert Numerical Values") and Convert Date Time (see Section 8.3.21, "How to Convert Date and Time Values") converters to facilitate the conversion of numerical and date-and-time-related data for the Output Text components.

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery and LayoutDemo, ADF Mobile sample applications located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.5 How to Use Buttons

The Button (`commandButton`) component is used to trigger actions (for example, Save, Cancel, Send) and to enable navigation to other pages within the application (for example, Back: see Section 8.3.5.6, "Enabling the Back Button Navigation" for more information).

You may use the Button in one of the following ways:

- Button with a text label.
- Button with a text label and an image icon.

> **Note:** You may define the icon image and placement as left or right of the text label.

- Button with an image icon only (for example, the " + " and " - " buttons for adding or deleting records).

ADF Mobile supports one default Button type for the following three display areas:

1. Buttons that appear in the top header bar: in ADF Mobile AMX pages, the header is represented by the Panel Page component (see Section 8.2.2, "How to Use a Panel Page Component") in combination with the header, primary, and secondary facets, which is typical on iPhones:
   - Header Facet contains the page title.
   - Primary Action Facet represents an area that appears in the left corner of the header bar and typically hosts Button or Link components, but can contain any component type.
   - Secondary Action Facet represents an area that appears in the right corner of the header bar and typically hosts Button or Link components, but can contain any component type.
2. Buttons that appear in the content area of a page.

3. Buttons that appear in the footer bar of a page. In ADF Mobile AMX pages, the footer is represented by the Panel Page component (see Section 8.2.2, "How to Use a Panel Page Component") in combination with the footer facet:
   - Footer Facet represents an area that appears below the content area and typically hosts Button or Link components, but can contain any component type.

All Button components of any type have three states:

1. Normal.
2. Selected: represents appearance when the Button is tapped or touched by the end user. When a button is tapped (touch and release), the button action is performed. Upon touch, the selected appearance is displayed; upon release, the action is performed. If the end user touches the button, then drags their finger away from the button, the action is not performed. However, for the period of time the button is touched, the selected appearance is displayed.
3. Disabled.

The appearance of a Button component is defined by its `styleClass` attribute that you set to an `adfmf-commandButton-<style>`. You can apply any of the styles detailed in Table 8–6 to a Button placed in any valid location within the ADF Mobile AMX page.

Table 8–6 Button Styles

Button Style Name	Description
Default	The default style of a Button placed: ■ In any of the Panel Page facets (Primary, Secondary, Header, Footer). For more information, see Section 8.3.5.1, "Displaying Default Style Buttons." ■ Anywhere in the content area of an ADF Mobile AMX page. This style is used for buttons that are to perform specific actions within a page, typically based on their location or context within the page.
Back	The back style of a Button placed in any of the Panel Page facets (Primary, Secondary, Header, Footer). This style may be applied to the default Button to give the "back to page" appearance. This button style is typical for "Back to Springboard" or any "Back to Page" buttons. For more information, see Section 8.3.5.2, "Displaying Back Style Buttons."
Highlight	The highlight style of a Button placed in any of the Panel Page facets (Primary, Secondary, Header, Footer) or the content area of an ADF Mobile AMX page. This style may be added to a Button to provide the iPhone button appearance typical of Save (or Done) buttons. For more information, see Section 8.3.5.3, "Displaying Highlight Style Buttons."
Alert	The Alert style adds the delete appearance to a button. For more information, see Section 8.3.5.4, "Displaying Alert Style Buttons."

There is an additional Rounded style (`adfmf-commandButton-rounded`) that you can apply to a Button to decorate it with a thick rounded border. You can define this style in combination with any other style.

There is a particular order in which ADF Mobile AMX processes the Button component's child operations and attributes. For more information, see Section 8.3.5.7,

"What You May Need to Know About the Order of Processing Operations and Attributes."

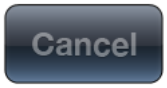
8.3.5.1 Displaying Default Style Buttons

Various types of default style buttons that are placed within Panel Page facets or content area are displayed on the mobile device as follows:

- Normal Button with a text label only:

- Selected Button with a text label only:

- Disabled Button with a text label only:



- Normal Button with an image icon only:

- Selected Button with an image icon only:

- Disabled Button with an image icon only:

Example 8–31 and Example 8–32 demonstrate the `commandButton` element declared in an ADF Mobile AMX file.

Example 8–31 Default Button with Text Label Definition

```
<amx:panelPage id="pp1">
   .<amx:facet name="primary">
      <amx:commandButton id="cb1"
                         text="Cancel"
                         action="cancel"
                         actionListener="#{myBean.rollback}"/>
   </amx:facet>
</amx:panelPage>
```

Example 8–32 Default Button with Image Icon Definition

```
<amx:panelPage id="pp1">
```

```
   .<amx:facet name="primary">
      <amx:commandButton id="cb1"
                         icon="plus.png"
                         action="add"
                         actionListener="#{myBean.AddItem}"/>
   </amx:facet>
</amx:panelPage>
```

Example 8–33 shows the `commandButton` element declared inside the Panel Page's footer facet.

Example 8–33 Default Button with Text Label and Image in Footer Facet Definition

```
<amx:panelPage id="pp1">
   <amx:facet name="footer">
      <amx:commandButton id="cb2"
                         icon="folder.png"
                         text="Move (#{myBean.mailcount})"
                         action="move"/>
   </amx:facet>
</amx:panelPage>
```

Example 8–34 demonstrates the `commandButton` element declared as a part of the Panel Page content area.

Example 8–34 Default Button with Text Label Definition in the Page Content Area

```
<amx:panelPage id="pp1">
   <amx:commandButton id="cb1"
                      text="Reply"
                      actionListener="#{myBean.share}"/>
</amx:panelPage>
```

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

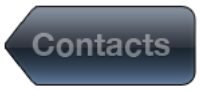
8.3.5.2 Displaying Back Style Buttons

Various types of back style buttons that are placed within Panel Page facets or content area are displayed on the mobile device as follows:

- Normal Button with a text label only:

- Selected Button with a text label only:

- Disabled Button with a text label only:



- Normal Button with an image icon only:

- Selected Button with an image icon only:

- Disabled Button with an image icon only:

Example 8–35 demonstrates the `commandButton` element declared in an ADF Mobile AMX file.

Example 8–35 Back Button with Text Label Definition

```
<amx:panelPage id="pp1">
   <amx:facet name="primary">
      <amx:commandButton id="cb1"
                         text="Back"
                         action"__back"/>
   </amx:facet>
</amx:panelPage>
```

Every time you place a Button component within the primary facet and set its `action` attribute to `__back`, ADF Mobile AMX automatically applies the back arrow styling to it.

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.5.3 Displaying Highlight Style Buttons

Various types of highlight style buttons that are placed within Panel Page facets or content area are displayed on the mobile device as follows:

- Normal Button

- Selected Button

- Disabled Button

Example 8–36 demonstrates the `commandButton` element declared in an ADF Mobile AMX file.

Example 8–36 Highlight Button with Text Label Definition

```
<amx:panelPage id="pp1">
   <amx:facet name="secondary">
      <amx:commandButton id="cb2"
                         text="Save"
                         action"save"
                         styleClass="adfmf-commandButton-highlight"/>
   </amx:facet>
</amx:panelPage>
```

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.5.4 Displaying Alert Style Buttons

Various types of alert style buttons that are placed within Panel Page content are displayed on the mobile device as follows:

- Normal Button with a text label only:

- Selected Button with a text label only:

- Disabled Button with a text label only:

Example 8–37 demonstrates the `commandButton` element declared in an ADF Mobile AMX file.

Example 8–37 Alert Button with Text Label Definition

```
<amx:commandButton id="cb1"
                   text="Delete"
```

```
                actionListener="#{myBean.delete}"
                styleClass="adfmf-commandButton-alert" />
```

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.5.5 Using Buttons Within the Application

In your ADF Mobile application, you can use the Button component within the following contexts:

- Navigation Bar
- The Content Area to perform specific actions
- Action Sheets
- Popup-style Alert Messages

Navigation Bar

ADF Mobile lets you create standard buttons for use on a navigation bar:

- Edit button allows the end user to enter an editing or content-manipulation mode:

- Cancel button allows the end user to exit the editing or content-manipulation mode without saving changes.
- Save button allows the end user to exit the editing or content-manipulation mode by saving changes.

- Done button allows the end user to exit the current mode and save changes, if any.
- Undo button allows the end user to undo the most recent action.
- Redo button allows the end user to redo the most recent undone action.
- Back button allows the end user to navigate back to the springboard.

- Back to Page button allows the end user to navigate back to the page identified by the button text label.

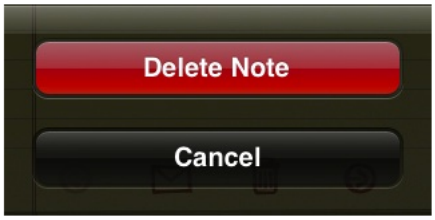
- Add button allows the end user to add or create a new object.

Content Area

Buttons that are positioned within the content area of a page perform a specific action given the location and context of the button within the page. These buttons may have a different visual appearance than buttons positioned with the navigation bar:

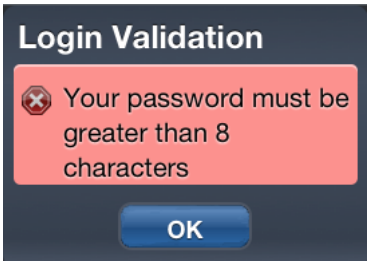
Action Sheets

The following is an example of buttons placed within an action sheet:



Alert Messages

The following is an example of a button placed within a validation message:



8.3.5.6 Enabling the Back Button Navigation

ADF Mobile AMX supports navigation using the back button, with the default behavior of going back to the previously visited page. For more information, see Section 7.2.9, "How to Specify Action Outcomes Using UI Components."

If any Button component is added to the primary facet of a Panel Page that is equipped with the `__back` navigation, this Button is automatically given the back arrow visual styling (see Section 8.3.5.2, "Displaying Back Style Buttons"). To disable this, set the `styleClass` attribute to `amx-commandButton-normal`.

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.5.7 What You May Need to Know About the Order of Processing Operations and Attributes

The following is the order in which ADF Mobile AMX processes operations and attributes when such components as a Button, Link, and List Item are activated:

1. The following child operations are processed in the order they appear in the XML file:
   - Set Property Listener
   - Action Listener
   - Show Popup Behavior
   - Close Popup Behavior
2. The Action Listener (`actionListener`) attribute is processed and the associated Java method is invoked.
3. The Action (`action`) attribute is processed and any navigation case is followed.

8.3.6 How to Use Links

You use the Link (`commandLink`) component to trigger actions and enable navigation to other views.

Similar to the Panel Group Layout, the Link component can have any type of component defined as its child. By using such components as Set Property Listener (see Section 8.3.20, "How to Use the Set Property Listener"), Action Listener (see Section 8.3.19, "How to Use the Action Listener"), Show Popup Behavior, and Close Popup Behavior see Section 8.2.8, "How to Use a Popup Component") as children of the Link component, you can create an actionable area within which clicks and gestures can be performed.

By placing an Image component (see Section 8.3.7, "How to Display Images") inside a Link you can create a clickable image.

Example 8–38 demonstrates the `commandLink` element declared in an ADF Mobile AMX file.

Example 8–38 Link Definition

```
<amx:commandLink id="cl1"
                 text="linked"
                 action="gotolink"
                 actionListener="#{myBean.doSomething}"/>
```

Figure 8–29 shows the Link component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:commandLink id="commandLink1"
                 text="Link"
                 action="__back"/>
```

Figure 8–29 Link at Design Time

There is a particular order in which ADF Mobile AMX processes the Link component's child operations and attributes. For more information, see Section 8.3.5.7, "What You May Need to Know About the Order of Processing Operations and Attributes."

ADF Mobile AMX provides another component that is similar to the Link, but does not allow for navigation between pages: Link Go (`goLink`) component. You use this component to enable linking to external pages. Figure 8–30 shows the Link Go component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:goLink id="goLink1"
            text="Go Link"
            url="http://example.com"/>
```

Figure 8–30 Link Go at Design Time

Image is the only component that you can specify as a child of the Link Go component.

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.7 How to Display Images

ADF Mobile AMX enables the display of images on iOS and Android-powered devices using the Image (`image`) component represented by a bitmap.

In addition to placing an Image in a Button and List View, you can place it inside a Link component (see Section 8.3.6, "How to Use Links") to create a clickable image.

Example 8–39 demonstrates the `image` element definition in an ADF Mobile AMX file.

Example 8–39 Image Definition

```
<amx:image id="i1"
           styleClass="prod-thumb"
           source="images/img-big-#{pageFlowScope.product.uid}.png" />
```

The following are supported formats on Android platform:

- GIF
- JPEG
- PNG
- BMP

The following are supported formats on iOS platform:

- PNG

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*

- CompGallery and LayoutDemo, ADF Mobile sample applications located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.8 How to Use the Checkbox Component

The Checkbox (`selectBooleanCheckbox`) component represents a check box that you create to enable single selection of `true` or `false` values, which allows toggling between selected and deselected states.

You can use the `label` attribute of the Checkbox component to place text to the left of the checkbox, and the `text` attribute places text on the right.

Example 8–40 demonstrates the `selectBooleanCheckbox` element declared in an ADF Mobile AMX file.

Example 8–40 Checkbox Definition

```
<amx:selectBooleanCheckbox id="check1"
                           label="Agree to the terms:"
                           value="#{myBean.bool1}"/>
```

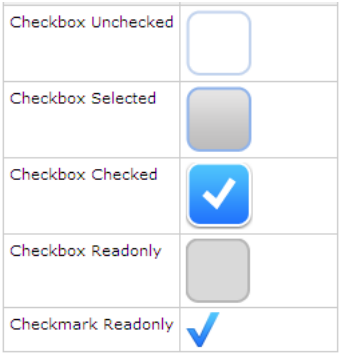
Figure 8–31 shows the Checkbox component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectBooleanCheckbox id="selectBooleanCheckbox1"
                           label="Checkbox"
                           value="false"/>
```

Figure 8–31 Checkbox at Design Time

Figure 8–32 shows the visual representation of the Checkbox component in its various states.

Figure 8–32 States of Checkbox



For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_`*

install/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples directory on your development computer

8.3.8.1 Support for Checkbox Components on iOS Platform

iOS does not support a native Checkbox component. The Boolean Switch is usually used in Properties pages to enable a boolean selection (see Section 8.3.12, "How to Use the Boolean Switch Component").

8.3.8.2 Support for Checkbox Components on Android Platform

Android provides support for a native Checkbox component. This component is used extensively on Settings pages to turn on or off individual setting values.

8.3.9 How to Use the Select Many Checkbox Component

The Select Many Checkbox (`selectManyCheckbox`) component represents a group of check boxes that you use to enable multiple selection of `true` or `false` values, which allows toggling between selected and deselected states of each check box in the group. The selection mechanism is provided by the Select Items or Select Item component (see Section 8.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Select Many Checkbox component.

> **Note:** The Select Many Checkbox component can contain one Select Items component or one or more Select Item components.

Example 8–41 demonstrates the `selectManyCheckbox` element declared in an ADF Mobile AMX file.

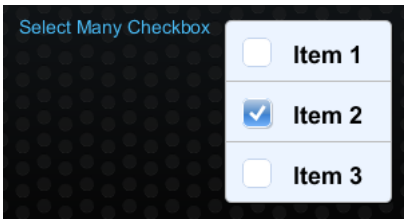
Example 8–41 Select Many Checkbox Definition

```
<amx:selectManyCheckbox id="selectManyCheckbox1"
                        label="Select shipping options"
                        value="#{myBean.shipping}"
   <amx:selectItem label="Air" value="#{myBean.shipping.air}"/>
   <amx:selectItem label="Rail" value="#{myBean.shipping.rail}"/>
   <amx:selectItem label="Water" value="#{myBean.shipping.water}"/>
</amx:selectManyCheckbox>
```

Figure 8–33 shows the Select Many Checkbox component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectManyCheckbox id="selectManyCheckbox1"
                        label="Select Many Checkbox"
                        value="value2">
   <amx:selectItem label="Selection 1" value="value1"/>
   <amx:selectItem label="Selection 2" value="value2"/>
   <amx:selectItem label="Selection 3" value="value3"/>
</amx:selectManyCheckbox>
```

Figure 8–33 Select Many Checkbox at Design Time



For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.9.1 What You May Need to Know About the User Interaction with Select Many Checkbox Component

ADF Mobile AMX provides two alternative ways for displaying the Select Many Checkbox component: pop-up style (default) and list style that is used when the number of available choices exceeds the device screen size.

The end user interaction with a pop-up style Select Many Checkbox component on both iPhone and iPad occurs as follows: when the end user taps the component, the list of choices is displayed in a popup. To make a choice, the end user taps one or more choices. To save the selections, the end user either taps outside the popup or closes the popup using the close (" x ") button.

Upon closing of the popup, the value displayed in the component is updated with the selected value.

When the number of choices exceed the dimensions of the device, a full-page popup containing a scrollable List View (see Section 8.2.7, "How to Use List View and List Item Components") is generated.

The end user interaction with a list-style Select Many Checkbox component on both iPhone and iPad occurs as follows: when the end user taps the component, the list of choices is displayed. To make a choice, the end user scrolls up or down to browse available choices, and then taps one or more choices. To save the selections, the end user taps the close (" x ") button.

Upon closing of the list, the value displayed in the component is updated with the selected value.

Note: In both cases, there is no mechanism provided to cancel the selection.

8.3.10 How to Use the Choice Component

The Choice (`selectOneChoice`) component represents a combo box that is used to enable selection of a single value from a list. The selection mechanism is provided by the Select Items or Select Item component (see Section 8.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Choice component.

Note: The Choice component can contain one Select Items component or one or more Select Item components.

Example 8–42 demonstrates the `selectOneChoice` element definition with the `selectItems` subelement in an ADF Mobile AMX file.

Example 8–42 Choice Definition Using Select Item Component

```
<amx:selectOneChoice id="choice1"
                     label="Your state:"
                     value="#{myBean.myState}">
   <amx:selectItem label="Alaska" value="AK"/>
   <amx:selectItem label="Alabama" value="AL"/>
   <amx:selectItem label="California" value="CA"/>
   <amx:selectItem label="Connecticut" value="CT"/>
</amx:selectOneChoice>
```

Example 8–43 Choice Definition Using Select Items Component

```
<amx:selectOneChoice id="choice1"
                     label="Your state:"
                     value="#{myBean.myState}">
   <amx:selectItems value="myBean.allStates"/>
</amx:selectOneChoice>
```

Figure 8–34 shows the Choice component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectOneChoice id="selectOneChoice1"
                     label="Choice"
                     value="value1">
   <amx:selectItem label="Value 1" value="value1"/>
   <amx:selectItem label="Value 2" value="value2"/>
   <amx:selectItem label="Value 3" value="value3"/>
</amx:selectOneChoice>
```

Figure 8–34 Choice at Design Time

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.10.1 What You May Need to Know About the User Interaction with Choice Component on iOS Platform

ADF Mobile AMX provides two alternative ways for displaying the Choice component: pop-up style and drop-down style.

On an iPhone, the end user interaction with a native Choice component occurs as follows: when the end user taps the component, the list of choices is displayed, with the first option selected by default. To make a choice, the end user scrolls up or down

to browse available choices. To save the selection, the end user taps Done in the tool bar.

On an iPad, the user interaction is similar to the interaction on an iPhone, except the following:

- The list of choices is displayed in a popup dialog.
- iPad styling is implemented around the list of choices, with a notch used to indicate the source of the list.

To close the list of choices without selecting an item, the end user must tap outside the popup dialog.

> **Note:** The UI to display the list of choices and the tool bar are native to the browser and cannot be styled using CSS.

List values within the Choice component may be displayed as disabled.

When the number of choices exceeds the dimensions of the device display, a list page is generated that may be scrolled in a native way.

8.3.10.2 What You May Need to Know About the User Interaction with Choice Component on Android Platform

The end user interaction with a native Choice component on an Android-powered device occurs as follows: when the end user taps the component, the list of choices in the form of a popup dialog is displayed. A simple popup is displayed if the number of choices fits within the dimensions of the device, in which case:

- A single tap on an item from the selection list selects that item and closes the popup; the selection is reflected in the Choice component label.
- A single tap outside the popup or a click on the Back key closes the popup with no changes applied.

If the number of choices to be displayed does not fit within the device dimensions, the popup contains a scrollable list, in which case:

- A single tap on an item from the selection list selects that item and closes the popup; the selection is reflected in the Choice component label.
- A click on the Back key closes the popup with no changes applied.

8.3.10.3 What You May Need to Know About Differences Between Select Items and Select Item Components

The Select Items (`selectItems`) component is patterned after the JSF `selectItems` element and provides a list of objects that can be selected in multiple-selection components. For more information, see JSF Toolbox page at `http://www.jsftoolbox.com`.

The Select Item (`selectItem`) component is patterned after ADF's `selectItem` element and represents a single selectable item of selection components. For more information, see <af:selectItem> page in *Oracle Fusion Middleware Tag Reference for Oracle ADF Faces*.

8.3.11 How to Use the Select Many Choice Component

The Select Many Choice (`selectManyChoice`) component allows selection of multiple values from a list. The selection mechanism is provided by the Select Items or Select Item component (see Section 8.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Select Many Checkbox component.

> **Note:** The Select Many Checkbox component can contain one Select Items component or one or more Select Item components.

Example 8–44 demonstrates the `selectManyChoice` element declared in an ADF Mobile AMX file.

Example 8–44 Select Many Choice Definition Using Select Item Component

```
<amx:selectManyChoice id="check1"
                      label="Select Shipping Options:"
                      value="#{myBean.shipping}">
   <amx:selectItem label="Signature Required" value="signature" />
   <amx:selectItem label="Insurance" value="insurance" />
   <amx:selectItem label="Delivery Confirmation" value="deliveryconfirm"/>
</amx:selectManyChoice>
```

Example 8–45 Select Many Choice Definition Using Select Items Component

```
<amx:selectManyChoice id="check1"
                      label="Select Shipping Options:"
                      value="#{myBean.shipping}">
   <amx:selectItems value="#{myBean.shippingOptions}"/>
</amx:selectManyChoice>
```

At design time, the Select Many Choice component looks identical to the Choice component. For more information, see Figure 8–34 and Section 8.3.10, "How to Use the Choice Component."

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the PublicSamples.zip file within the *jdev_install*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

The look and behavior of the Select Many Choice component on all supported devices is almost identical to the Select Many Checkbox component (see Section 8.3.9, "How to Use the Select Many Checkbox Component" for more information).

8.3.12 How to Use the Boolean Switch Component

The Boolean Switch (`selectBooleanSwitch`) component allows editing of boolean values as a switch metaphor instead of a checkbox.

Similar to other ADF Mobile AMX UI components, this component has a normal and selected state. To toggle the value, the end user taps (touches and releases) the switch once. Each tap toggles the switch.

Example 8–46 demonstrates the `selectBooleanSwitch` element defined in an ADF Mobile AMX file.

Example 8–46 Boolean Switch Definition

```
<amx:selectBooleanSwitch id="switch1"
                         label="Flip switch:"
                         onLabel="On"
                         offLabel="Off"
                         value="#{myBean.bool1}"/>
```

Figure 8–35 shows the Boolean Switch component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectBooleanSwitch id="selectBooleanSwitch1"
                         label="Switch"
                         value="value1">
```

Figure 8–35 Boolean Switch at Design Time

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.12.1 Support for Boolean Switch Components on iOS Platform

On iOS, Boolean Switch components are often used on Settings pages to enable or disable an attribute value.

8.3.12.2 Support for Boolean Switch Components on Android Platform

Android platform does not directly support a Boolean Switch component. Instead, Android provides a toggle button that allows tapping to switch between selected and deselected states.

8.3.13 How to Use the Select Button Component

The Select Button (`selectOneButton`) component represents a button group that lists actions, with a single button active at any given time. The selection mechanism is provided by the Select Items or Select Item component (see Section 8.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Select Button component.

> **Note:** The Select Button component can contain one Select Items component or one or more Select Item components.

Example 8–47 demonstrates the `selectOneButton` element defined in an ADF Mobile AMX file.

Example 8–47 Select Button Definition

```
<amx:selectOneButton id="bg1" value="#{myBean.myState}">
   <amx:selectItem label="Yes" value="yes"/>
   <amx:selectItem label="No" value="no"/>
   <amx:selectItem label="Maybe" value="maybe"/>
</amx:selectOneButton>
```

Figure 8–36 shows the Select Button component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectOneButton id="selectOneButton1"
                     label="Select Button"
                     value="value1">
   <amx:selectItem label="Value 1" value="value1"/>
   <amx:selectItem label="Value 2" value="value2"/>
   <amx:selectItem label="Value 3" value="value3"/>
</amx:selectOneButton>
```

Figure 8–36 Select Button at Design Time

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.14 How to Use the Radio Button Component

The Radio Button (`selectOneRadio`) component represents a group of radio buttons that lists available choices. The selection mechanism is provided by the Select Items or Select Item component (see Section 8.3.10.3, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Radio Button component.

> **Note:** The Radio Button component can contain one Select Items component or one or more Select Item components.

Example 8–48 and Example 8–49 demonstrate the `selectOneRadio` element definition in an ADF Mobile AMX file.

Example 8–48 Radio Button Definition Using Select Item Component

```
<amx:selectOneRadio id="radio1"
                    label="Choose a pet:"
                    value="#{myBean.myPet}">
   <amx:selectItem label="Cat" value="cat"/>
   <amx:selectItem label="Dog" value="dog"/>
   <amx:selectItem label="Hamster" value="hamster"/>
   <amx:selectItem label="Lizard" value="lizard"/>
</amx:selectOneRadio>
```

Example 8–49 Radio Button Definition Using Select Items Component

```
<amx:selectOneRadio id="radio1"
                    label="Choose a pet:"
                    value="#{myBean.myPet}">
   <amx:selectItems value="myBean.allPets"/>
</amx:selectOneRadio>
```

Figure 8–37 shows the Boolean Switch component displayed in the Preview pane. This component has its parameters set as follows:

```
<amx:selectOneRadio id="selectOneRadio1"
                     label="Radio Button"
                     value="value1">
   <amx:selectItem label="Value 1" value="value1"/>
   <amx:selectItem label="Value 2" value="value2"/>
   <amx:selectItem label="Value 3" value="value3"/>
</amx:selectOneRadio>
```

Figure 8–37 Radio Button at Design Time

For more information, illustrations, and examples, see the following:

- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.15 How to Use Carousel Component

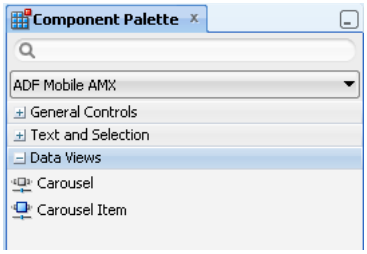
You use the Carousel (`carousel`) component to display other components, such as images, in a revolving carousel. The end user can change the active item by using either the slider or by dragging another image to the front. For more information, see the "Using the ADF Faces Carousel Component" section of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

The Carousel component contains a Carousel Item (`carouselItem`) component, whose text represented by the `text` attribute is displayed when it is the active item of the carousel. Although typically the Carousel Item contains an Image component, other components may be used. For example, you can use a Link as a child that surrounds an image.

> **Tip:** To minimize any negative effect on performance of your application, you should avoid using heavy-weight components as children: a complex structure creates a multiplied effect because several Carousel Items stamps are displayed simultaneously.

In JDeveloper, the Carousel is located under Data Views in the Component Palette (see Figure 8–38).

Figure 8–38 Carousel in Component Palette



Example 8–50 demonstrates the `carousel` element definition in an ADF Mobile AMX file. When defining the `carousel` element, you must place the `carouselItem` component inside of a `carousel` component's `nodeStamp` facet.

Example 8–50 Carousel Definition

```
<amx:carousel id="carousel1"
              value="#{bindings.products.collectionModel}"
              var="item"
              auxiliaryOffset="0.9"
              auxiliaryPopOut="hover"
              auxiliaryScale="0.8"
              controlArea="full"
              displayItems="circular"
              halign="center"
              valign="middle"
              disabled="false"
              shortDesc=""
              orientation="horizontal"
              styleClass="AMXStretchWidth"
              inlineStyle="height:250px;background-color:#EFEFEF;">
   <amx:facet name="nodeStamp">
      <amx:carouselItem id="item1" text="#{item.name}"
                        shortDesc="Product: #{item.name}">
         <amx:commandLink id="link1" action="goto-productDetails"
                          actionListener="#{someMethod()}">
            <amx:image id="image1" styleClass="prod-thumb"
                       source="images/img-big-#{item.uid}.png"/>
            <amx:setPropertyListener from="#{item}"
                                     to="#{pageFlowScope.product}"
                                     type="action"/>
         </amx:commandLink>
      </amx:carouselItem>
   </amx:facet>
</amx:carousel>
```

Figure 8–39 shows a Carousel component displayed on an iPhone.

Figure 8–39 Carousel Component on iPhone

For more information, illustrations, and examples, see the following:

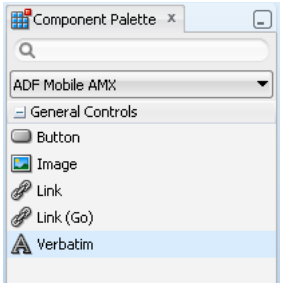
- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*
- CompGallery, an ADF Mobile sample application located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

8.3.16 How to Use Verbatim Component

You use the Verbatim (`verbatim`) operation to insert your own HTML into a page in cases where such a component does not exist or you prefer laying it out yourself using HTML.

In JDeveloper, Verbatim is located under General Controls in the Component Palette (see Figure 8–41).

Figure 8–40 Verbatim in Component Palette



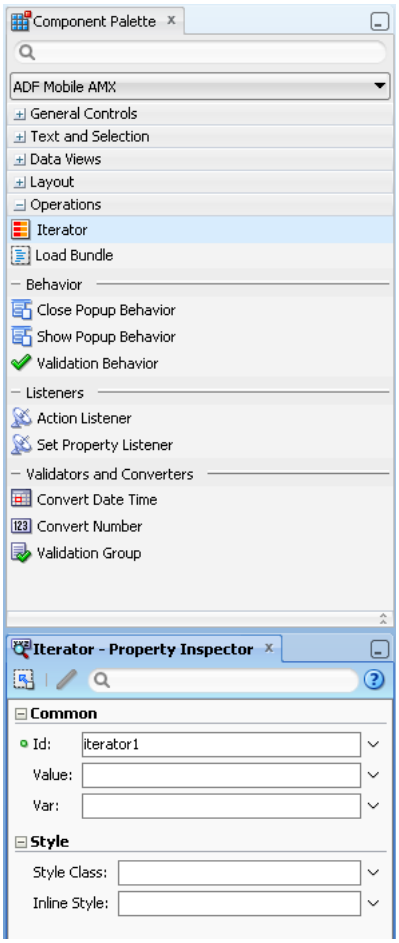
For more information and examples, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.3.17 How to Enable Iteration

You use the Iterator (`iterator`) operation to stamp an arbitrary number of items with the same kind of data, which allows you to iterate through the data and produce UI for each element.

In JDeveloper, the Iterator is located under Operations in the Component Palette (see Figure 8–41).

Figure 8–41 Iterator in Component Palette



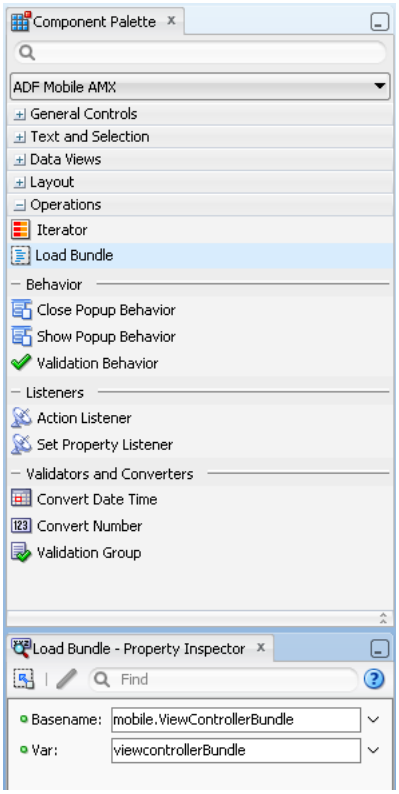
For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.3.18 How to Load a Resource Bundle

The Load Bundle (`loadBundle`) operation allows you to specify the resource bundle that provides localized text for the ADF Mobile AMX UI components on a page. For more information, see Section 8.7, "Localizing UI Components."

In JDeveloper, the Load Bundle is located under Operations in the Component Palette (see Figure 8–42).

Figure 8–42 Load Bundle in Component Palette



In your ADF Mobile AMX file, you declare the `loadBundle` element as a child of the `view` element.

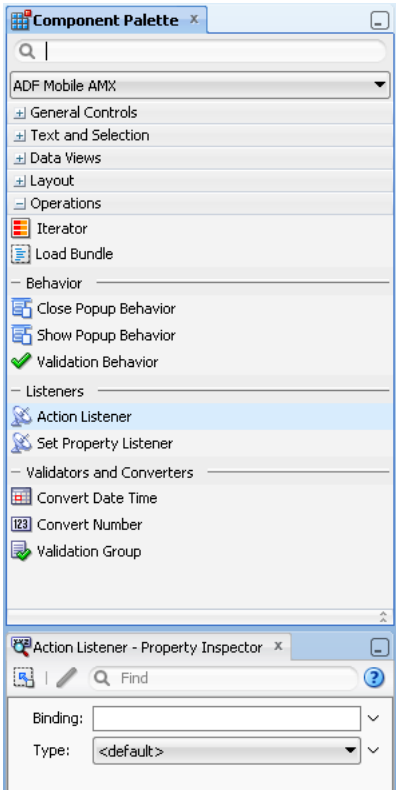
For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.3.19 How to Use the Action Listener

The Action Listener (`actionListener`) component allows you to declaratively invoke commands through EL based on the type of the parent component's usage.

In JDeveloper, the Action Listener component is located under Listeners in the Component Palette (see Figure 8–43).

Figure 8–43 Action Listener in Component Palette



You can add zero or more Action Listener or Set Property Listener components as children of any command component (Button, Link, List Item). Listener attributes, such as `actionListener`, `valueChangeListener`, and `moveListener`, are defined for various components. The `type` attribute of the Action Listener or Set Property Listener component can denote which event this listener component is to handle and is represented by the first portion of the parent's listener attribute name: `action`, `valueChange`, `move`. Alternatively, the `type` attribute can also represent a gesture, such as `swipeLeft`, `swipeRight`, `tapHold`, and so on.

For more information, see the following:

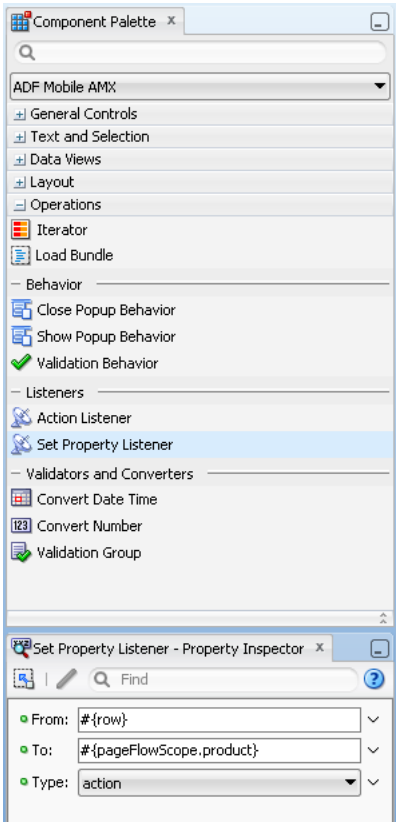
- Section 8.10, "Using Event Listeners"
- Section 8.3.20, "How to Use the Set Property Listener"
- Section 8.4, "Enabling Gestures"
- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*

8.3.20 How to Use the Set Property Listener

The Set Property Listener (`setPropertyListener`) component allows you to declaratively move variable values from one location (defined by the component's `from` attribute) to another (defined by the component's `to` attribute) without having to write code.

In JDeveloper, the Set Property Listener component is located under Listeners in the Component Palette (see Figure 8–44).

Figure 8–44 Set Property Listener in Component Palette



Example 8–51 demonstrates the `setPropertyListener` element definition in an ADF Mobile AMX file.

Example 8–51 Set Property Listener Definition

```
<amx:listView value="#{bindings.products.collectionModel}" var="row" id="lv1">
   <amx:listItem action="details">
      <amx:outputText value="#{row.name}" id="ot1" />
      <amx:setPropertyListener from="#{row}"
                               to="#{pageFlowScope.product}"
                               type="action"/>
   </amx:listItem>
</amx:listView>
```

You can add zero or more Set Property Listener or Action Listener components as children of any command component (Button, Link, List Item). Listener attributes, such as `actionListener`, `valueChangeListener`, and `moveListener`, are defined for various components. The `type` attribute of the Set Property Listener or Action Listener component can denote which event this listener component is to handle and is represented by the first portion of the parent's listener attribute name: `action`, `valueChange`, `move`. Alternatively, the `type` attribute can also represent a gesture, such as `swipeLeft`, `swipeRight`, `tapHold`, and so on.

For more information, see the following:

- Section 8.10, "Using Event Listeners"
- Section 8.3.19, "How to Use the Action Listener"
- Section 8.4, "Enabling Gestures"

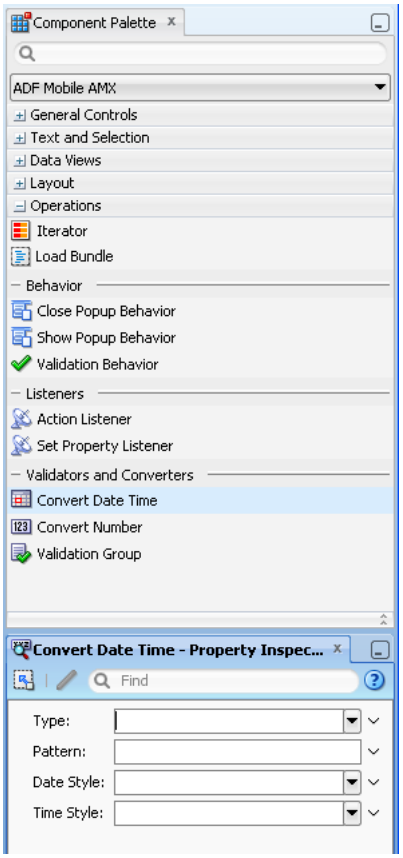
- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*

8.3.21 How to Convert Date and Time Values

The Convert Date Time (`convertDateTime`) is not an independent UI component: it is a converter operation that you use in conjunction with an Output Text and Input Text component to display converted date, time, or a combination of date and time in a variety of formats following the specified pattern.

In JDeveloper, the Convert Date Time is located under Validators and Converters in the Component Palette (see Figure 8–45).

Figure 8–45 Convert Date Time in Component Palette



Example 8–52 demonstrates the `convertDateTime` element declared in an ADF Mobile AMX file.

Example 8–52 Using Convert Date Time

```
<amx:panelPage id="pp1">
   <amx:inputText styleClass="ui-text" value="Order Date" id="it1" >
      <amx:convertDateTime type="both"/>
   </amx:inputText>
</amx:panelPage>
```

To convert date and time values:

1. From the **Components** palette, drag a **Convert Date Time** component and insert it within an Output Text or Input Text component, making it a child element of that component.

2. Open the **Property** editor for the Convert Date Time component and define its attributes. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

> **Note:** The Convert Date Time component does not produce any effect at design time.

The Convert Date Time component allows a level of leniency while converting an input value string to date:

- A converter with associated pattern `MMM` for month, when attached to any value holder, accepts values with month specified in the form `MM` or `M` as valid.
- Allows use of such separators as dash (-) or period (.) or slash (/), irrespective of the separator specified in the associated pattern.
- Leniency in pattern definition set by the `pattern` attribute.

For example, when a pattern on the converter is set to `"MMM/d/yyyy"`, the following inputs are accepted as valid by the converter:

```
Jan/4/2004
Jan-4-2004
Jan.4.2004
01/4/2004
01-4-2004
01.4.2004
1/4/2004
1-4-2004
1.4.2004
```

The converter supports the same parsing and formatting conventions as the `java.text.SimpleDateFormat` (specified using the `dateStyle`, `timeStyle`, and `pattern` attributes), except the case when the time zone is specified to have a long display, such as `timeStyle=full` or a pattern set with `zzzz`. Instead of a long descriptive string, such as "Pacific Standard Time", the time zone is displayed in the General Timezone format (GMT +/- offset) or RFC-822 time zones.

The exact result of the conversion depends on the locale, but typically the following rules apply:

- `SHORT` is completely numeric, such as 12.13.52 or 3:30pm
- `MEDIUM` is longer, such as Jan 12, 1952
- `LONG` is longer, such as January 12, 1952 or 3:30:32pm
- `FULL` is completely specified, such as Tuesday, April 12, 1952 AD or 3:30:42pm PST

8.3.21.1 What You May Need to Know About Date and Time Patterns

As per `java.text.SimpleDateFormat` definition, date and time formats are specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from `A` to `Z` and from `a` to `z` are interpreted as pattern letters representing the components of a date or time string. Text can be quoted using single quotes (`'`) to avoid interpretation. `" ' ' "` represents a single quote. All other characters are not interpreted; instead, they are simply copied into the output string during formatting, or matched against the input string during parsing.

Table 8–7 lists the defined pattern letters (all other characters from `A` to `Z` and from `a` to `z` are reserved).

Table 8–7 Date and Time Pattern Letters

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	▪ AD
y	Year	Year	▪ 1996 ▪ 96
M	Month in year	Month	▪ July ▪ Jul ▪ 07
w	Week in year	Number	▪ 27
W	Week in month	Number	▪ 2
D	Day in year	Number	▪ 189
d	Day in month	Number	▪ 10
F	Day of week in month	Number	▪ 2
E	Day in week	Text	▪ Tuesday ▪ Tue
a	Am/pm marker	Text	▪ PM
H	Hour in day (0-23)	Number	▪ 0
k	Hour in day (1-24)	Number	▪ 24
K	Hour in am/pm (0-11)	Number	▪ 0
h	Hour in am/pm (1-12)	Number	▪ 12
m	Minute in hour	Number	▪ 30
s	Second in minute	Number	▪ 55
S	Millisecond	Number	▪ 978
z	Time zone	General time zone	▪ Pacific Standard Time ▪ PST ▪ GMT-08:00
Z	Time zone	RFC 822 time zone	▪ -0800

Pattern letters are usually repeated, as their number determines the exact presentation.

8.3.22 How to Convert Numerical Values

The Convert Number (`convertNumber`) is not an independent UI component: it is a converter operation that you use in conjunction with an Output Text or Input Text component to display converted number or currency figures in a variety of formats following a specified pattern.

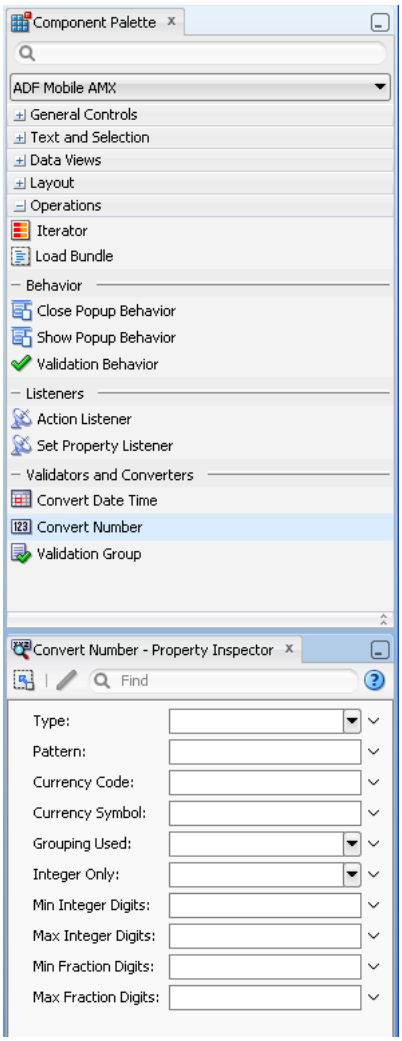
The Convert Number component provides the following types of conversion:

- From value to string, for display purposes.
- From formatted string to value, when formatted input value is parsed into its underlying value.

When the Convert Number is specified as a child of an Input Text component, the numeric keyboard is displayed on a mobile device by default.

In JDeveloper, the Convert Number is located under Validators and Converters in the Component Palette (see Figure 8–46).

Figure 8–46 Convert Number in Component Palette



Example 8–53 demonstrates the `convertNumber` element defined in an ADF Mobile AMX file.

Example 8–53 Using Convert Number

```
<amx:panelPage id="pp1">
   <amx:inputText styleClass="ui-text" value="Product Price" id="it1" >
      <amx:convertNumber type="percent" groupingUsed="false" integerOnly="true"/>
   </amx:inputText>
</amx:panelPage>
```

To convert numerical values:

1. From the **Components** palette, drag a **Convert Number** component and insert it within an Output Text or Input Text component, making it a child element of that component.

2. Open the **Property** editor for the Convert Number component and define its attributes. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

> **Note:** The Convert Number component does not produce any effect at design time.

8.4 Enabling Gestures

You can configure Button, Link, and List Item components to react to the following gestures:

- Swipe to the right
- Swipe to the left
- Swipe up
- Swipe down
- Tap-and-hold

You can define `swipeRight`, `swipeLeft`, `swipeUp`, `swipeDown`, and `tapHold` values for the `type` attribute of the following operations:

- Set Property Listener (see Section 8.3.20, "How to Use the Set Property Listener")
- Action Listener (see Section 8.3.19, "How to Use the Action Listener")
- Show Popup Behavior (see Section 8.2.8, "How to Use a Popup Component")
- Close Popup Behavior (see Section 8.2.8, "How to Use a Popup Component")

The values of the `type` attribute are restricted based on the parent component and are supported only for Button, Link, and List Item components.

Example 8–54 demonstrates use of the `tapHold` value of the `type` attribute in an ADF Mobile AMX file. In this example, the tap-and-hold gesture triggers the display of a Popup component.

Example 8–54 Using Tap-and-Hold Gesture

```
<amx:panelPage id="pp1">
   <amx:listView id="lv1"
                 value="#{bindings.data.collectionModel}"
                 var="row">
      <amx:listItem action="gosomewhere">
         <amx:outputText id="ot1" value="#{row.description}"/>
         <amx:setPropertyListener from="#{row.rowKey}"
                                  to="#{mybean.currentRow}"
                                  type="tapHold"/>
         <amx:showPopupBehavior type="tapHold"
                                alignid="pp1"
                                popupid="pop1"
                                align="startAfter"/>
      </amx:listItem>
   </amx:listView>>
</amx:panelPage>
<amx:popup id="pop1">
   <amx:panelGroupLayout id="pgl1" layout="horizontal">
      <amx:commandLink id="cm1" actionListener="#{mybean.doX}">
         <amx:image id="i1" source="images/x.png"/>
```

```
         <amx:closePopupBehavior type="action" popupid="pop1"/>
      </amx:commandLink>
      <amx:commandLink id="cm2" actionListener="#{mybean.doY}">
         <amx:image id="i2" source="images/y.png"/>
         <amx:closePopupBehavior type="action" popupid="pop1"/>
      </amx:commandLink>
      <amx:commandLink id="cm3" actionListener="#{mybean.doZ}">
         <amx:image id="i3" source="images/y.png"/>
         <amx:closePopupBehavior type="action" popupid="pop1"/>
      </amx:commandLink>
   </amx:panelGroupLayout>
</amx:popup>
```

Example 8–55 demonstrates use of the `swipeRight` gesture in an ADF Mobile AMX file.

Example 8–55 Using Swipe Right Gesture

```
<amx:panelPage id="pp1">
   <amx:listView id="lv1"
                 value="#{bindings.data.collectionModel}"
                 var="row">
      <amx:listItem action="gosomewhere">
         <amx:outputText id="ot1" value="#{row.description}"/>
         <amx:setPropertyListener from="#{row.rowKey}"
                                  to="#{mybean.currentRow}"
                                  type="swipeRight"/>
         <actionListener binding="#{mybean.DoX}" type="swipeRight"/>
      </amx:listItem>
   </amx:listView>>
</amx:panelPage>
```

For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

An ADF Mobile sample application called GestureDemo demonstrates how to use gestures with a variety of ADF Mobile AMX UI components. This sample application is located in the `PublicSamples.zip file` within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer.

8.5 Providing Data Visualization

ADF Mobile employs a set of Data Visualization Tools that you can use to create various charts, gauges, and maps to represent data in your ADF Mobile AMX feature. You can declare the following elements under the `<dvtm>` namespace in an ADF Mobile AMX file:

- `areaChart` (see Section 8.5.1, "How to Create an Area Chart")
- `barChart` (see Section 8.5.2, "How to Create a Bar Chart")
- `horizontalBarChart` (see Section 8.5.3, "How to Create a Horizontal Bar Chart")
- `bubbleChart` (see Section 8.5.4, "How to Create a Bubble Chart")
- `comboChart` (see Section 8.5.5, "How to Create a Combo Chart")
- `lineChart` (see Section 8.5.6, "How to Create a Line Chart")
- `pieChart` (see Section 8.5.7, "How to Create a Pie Chart")
- `scatterChart` (see Section 8.5.8, "How to Create a Scatter Chart")

- `sparkChart` (see Section 8.5.9, "How to Create a Spark Chart")
- `ledGauge` (see Section 8.5.10, "How to Create a LED Gauge")
- `statusMeterGauge` (see Section 8.5.11, "How to Create a Status Meter Gauge")
- `dialGauge` (see Section 8.5.12, "How to Create a Dial Gauge")
- `geographicMap` (see Section 8.5.14, "How to Create a Geographic Map Component")
- `thematicMap` (see Section 8.5.15, "How to Create a Thematic Map Component")

Chart, gauge, and map elements have a number of attributes that are common to all or most of them. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

In JDeveloper, the chart components are located under **DVT Mobile AMX > Chart** in the Component Palette (see Figure 8–47), gauge components are located under **DVT Mobile AMX > Gauge** (see Figure 8–48), and map components are located under **DVT Mobile AMX > Map** (see Figure 8–49).

Figure 8–47 Charts in the Component Palette

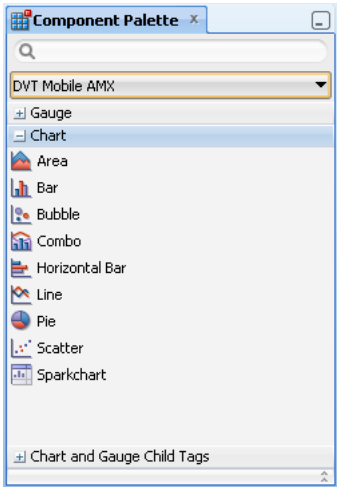


Figure 8–48 Gauge Components in the Component Palette

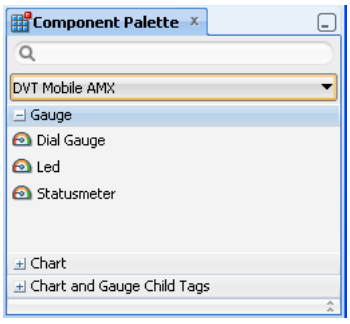
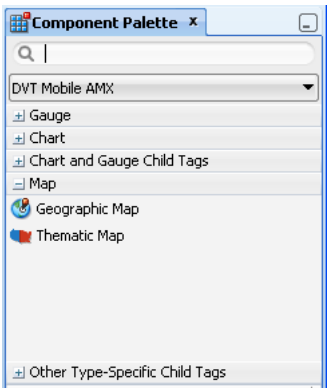


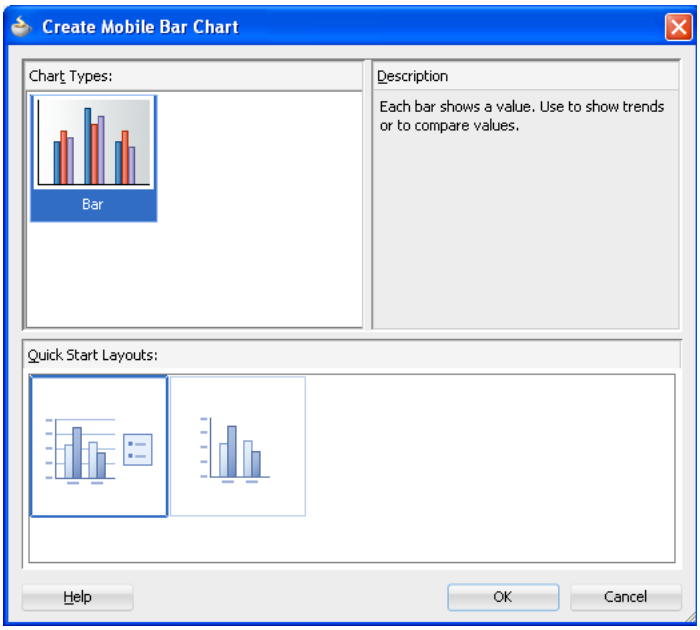
Figure 8–49 Map Components in the Component Palette



When you drag and drop a chart, gauge, or map, one of the following dialogs opens to display the information about the type of component you are creating:

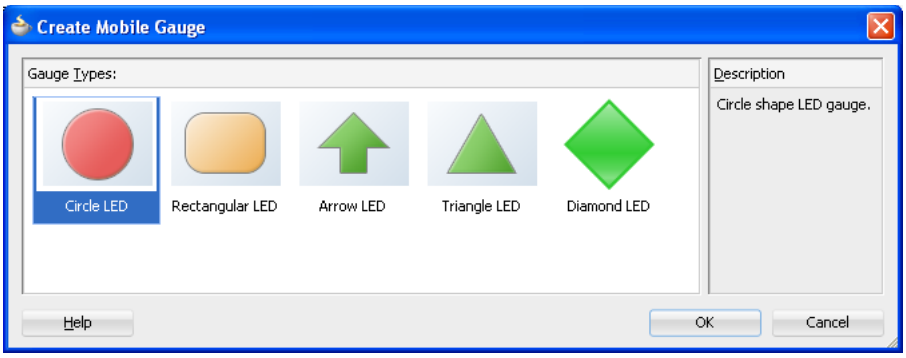
- **Create Mobile Chart** (see Figure 8–50)

Figure 8–50 Creating Chart Components



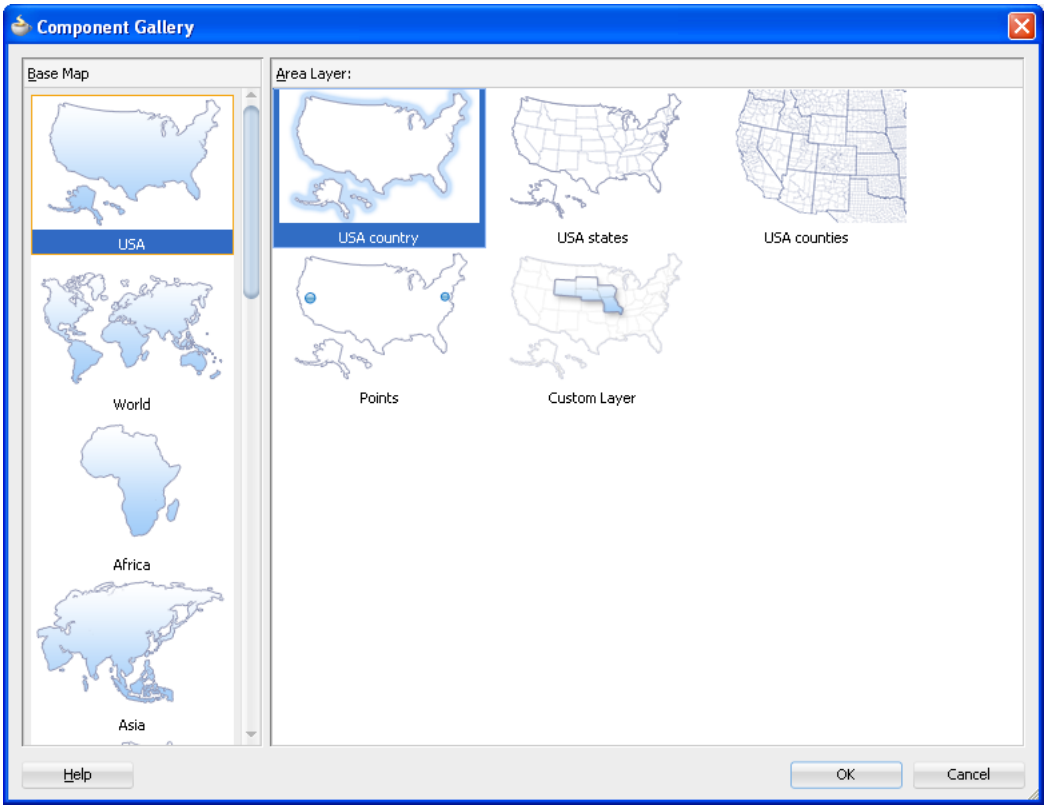
- **Create Mobile Gauge** (see Figure 8–51)

Figure 8–51 Creating Gauge Components



- **Component Gallery** (see Figure 8–52)

Figure 8–52 Creating Map Components



You can add event listeners to data visualization components. For more information, see Section 8.10, "Using Event Listeners." Event listeners are applicable to components for the ADF Mobile AMX run-time description on both iOS and Android-powered devices, but the listeners do not have any effect at design time.

For information on databound data visualization components that are created from the Data Controls panel, see Section 8.5.16, "How to Create Databound Data Visualization Components."

An ADF Mobile sample application called CompGallery demonstrates how to use various data visualization components in your ADF Mobile AMX application feature. This sample application is located in the `PublicSamples.zip` file within the *`jdev_`*

install/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples directory on your development computer.

For information on limitations to the usage of ADF Mobile AMX data visualization components, see Section C.5.2, "Data Visualization Components Limitations."

For information on Oracle ADF data visualization components, see the "Using ADF Data Visualization Components" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

8.5.1 How to Create an Area Chart

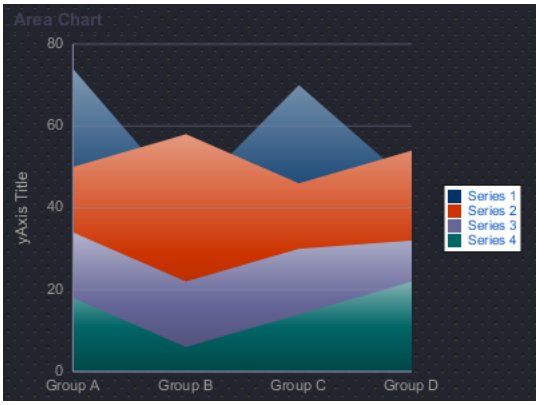
You use the Area Chart (areaChart) to visually represent data where sets of data items are related and categorized into groups and series. The series are visualized using graphical elements with some common style properties (such as, for example, an area color or pattern). Those properties have to be applied at the series level instead of per each individual data item. You have an option to use the default or custom series styles. For information about defining custom series styles, see Section 8.5.6, "How to Create a Line Chart."

Example 8–56 shows the areaChart element defined in an ADF Mobile AMX file. To create a basic area chart with default series style, you pass it a collection and specify the dataStamp facet with a nested chartDataItem element.

Example 8–56 Area Chart Definition with Default Series Styles

```
<dvtm:areaChart id="areaChart1"
                value="#{bindings.lineData.collectionModel}"
                var="row"
                inlineStyle="width: 400px; height: 300px;"
                title="Chart Title"
                animationOnDisplay="auto"
                animationDuration="1500" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
   </amx:facet>
   <dvtm:yAxis axisMaxValue="80.0" majorIncrement="20.0" title="yAxis Title" />
   <dvtm:legend position="end" />
</dvtm:areaChart>
```

Figure 8–53 Area Chart at Design Time



Data items are initialized in the collection model and equipped with the stamping mechanism. At a minimum, each collection `row` must include the following properties:

- `series`: name of the series to which this data item belongs;
- `group`: name of the group to which this data item belongs;
- `value`: the data item value.

The collection `row` might also include other properties, such as `color` or `shape`, applicable to individual data items.

You can use attribute groups (`attributeGroups` element) to set style properties for a group of data items based on some grouping criteria, as Figure 8–57 shows. In this case, the data item `color` and `shape` attributes are set based on the additional grouping expression.

Example 8–57 Area Chart Definition with Default Series Styles and Grouping

```
<dvtm:areaChart id="areaChart1"
                value="#{bindings.lineData.collectionModel}"
                var="row"
                inlineStyle="width: 400px; height: 300px;"
                title="Chart Title"
                animationOnDisplay="auto"
                animationDuration="1500" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
         <dvtm:attributeGroups id="ag_1"
                               type="color shape"
                               value="#{row.brand}" />
   </amx:facet>
   <dvtm:yAxis axisMaxValue="80.0" majorIncrement="20.0" title="yAxis Title" />
   <dvtm:legend position="end" />
</dvtm:areaChart>
```

Note: In Example 8–56 and Figure 8–57, since custom styles are not set at the series level, series are displayed with the colors based on the default color ramp.

For information on attributes of the `areaChart` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the `areaChart`:

- `chartDataItem` (see Section 8.5.13.1, "Defining Chart Data Item")
- `xAxis`, `yAxis`, and `y2Axis` (see Section 8.5.13.3, "Defining X Axis, YAxis, and Y2Axis")
- `legend` (see Section 8.5.13.2, "Defining Legend")

8.5.2 How to Create a Bar Chart

You use a bar chart (`barChart`) to visually display data as vertical bars, where sets of data items are related and categorized into groups and series. The series are visualized

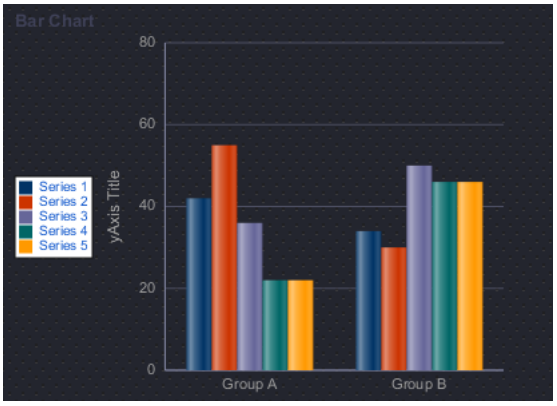
using graphical elements with some common style properties that you have to apply at the series level instead of per each individual data item.

Example 8–58 shows the `barChart` element defined in an ADF Mobile AMX file. The `dataStamp` facet is specified with a nested `chartDataItem` element.

Example 8–58 Bar Chart Definition

```
<dvtm:barChart id="barChart1"
               value="#{bindings.barData.collectionModel}"
               var="row"
               inlineStyle="width: 400px; height: 300px;"
               title="Bar Chart"
               animationOnDisplay="zoom"
               animationDuration="3000" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
   </amx:facet>
   <dvtm:yAxis axisMaxValue="80.0" majorIncrement="20.0" title="yAxis Title" />
   <dvtm:legend position="start" />
</dvtm:barChart>
```

Figure 8–54 Bar Chart at Design Time



The data model for a bar chart is represented by a collection of items (rows) that describe individual bars. Typically, properties of each bar include the following:

- `series`: name of the series to which this bar belongs;
- `group`: name of the group to which this bar belongs;
- `value`: the data item value (required).

Data must include the same number of groups per series. If any of the series or data pairs are missing, it is passed to the API as `null`.

For information on attributes of the `barChart` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the `barChart`:

- `chartDataItem` (see Section 8.5.13.1, "Defining Chart Data Item")
- `xAxis`, `yAxis`, and `y2Axis` (see Section 8.5.13.3, "Defining X Axis, YAxis, and Y2Axis")

- `legend` (see Section 8.5.13.2, "Defining Legend")

8.5.3 How to Create a Horizontal Bar Chart

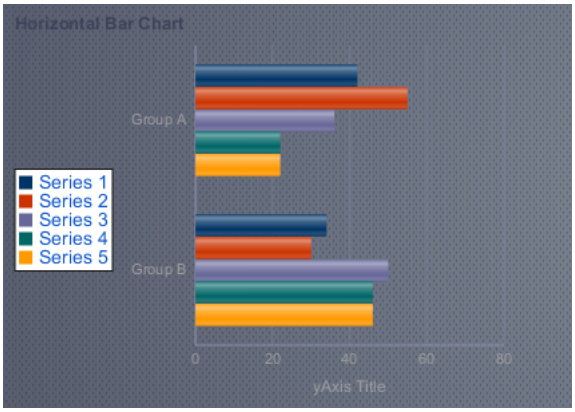
You use a horizontal bar chart (`barChart`) to visually display data as horizontal bars, where sets of data items are related and categorized into groups and series. The series are visualized using graphical elements with some common style properties that you have to apply at the series level instead of per each individual data item.

Example 8–58 shows the `horizontalBarChart` element defined in an ADF Mobile AMX file. The `dataStamp` facet is specified with a nested `chartDataItem` element.

Example 8–59 Horizontal Bar Chart Definition

```
<dvtm:horizontalBarChart id="horizBarChart1"
                  value="#{bindings.barData.collectionModel}"
                  var="row"
                  inlineStyle="width: 400px; height: 300px;"
                  title="Horizontal Bar Chart"
                  dataSelection="#{pageFlowScope.dataSelection}"
                  hideAndShowBehavior="#{pageFlowScope.hideAndShowBehavior}"
                  rolloverBehavior="#{pageFlowScope.rolloverBehavior}"
                  stack="#{pageFlowScope.stack}" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
   </amx:facet>
   <dvtm:yAxis axisMaxValue="80.0" majorIncrement="20.0" title="yAxis Title" />
   <dvtm:legend position="start" />
</dvtm:horizontalBarChart>
```

Figure 8–55 Horizontal Bar Chart at Design Time



The data model for a horizontal bar chart is represented by a collection of items (rows) that describe individual bars. Typically, properties of each bar include the following:

- `series`: name of the series to which this bar belongs;
- `group`: name of the group to which this bar belongs;
- `value`: the data item value (required).

Data must include the same number of groups per series. If any of the series or data pairs are missing, it is passed to the API as `null`.

For information on attributes of the `horizontalBarChart` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the `horizontalBarChart`:

- `chartDataItem` (see Section 8.5.13.1, "Defining Chart Data Item")
- `xAxis`, `yAxis`, and `y2Axis` (see Section 8.5.13.3, "Defining X Axis, YAxis, and Y2Axis")
- `legend` (see Section 8.5.13.2, "Defining Legend")

8.5.4 How to Create a Bubble Chart

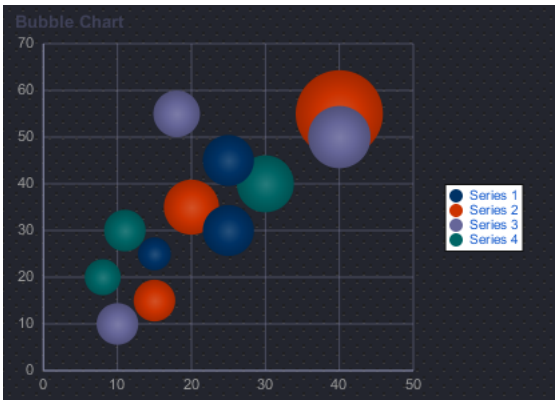
A bubble chart (`bubbleChart`) displays a set of data items where each data item has `x`, `y` coordinates and size (bubble). In addition, each data item can have various style attributes, such as `color` and `shape`. You can either set properties of each data item individually, or categorize the data items into groups based on various criteria. You may use multiple grouping criteria at the same time, and may also use different style attributes to visualize the data items relationships. However, unlike line charts (see Section 8.5.6, "How to Create a Line Chart") or area charts (see Section 8.5.1, "How to Create an Area Chart"), bubble charts do not have a strict notion of the series and groups.

Example 8–60 shows the `bubbleChart` element defined in an ADF Mobile AMX file. The `dataStamp` facet is specified with a nested `chartDataItem` element. The `color` and `shape` attributes of each data item are set individually based on the values supplied in the data model:

Example 8–60 Bubble Chart Definition with Custom Data Item Properties

```
<dvtm:bubbleChart id="bubbleChart1"
                  value="#{bindings.bubbleData.collectionModel}"
                  inlineStyle="width: 400px; height: 300px;"
                  dataSelection="multiple"
                  rolloverBehavior="dim"
                  animationOnDisplay="auto"
                  title="Bubble Chart"
                  var="row">
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem label="#{row.label}" x="#{row.x}" y="#{row.y}"
                          size="#{row.size}" color="#{row.color}"
                          shape="#{row.shape}" />
   </amx:facet>
</dvtm:bubbleChart>
```

Figure 8–56 Bubble Chart at Design Time



In Example 8–61, the `attributeGroups` element is used to set common style attributes for a related group of data items.

Example 8–61 Bubble Chart Definition with Attribute Groups

```
<dvtm:bubbleChart id="bubbleChart1"
                  value="#{bindings.bubbleData.collectionModel}"
                  dataSelection="multiple"
                  rolloverBehavior="dim"
                  animationOnDisplay="auto"
                  title="Bubble Chart"
                  var="row">
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem label="#{row.label}" x="#{row.x}" y="#{row.y}" >
         <dvtm:attributeGroups id="ag1" type="color" value="#{row.category}" />
         <dvtm:attributeGroups id="ag2" type="shape" value="#{row.brand}" />
      </dvtm:chartDataItem>
   </amx:facet>
</dvtm:bubbleChart>
```

Example 8–62 shows a Bubble Chart with all the possible child elements (`chartDataItem`, `xAxis`, `yAxis`, and `legend`) defined.

Example 8–62 Bubble Chart Definition with All Attributes

```
<dvtm:bubbleChart id="bubbleChart1"
                  inlineStyle="width: 400px; height: 300px;"
                  value="#{bindings.bubbleData.collectionModel}"
                  var="row"
                  title="Chart Title"
                  animationOnDisplay="zoom"
                  animationDuration="3000" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem label="#{row.group}" x="#{row.data.x}"
                          y="#{row.data.y}" size="#{row.data.z}" >
         <dvtm:attributeGroups type="color" value="#{row.series}" />
         <dvtm:attributeGroups type="shape" value="#{row.group}" />
      </dvtm:chartDataItem>
   </amx:facet>
```

The data model for a bubble chart is represented by a collection of items (rows) that describe individual data items. Typically, properties of each bar include the following:

- `label`: data item label (optional);

- x, y: value coordinates (required);
- z: the size of data item (required).

The data must include the same number of groups per series. If any of the series or data pairs are missing, it is passed to the API as null.

For information on attributes of the bubbleChart element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the barChart:

- chartDataItem (see Section 8.5.13.1, "Defining Chart Data Item")
- xAxis, yAxis, and y2Axis (see Section 8.5.13.3, "Defining X Axis, YAxis, and Y2Axis")
- legend (see Section 8.5.13.2, "Defining Legend")

8.5.5 How to Create a Combo Chart

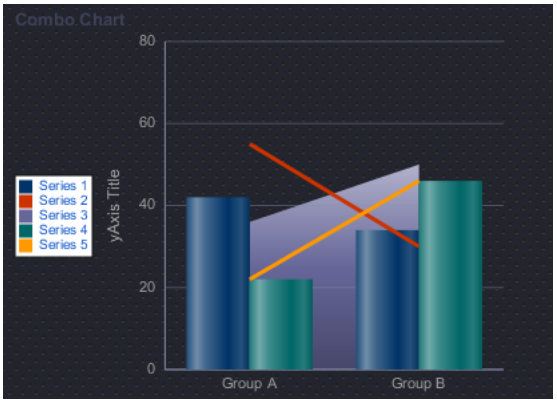
A combo chart (comboChart) represents an overlay of two or more different charts, such as a line and bar chart.

Example 8–63 shows the comboChart element defined in an ADF Mobile AMX file. The dataStamp facet is specified with a nested chartDataItem element. The seriesStamp facet overrides the default style properties for the series and sets custom series styles using the seriesStyle elements.

Example 8–63 Combo Chart Definition

```
<dvtm:comboChart id="comboChart1"
                 value="#{bindings.barData.collectionModel}"
                 var="row"
                 seriesVar="series"
                 inlineStyle="width: 400px; height: 300px;"
                 title="Combo Chart"
                 animationOnDisplay="auto"
                 animationDuration="1500" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
   </amx:facet>
   <amx:facet name="seriesStamp">
      <dvtm:seriesStyle series="#{series.name}" type="bar"
                        rendered="#{(series.name eq 'Series 1') or
                                    (series.name eq 'Series 2') or
                                    (series.name eq 'Series 3')}" />
      <dvtm:seriesStyle series="#{series.name}" type="line" lineWidth="5"
                        rendered="#{(series.name eq 'Series 4') or
                                    (series.name eq 'Series 5')}" />
   </amx:facet>
   <dvtm:yAxis axisMaxValue="80.0" majorIncrement="20.0" title="yAxis Title" />
   <dvtm:legend position="start" />
</dvtm:comboChart>
```

Figure 8–57 Combo Chart at Design Time



For information on attributes of the `comboChart` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the `comboChart`:

- `chartDataItem` (see Section 8.5.13.1, "Defining Chart Data Item")
- `xAxis`, `yAxis`, and `y2Axis` (see Section 8.5.13.3, "Defining X Axis, YAxis, and Y2Axis")
- `legend` (see Section 8.5.13.2, "Defining Legend")

8.5.6 How to Create a Line Chart

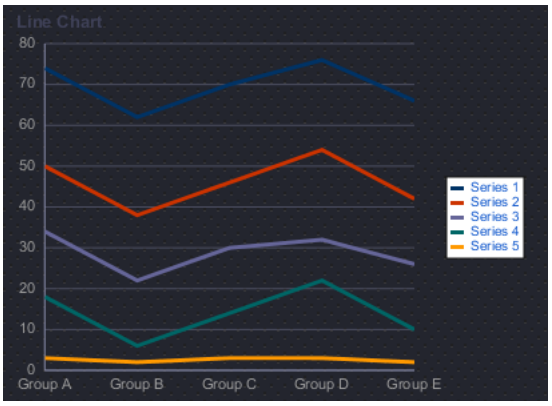
You use the line chart (`lineChart`) to visually represent data where sets of data items are related and categorized into groups and series. The series are visualized using graphical elements with some common style properties (such as, for example, a line color, width, or style). Those properties have to be applied at the series level instead of per each individual data item. You have an option to use the default or custom series styles.

Example 8–64 shows the `lineChart` element defined in an ADF Mobile AMX file. To create a basic line chart with default series style, you pass it a collection and specify the `dataStamp` facet with a nested `chartDataItem` element.

Example 8–64 Line Chart Definition with Default Series Styles

```
<dvtm:lineChart id="lineChart1"
                inlineStyle="width: 400px; height: 300px;"
                rolloverBehavior="dim"
                animationOnDisplay="auto"
                title="Line Chart"
                value="#{bindings.lineData1.collectionModel}"
                var="row" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}"
                          color="#{row.color}" />
   </amx:facet>
</dvtm:lineChart>
```

Figure 8–58 Line Chart at Design Time



Data items are initialized in the collection model and equipped with the stamping mechanism. At a minimum, each collection `row` must include the following properties:

- `series`: name of the series to which this line belongs;
- `group`: name of the group to which this line belongs;
- `value`: the data item value.

The collection `row` might also include other properties, such as `color` or `shape`, applicable to individual data items.

You can use attribute groups (`attributeGroups` element) to set style properties for a group of data items based on some grouping criteria, as Figure 8–65 shows. In this case, the data item `color` and `shape` attributes are set based on the additional grouping expression.

Example 8–65 Line Chart Definition with Default Series Styles and Grouping

```
<dvtm:lineChart id="lineChart1"
                inlineStyle="width: 400px; height: 300px;"
                rolloverBehavior="dim"
                animationOnDisplay="auto"
                title="Line Chart"
                titleStyle="font-size: 12px; color: black;"
                value="#{bindings.lineData1.collectionModel}"
                var="row" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
         <dvtm:attributeGroups id="ag_1"
                               type="color shape"
                               value="#{row.brand}" />
      </dvtm:chartDataItem>
   </amx:facet>
</dvtm:lineChart>
```

Note: In Example 8–64 and Figure 8–65, since custom styles are not set at the series level, series are displayed with the colors based on the default color ramp.

To override the default style properties for the series, you can define an optional `seriesStamp` facet and set custom series styles using the `seriesStyle` elements, as Figure 8–66 shows.

Example 8–66 Line Chart Definition with Custom Series Styles

```
<dvtm:lineChart id="lineChart1"
                inlineStyle="width: 400px; height: 300px;"
                rolloverBehavior="dim"
                animationOnDisplay="auto"
                title="Line Chart"
                value="#{bindings.lineData1.collectionModel}"
                var="row"
                seriesVar="series" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}" />
      </dvtm:chartDataItem>
   <amx:facet name="seriesStamp">
      <dvtm:seriesStyle series="#{series.name}"
                        lineStyle="#{series.lineStyle}"
                        lineWidth="#{series.lineWidth}" />
   </amx:facet>
</dvtm:lineChart>
```

In the preceding example, the `seriesVar` attribute is added to the main `lineChart` element. This attribute defines the name of the variable to be used when processing the `seriesStyle` elements in the `seriesStamp` facet. The `seriesVar` refers to items of an internal collection created and populated automatically while processing the `dataStamp` facet. At run time, the `chartDataItem` elements contained in the `dataStamp` facet are processed first. For each unique `series` attribute value the wrapper creates a new JavaScript object instance representing the common series properties. At a minimum, the instance includes the `name` property. In addition, the object includes properties from the original data collection that have the same values for all rows with the same series name. Figure 8–67 shows a sample collection of data that, upon processing of the `dataStamp` facet, results in the internal series collection that Figure 8–68 shows.

Example 8–67 Data Collection for Line Chart

```
row[0]: { series: 'A', value: 81, lineColor: 'red', lineStyle: 'solid' }
row[1]: { series: 'A', value: 26, lineColor: 'red', lineStyle: 'solid' }
row[2]: { series: 'A', value: 53, lineColor: 'red', lineStyle: 'solid' }
row[3]: { series: 'B', value: 10, lineColor: 'blue', lineStyle: 'solid' }
row[4]: { series: 'B', value: 44, lineColor: 'blue', lineStyle: 'solid' }
row[5]: { series: 'B', value: 35, lineColor: 'blue', lineStyle: 'solid' }
row[6]: { series: 'C', value: 83, lineColor: 'green', lineStyle: 'dot' }
row[7]: { series: 'C', value: 23, lineColor: 'green', lineStyle: 'dot' }
row[8]: { series: 'C', value: 54, lineColor: 'green', lineStyle: 'dot' }
```

Example 8–68 Processed Data Collection for Line Chart

```
series[0]: { name: 'A', lineColor: 'red', lineStyle: 'solid' }
series[1]: { name: 'B', lineColor: 'blue', lineStyle: 'solid' }
series[2]: { name: 'C', lineColor: 'green', lineStyle: 'dot' }
```

Alternatively, you can control the series styles in an ADF Mobile AMX page design using the `rendered` attribute of the `seriesStyle` element, as Figure 8–69 shows.

Example 8–69 Line Chart Definition with Filtered Series Styles

```
<dvtm:lineChart id="lineChart1"
                inlineStyle="width: 400px; height: 300px;"
                rolloverBehavior="dim"
                animationOnDisplay="auto"
                title="Line Chart"
                titleStyle="font-size: 12px; color: black;"
                value="#{bindings.lineData1.collectionModel}"
                var="row"
                seriesVar="series" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem series="#{row.series}"
                          group="#{row.group}"
                          value="#{row.value}"
                          color="#{row.color}" />
   </amx:facet>
   <amx:facet name="seriesStamp">
      <dvtm:seriesStyle series="#{series.name}"
                        color="red"
                        lineWidth="3"
                        lineStyle="solid"
                        rendered="#{series.name == 'Coke'}" />
      <dvtm:seriesStyle series="#{series.name}"
                        color="blue"
                        lineWidth="2"
                        lineStyle="dot"
                        rendered="#{series.name == 'Pepsi'}" />
   </amx:facet>
</dvtm:lineChart>
```

For information on attributes of the `lineChart` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the `lineChart`:

- `chartDataItem` (see Section 8.5.13.1, "Defining Chart Data Item")
- `xAxis`, `yAxis`, and `y2Axis` (see Section 8.5.13.3, "Defining X Axis, YAxis, and Y2Axis")
- `legend` (see Section 8.5.13.2, "Defining Legend")

8.5.7 How to Create a Pie Chart

You use a pie chart (`pieChart`) to illustrate proportional division of data, with each data item represented by a pie segment (slice).

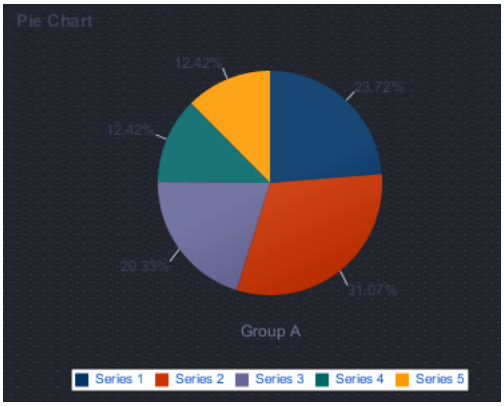
Example 8–70 shows the `pieChart` element defined in an ADF Mobile AMX file. The `dataStamp` facet is specified with a nested `pieDataItem` element.

Example 8–70 Pie Chart Definition

```
<dvtm:pieChart id="pieChart1"
               inlineStyle="width: 400px; height: 300px;"
               value="#{bindings.pieData.collectionModel}"
               var="row"
               title="Pie Chart"
               animationOnDisplay="zoom"
               animationDuration="3000" >
   <amx:facet name="dataStamp">
      <dvtm:pieDataItem label="#{row.name}" value="#{row.data}" />
```

```
    </amx:facet>
    <dvtm:legend position="bottom" />
</dvtm:pieChart>
```

Figure 8–59 Pie Chart at Design Time



The data model for a pie chart is represented by a collection of items that define individual pie data items. Typically, properties of each data item include the following:

- `label`: slice label;
- `value`: slice value.

The model might also define other properties of the data item, such as the following:

- `borderColor`: slice border color;
- `color`: slice color;
- `explode`: slice explosion offset.

For information on attributes of the `pieChart` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the `pieChart`:

- `pieDataItem` (see Section 8.5.13.4, "Defining Pie Data Item")
- `legend` (see Section 8.5.13.2, "Defining Legend")

8.5.8 How to Create a Scatter Chart

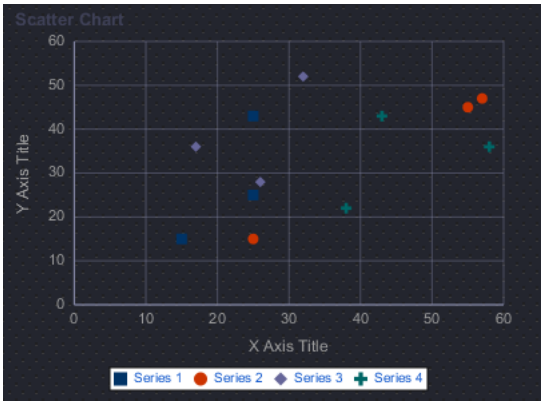
A scatter chart (`scatterChart`) displays data as unconnected dots that represent data items, where each item has `x`, `y` coordinates and size. In addition, each data item can have various style attributes, such as `color` and `shape`. You can either set properties of each data item individually, or categorize the data items into groups based on various criteria. You may use multiple grouping criteria at the same time, and may also use different style attributes to visualize the data items relationships. However, unlike line charts (see Section 8.5.6, "How to Create a Line Chart") or area charts (see Section 8.5.1, "How to Create an Area Chart"), scatter charts do not have a strict notion of the series and groups.

Example 8–71 shows the `scatterChart` element defined in an ADF Mobile AMX file. The `dataStamp` facet is specified with a nested `chartDataItem` element. The `color` and `shape` attributes of each data item are set individually based on the values supplied in the data model.

Example 8–71 Scatter Chart Definition

```
<dvtm:scatterChart id="scatterChart1"
                   inlineStyle="width: 400px; height: 300px;"
                   title="Scatter Chart"
                   animationOnDisplay="zoom"
                   animationDuration="3000"
                   value="#{bindings.scatterData.collectionModel}"
                   var="row" >
   <amx:facet name="dataStamp">
      <dvtm:chartDataItem label="#{row.group}"
                          x="#{row.data.x}"
                          y="#{row.data.y}">
         <dvtm:attributeGroups type="color shape"
                               value="#{row.series}" />
      </dvtm:chartDataItem>
   </amx:facet>
   <dvtm:xAxis title="X Axis Title" />
   <dvtm:yAxis title="Y Axis Title" />
   <dvtm:legend position="bottom" />
</dvtm:scatterChart>
```

Figure 8–60 Scatter Chart at Design Time



The data model for a scatter chart is represented by a collection of items (rows) that describe individual data items. Attributes of each data item are defined by stamping (`dataStamp`) and usually include the following:

- `x`, `y`: value coordinates (required);
- `size`: the size of the marker (optional).

The model might also define other properties of the data item, such as the following:

- `borderColor`: data item border color;
- `color`: data item color;
- `tooltip`: custom tooltip.

For information on attributes of the `scatterChart` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the `scatterChart`:

- `chartDataItem` (see Section 8.5.13.1, "Defining Chart Data Item")
- `xAxis`, `yAxis`, and `y2Axis` (see Section 8.5.13.3, "Defining X Axis, YAxis, and Y2Axis")

- `legend` (see Section 8.5.13.2, "Defining Legend")

8.5.9 How to Create a Spark Chart

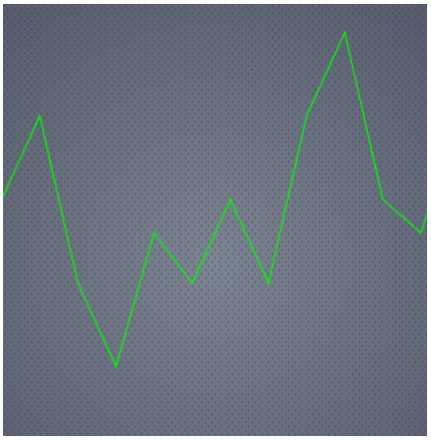
A spark chart (`sparkChart`) is a simple, condensed chart that displays trends or variations, often in the column of a table. The charts are often used in a dashboard to provide additional context to a data-dense display.

Example 8–72 shows the `sparkChart` element defined in an ADF Mobile AMX file. The `dataStamp` facet is specified with a nested `sparkDataItem` element.

Example 8–72 Spark Chart Definition

```
<dvtm:sparkChart id="sparkChart1"
                 value="#{bindings.sparkData.collectionModel}"
                 var="row"
                 type="line"
                 inlineStyle="width:400px; height:300px; float:left;">
   <amx:facet name="dataStamp">
      <dvtm:sparkDataItem value="#{row.value}" />
   </amx:facet>
</dvtm:sparkChart>
```

Figure 8–61 Spark Chart at Design Time



The data model for a spark chart is represented by a collection of items (rows) that describe individual spark data items. Typically, properties of each data item include the following:

- `value`: spark value.

For information on attributes of the `sparkChart` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child elements for the `sparkChart`:

- `sparkDataItem` (see Section 8.5.13.5, "Defining Spark Data Item")
- `referenceObjects` (see Section 8.5.13.6, "Defining Reference Object")

8.5.10 How to Create a LED Gauge

Unlike charts, gauges focus on a single data point and examine that point relative to minimum, maximum, and threshold indicators to identify problem areas. A LED (lighted electronic display) gauge (`ledGauge`) graphically depicts a measurement, such

as key performance indicator (KPI). There are several styles of LED gauges. The ones with arrows are used to indicate good (up arrow), fair (left- or right-pointing arrow), or poor (down arrow). You can specify any number of thresholds for a gauge. However, some LED gauges (such as those with arrow or triangle indicators) support a limited number of thresholds because there is a limited number of meaningful directions for them to point. For arrow or triangle indicators, the threshold limit is three.

Example 8–73 shows the `ledGauge` element defined in an ADF Mobile AMX file.

Example 8–73 LED Gauge Definition

```
<dvtm:ledGauge id="ledGauge1"
               value="65"
               type="circle"
               labelDisplay="on"
               inlineStyle="width: 100px; height: 80px; float: left;
                            border-color: navy; background-color: lightyellow;">
   <dvtm:threshold text="Low" maxValue="40" />
   <dvtm:threshold text="Medium" maxValue="60" />
   <dvtm:threshold text="High" maxValue="80" />
</dvtm:ledGauge>
```

Figure 8–62 LED Gauge at Design Time

The data model for a LED gauge is represented by a single metric value which is specified by the `value` attribute.

For information on attributes of the `ledGauge` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child element for the `ledGauge`:

- `threshold` (see Section 8.5.13.7, "Defining Threshold")

8.5.11 How to Create a Status Meter Gauge

A Status Meter Gauge (`statusMeterGauge`) indicates the progress of a task or the level of some measurement along a horizontal rectangular bar. An inner rectangle shows the current level of a measurement against the ranges marked on an outer rectangle.

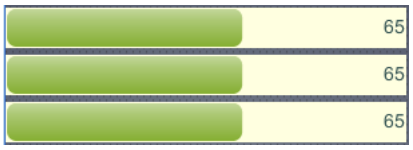
Example 8–74 shows the `statusMeterGauge` element defined in an ADF Mobile AMX file.

Example 8–74 Status Meter Gauge Definition

```
<dvtm:statusMeterGauge id="meterGauge1"
                       value="65"
                       animationOnDisplay="auto"
                       animationDuration="1000"
                       inlineStyle="width: 300px; height: 30px; float: left;
                                    border-color: black;
                                    background-color: lightyellow;"
                       labelDisplay="on"
                       minValue="0"
                       maxValue="100">
```

```
   <dvtm:threshold text="Low" maxValue="40" />
   <dvtm:threshold text="Medium" maxValue="60" />
   <dvtm:threshold text="High" maxValue="80" />
</dvtm:statusMeterGauge>
```

Figure 8–63 Status Meter Gauge at Design Time



The data model for a status meter gauge is a single metric value which is specified by the `value` attribute. In addition, the minimum and maximum values can also be specified by the `minValue` and `maxValue` attributes.

For information on attributes of the `statusMeterGauge` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child element for the `statusMeterGauge`:

- `threshold` (see Section 8.5.13.7, "Defining Threshold")

8.5.12 How to Create a Dial Gauge

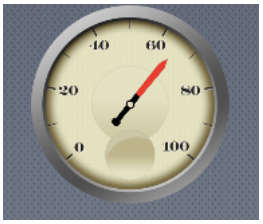
A Dial Gauge (`dialGauge`) specifies ranges of values (thresholds) that vary from poor to excellent. The gauge indicator specifies the current value of the metric while the graphic allows for evaluation of the status of that value.

Example 8–74 shows the `dialGauge` element defined in an ADF Mobile AMX file.

Example 8–75 Dial Gauge Definition

```
<dvtm:dialGauge id="dialGauge1"
                background="#{pageFlowScope.background}"
                indicator="#{pageFlowScope.indicator}"
                value="#{pageFlowScope.value}"
                minValue="#{pageFlowScope.minValue}"
                maxValue="#{pageFlowScope.maxValue}"
                animationDuration="1000"
                animationOnDataChange="auto"
                animationOnDisplay="auto"
                shortDesc="#{pageFlowScope.shortDesc}"
                labelDisplay="on"
                inlineStyle="#{pageFlowScope.inlineStyle}"
                styleClass="#{pageFlowScope.styleClass}"
                readOnly="true"
</dvtm:dialGauge>
```

Figure 8–64 Dial Gauge at Design Time



The data model for a dial gauge is a single metric value which is specified by the `value` attribute. In addition, the minimum and maximum values can be specified by the `minValue` and `maxValue` attributes.

For information on attributes of the `dialGauge` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

You can define the following child element for the `dialGauge`:

- `threshold` (see Section 8.5.13.7, "Defining Threshold")

8.5.13 How to Define Child Elements for Chart and Gauge Components

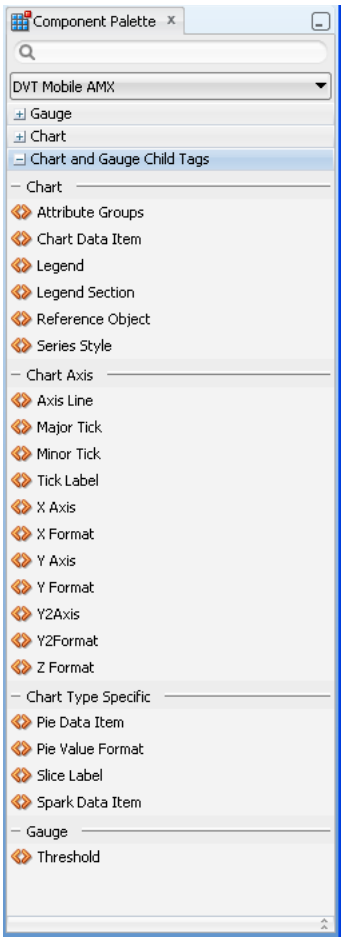
You can define a variety of child elements for charts and gauges. The following are some of these child elements:

- `chartDataItem` (see Section 8.5.13.1, "Defining Chart Data Item")
- `xAxis`, `yAxis`, and `y2Axis` (see Section 8.5.13.3, "Defining X Axis, YAxis, and Y2Axis")
- `legend` (see Section 8.5.13.2, "Defining Legend")
- `pieDataItem` (see Section 8.5.13.4, "Defining Pie Data Item")
- `sparkDataItem` (see Section 8.5.13.5, "Defining Spark Data Item")
- `referenceObject` (see Section 8.5.13.6, "Defining Reference Object")
- `threshold` (see Section 8.5.13.7, "Defining Threshold")

For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

In JDeveloper, the chart and gauge child components are located under **DVT Mobile AMX > Chart and Gauge Child Tags** in the Component Palette (see Figure 8–47).

Figure 8–65 Creating Chart and Gauge Child Components



8.5.13.1 Defining Chart Data Item

The Chart Data Item (`chartDataItem`) element specifies the parameters that chart data items use in all supported charts, except the pie chart.

For information on attributes of the `chartDataItem` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.13.2 Defining Legend

The Legend (`legend`) element specifies the legend parameters.

For information on attributes of the `legend` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.13.3 Defining X Axis, YAxis, and Y2Axis

X Axis (`xAxis`) and Y Axis (`yAxis`) elements define the X and Y axis for a chart. Y2Axis (`y2Axis`) defines an optional Y2 axis. These elements are declared as follows in an ADF Mobile AMX file:

```
<dvtm:xAxis scrolling="on" axisMinValue="0.0" axisMaxValue="50.0" />
```

For information on attributes and child elements of `xAxis`, `yAxis`, and `y2Axis` elements, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.13.4 Defining Pie Data Item

The Pie Data Item (`pieDataItem`) element specifies the parameters of the pie chart slices (see Section 8.5.7, "How to Create a Pie Chart").

For information on attributes of the `pieDataItem` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.13.5 Defining Spark Data Item

The Spark Data Item (`sparkDataItem`) element specifies the parameters of the spark chart items (see Section 8.5.9, "How to Create a Spark Chart").

For information on attributes of the `sparkDataItem` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.13.6 Defining Reference Object

The Reference Object (`referenceObject`) element specifies the reference objects for the axis of the spark chart (see Section 8.5.9, "How to Create a Spark Chart").

For information on attributes of the `referenceObjects` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.13.7 Defining Threshold

The Threshold (`threshold`) element specifies the threshold ranges of a gauge (see Section 8.5.10, "How to Create a LED Gauge" and Section 8.5.11, "How to Create a Status Meter Gauge").

For information on attributes of the `threshold` element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.14 How to Create a Geographic Map Component

A Geographic Map (`geographicMap`) represents business data in one or more interactive layers of information superimposed on a single map. You can configure this component to use either Google or Oracle maps as the underlying map provider (see Section 8.5.14.1, "Configuring Geographic Map Components With the Map Provider Information").

Example 8–76 shows the `geographicMap` element defined in an ADF Mobile AMX file.

Example 8–76 Geographic Map Definition

```
<dvtm:geographicMap id="g1" mapType="ROADMAP"
                    centerX="-98.57" centerY="39.82"
                    zoomLevel="2" initialZooming="auto">
   <dvtm:pointDataLayer id="pdl1"
                        var="row"
                        value="#{bindings.locationData.collectionModel}"
                        dataSelection="multiple"
                        selectionListener="#{myBean.doSomeGood}">
      <dvtm:pointLocation id="pl1" type="address" address="#{row.address}">
         <dvtm:marker shortDesc="#{row.shortDesc}" />
      </dvtm:pointLocation>
   </dvtm:pointDataLayer>
</dvtm:geographicMap>
```

You can define a `pointDataLayer` child element for the `geographicMap`. The `pointDataLayer` allows you to display data associated with a point on the map.

The `pointDataLayer` can have a `pointLocation` as a child element. The `pointLocation` specifies the columns in the data layer's model that determine the location of the data points. These locations can be represented either by address or by X and Y coordinates.

The `pointLocation` can have a `marker` as a child element. The `marker` is used to stamp out predefined or custom shapes associated with data points on the map. The `marker` supports a set of properties for specifying a URI to an image that is to be rendered as a marker. The marker can have a `convertNumber` as its child element (see Section 8.3.22, "How to Convert Numerical Values").

For information on attributes of the `geographicMap` element and its child elements, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.14.1 Configuring Geographic Map Components With the Map Provider Information

To configure a Geographic Map component to use a specific provider for the underlying map (Google or Oracle), you can set the following properties as name-value pairs in the application's `adf-config.xml` file:

- `mapProvider`: specify either `oraclemaps` or `googlemaps`.
- `geoMapKey`: specify the license key if the `mapProvider` is set to `googlemaps`.
- `geoMapClientId`: if the `mapProvider` is set to `googlemaps`, specify the client ID for Google maps business license.
- `mapViewerUrl`: if the `mapProvider` is set to `oraclemaps`, specify the map viewer URL for Oracle maps.
- `baseMap`: if the `mapProvider` is set to `oraclemaps`, specify the base map to use with Oracle maps.

> **Note:** To configure the Geographic Map component to use Google maps, you must obtain an appropriate license from Google.

Example 8–77 shows the configuration for Google maps.

Example 8–77 Google Maps Configuration

```
<adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/properties">
   <adf-property name="mapProvider" value="googlemaps"/>
   <adf-property name="geoMapKey" value="your key"/>
</adf-properties-child>
```

Example 8–78 shows the configuration for Oracle maps.

Example 8–78 Oracle Maps Configuration

```
<adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/properties">
   <adf-property name="mapProvider" value="oraclemaps"/>
   <adf-property name="mapViewerUrl"
                 value="http://elocation.oracle.com/mapviewer"/>
   <adf-property name="baseMap" value="ELOCATION_MERCATOR.WORLD_MAP"/>
</adf-properties-child>
```

If you do not specify the map provider information, the ADF Mobile AMX Geographic Map component uses Google maps for its map, but without the license key.

For information on the `adf-config.xml` file, see the following:

- Section 4.2.2.1, "About the Application-Level Resources"
- Table 4–1, " Mobile Application-Level Artifacts Accessed Through Application Resources"

8.5.15 How to Create a Thematic Map Component

A Thematic Map (`thematicMap`) represents business data as patterns in stylized areas or associated markers. Thematic maps focus on data without the geographic details.

ADF Mobile AMX supports most of the functionality, child elements, and properties of the Oracle ADF Thematic Map component. For more information, see the chapter on using map components in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*. The following is not supported:

- Drag and drop functionality.
- Control panel.
- Context menu.
- Custom regions (`customAreaLayer`).
- Drilling.
- Collapsible groups of legend sections.
- Collapsible legend.
- Patterns (`pattern`) on areas and markers.
- Images (ADF Mobile AMX Image (`image`) component). Instead, you can use images with the Marker (`marker`) child component. For more information, see Section 8.5.15.1, "How to Define Custom Markers."
- The `actionListener` attribute of the `marker`. Instead, the same functionality can be achieved by using the `selectionListener` or a combination of the `action` and `setPropertyListener` (see Section 8.10, "Using Event Listeners").

ADF Mobile AMX Thematic Map has the following functionality that is not available in Oracle ADF:

- Custom markers (see Section 8.5.15.1, "How to Define Custom Markers")
- Area isolation (see Section 8.5.15.2, "How to Define Isolated Areas")
- Initial zooming (see Section 8.5.15.3, "How to Enable Initial Zooming"
- Custom base maps (see Section 8.5.15.4, "How to Define a Custom Base Map"

Example 8–79 shows the `thematicMap` element and its children defined in an ADF Mobile AMX file.

Example 8–79 Defining Thematic Map

```
<dvtm:thematicMap id="tm1"
                  animationOnDisplay="#{pageFlowScope.animationOnDisplay}"
                  animationOnMapChange="#{pageFlowScope.animationOnMapChange}"
                  animationDuration="#{pageFlowScope.animationDuration}"
                  basemap="#{pageFlowScope.basemap}"
                           tooltipDisplay="#{pageFlowScope.tooltipDisplay}"
```

```
                inlineStyle="#{pageFlowScope.inlineStyle}"
                zooming="#{pageFlowScope.zooming}"
                panning="#{pageFlowScope.panning}"
                initialZooming="#{pageFlowScope.initialZooming}">
   <dvtm:areaLayer id="areaLayer1"
                layer="#{pageFlowScope.layer}"
                animationOnLayerChange=
                   "#{pageFlowScope.animationOnLayerChange}"
                areaLabelDisplay="#{pageFlowScope.areaLabelDisplay}"
                labelType="#{pageFlowScope.labelType}"
                rendered="#{pageFlowScope.rendered}">
      <dvtm:areaDataLayer id="areaDataLayer1"
                          animationOnDataChange=
                             "#{pageFlowScope.dataAnimationOnDataChange}"
                          animationDuration=
                             "#{pageFlowScope.dataAnimationDuration}"
                          dataSelection="#{pageFlowScope.dataSelection}"
                          var="row"
                          value="#{bindings.thematicMapData.collectionModel}">
         <dvtm:areaLocation id="areaLoc1" name="#{row.name}">
            <dvtm:area action="sales" id="area1" shortDesc="#{row.name}">
               <amx:setPropertyListener to=
                   "#{DvtProperties.areaChartProperties.dataSelection}"
                                        from="#{row.name}"
                                        type="action"/>
               <dvtm:attributeGroups id="ag1" type="color" value="#{row.cat1}" />
            </dvtm:area>
         </dvtm:areaLocation>
      </dvtm:areaDataLayer>
   </dvtm:areaLayer>
   <dvtm:legend position="end">
      <dvtm:legendSection source="ag1"/>
   </dvtm:legend>
</dvtm:thematicMap>
```

For information on attributes of the `thematicMap` element and its child elements, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.5.15.1 How to Define Custom Markers

ADF Mobile AMX Thematic Map does not support ADF Mobile AMX Image component. To use an image in the map's `pointLocation`, you can specify an image within the `pointLocation`'s `marker` child element by using its `source` attribute. If the `source` attribute is set on the Marker, its `shape` attribute is ignored by ADF Mobile AMX.

The `sourceHover`, `sourceSelected`, and `sourceHoverSelected` attributes allow you to specify images for hover and selection effects. If one of these is not specified, the image specified by the `source` attribute is used for that particular marker state. If `sourceSelected` is specified, then its value is used if `sourceHoverSelected` is not specified. The image can be of any format supported by the mobile device's browser, including PNG, JPG, SVG, and so on.

8.5.15.2 How to Define Isolated Areas

You can configure the ADF Mobile AMX Thematic Map component to render and zoom to fit on a single isolated area of the map by using the `isolatedRowKey` attribute of the `areaDataLayer`, in which case the rest of the areas in the area or area data layers is not rendered.

Note: You can isolate only one area on a map.

8.5.15.3 How to Enable Initial Zooming

The initial zooming allows the map component to be rendered as usual, and then zoom to fit on the data objects which includes both markers and areas. To enable this functionality, you use the `initialZooming` attribute of the Thematic Map.

8.5.15.4 How to Define a Custom Base Map

As part of the custom base map support, ADF Mobile AMX allows you to specify the following for the Thematic Map component:

- Layers with images for different resolutions.
- Point layers with named points that can be referenced from the Point Location (`pointLocation`).
- The Thematic Map's `source` attribute that points to the custom base map metadata XML file.

Note: ADF Mobile AMX does not support resource bundles for custom base maps. If you want to add locale-specific tool tips, you can use EL in the `shortDesc` attribute of the Marker (`marker`).

To create a custom base map, you specify an area layer which points to a definition in the metadata file (see Example 8–80). To define a basic custom base map, you specify a background layer and a pointer data layer. In the metadata file, you can specify different images for different screen resolutions and bidirectional screens (BiDi), similar to ADF Mobile AMX gauge components. Just like a gauge-type component, the Thematic Map chooses the correct image for the layer based on the screen resolution and locale.

Example 8–80 Metadata File With List of Images

```
<basemap id="car" >
   <layer id="exterior" >
      <image source="/maps/car-800x800.png"
             width="2560"
             height="1920" />
      <image source="/maps/car-800x800-bidi.png"
             width="2560"
             height="1920"
             bidi="true" />
      <image source="/maps/car-200x200.png"
             width="640"
             height="480" />
      <image source="/maps/car-200x200-bidi.png"
             width="640"
             height="480"
             bidi="true" />
   </layer>
</basemap>
```

Example 8–81 shows an ADF Mobile AMX file that declares a custom area layer with points. The ADF Mobile AMX file points to the metadata file shown in Example 8–80 containing a list of possible images.

Example 8–81 Declaring Custom Area Layer With Points

```
<dvtm:thematicMap id="tm1" basemap="car" source="customBasemaps/map1.xml" >
   <dvtm:areaLayer id="al1" layer="exterior" >
      <dvtm:pointDataLayer id="pdl1"
                           var="row"
                           value="{bindings.thematicMapData.collectionModel}" >
         <dvtm:pointLocation id="pl1"
                             type="pointXY"
                             pointX="#{row.x}"
                             pointY="#{row.y}" >
            <dvtm:marker id="m1" fillColor="#FFFFFF" shape="circle" />
         </dvtm:pointLocation>
      </dvtm:pointDataLayer>
   </dvtm:areaLayer>
</dvtm:thematicMap>
```

In the preceding example, the base map ID is matched with the `basemap` attribute of the `thematicMap`, and the layer ID is matched with the `layer` attribute of the `areaLayer`. The points are defined through the X and Y coordinates (just like for a predefined base map) to accommodate dynamic points that can change at the time the data are updated.

Example 8–82 shows an alternative way to declare a custom area layer with points. In this example, the `pointDataLayer` is a direct child of the `thematicMap`. Despite this variation, Example 8–82 renders the same result as the declaration demonstrated in Example 8–81.

Example 8–82 Declaring Custom Area Layer With Points Using Direct Child Element

```
<dvtm:thematicMap id="demo1" basemap="car" source="customBasemaps/map1.xml" >
   <dvtm:areaLayer id="al1" layer="exterior" />
   <dvtm:pointDataLayer id="pdl1"
                        var="row"
                        value="{bindings.thematicMapData.collectionModel}" >
      <dvtm:pointLocation id="pl1"
                          type="pointXY"
                          pointX="#{row.x}"
                          pointY="#{row.y}" >
         <dvtm:marker id="m1" fillColor="#FFFFFF" shape="circle" />
      </dvtm:pointLocation>
   </dvtm:pointDataLayer>
</dvtm:thematicMap>
```

To create a custom base map with static points, you specify the points by name in the metadata file shown in Example 8–83. This process is similar to adding city markers for a predefined base map.

Example 8–83 Metadata File With List of Named Points

```
<basemap id="car" >
   <layer id="exterior" >
      <image source="/maps/car-800x800.png"
             width="2560"
             height="1920" />
      <image source="/maps/car-800x800-bidi.png"
             width="2560"
             height="1920"
             bidi="true" />
      <image source="/maps/car-200x200.png"
             width="640"
```

```
            height="480" />
      <image source="/maps/car-200x200-bidi.png"
            width="640"
            height="480"
            bidi="true" />
   </layer>
   <points >
      <point name="hood" x="219.911" y="329.663" />
      <point name="frontLeftTire" x="32.975" y="32.456" />
      <point name="frontRightTire" x="10.334" y="97.982" />
   </points>
</basemap>
```

Example 8–84 shows an ADF Mobile AMX file that declares a custom area layer with named points. The ADF Mobile AMX file points to the metadata file shown in Example 8–80 containing a list of points and their names.

Example 8–84 Declaring Custom Area Layer With Named Points

```
<dvtm:thematicMap id="demo1" basemap="car" source="customBasemaps/map1.xml" >
   <dvtm:areaLayer id="al1" layer="exterior" />
   <dvtm:pointDataLayer id="pdl1"
                        var="row"
                        value="#{bindings.thematicMapData.collectionModel}" >
      <dvtm:pointLocation id="pl1" type="pointName" pointName="#{row.name}" >
         <dvtm:marker id="m1" fillColor="#FFFFFF" shape="circle" />
      </dvtm:pointLocation>
   </dvtm:pointDataLayer>
</dvtm:thematicMap>
```

8.5.15.5 How to Apply Custom Styling to the Thematic Map Component

You can style the Thematic Map component by overwriting the default CSS settings or using a custom JavaScript file. For more information on how to extend these files, see Section 8.6.3, "How to Style Data Visualization Components."

Example 8–85 shows the default CSS styles for the Thematic Map component.

Example 8–85 CSS Styling

```
.dvtm-thematicMap {
   background-color: #FFFFFF;
   -webkit-user-select: none;
   -webkit-touch-callout: none;
   -webkit-tap-highlight-color: rgba(0,0,0,0);
}

.dvtm-areaLayer {
   background-color: #B8CDEC;
   border-color: #FFFFFF;
   border-width: 0.5px;
   /* border style and color must be set when setting border width */
   border-style: solid;
   color: #000000;
   font-family: tahoma, sans-serif;
   font-size: 13px;
   font-weight: bold;
   font-style: normal;
}
```

```
.dvtm-area {
   border-color: #FFFFFF;
   border-width: 0.5px;
   /* border style and color must be set when setting border width */
   border-style: solid;
}

.dvtm-marker {
   background-color: #61719F;
   opacity: 0.7;
   color: #FFFFFF;
   font-family: tahoma, sans-serif;
   font-size: 13px;
   font-weight: bold;
   font-style: normal;
}
```

Some of the style settings cannot be specified using CSS. Instead, you must define them using a custom JavaScript file. Example 8–86 shows how to apply custom styling to the Thematic Map component without using CSS.

Example 8–86 Non-CSS Custom Styling

my-custom.js:

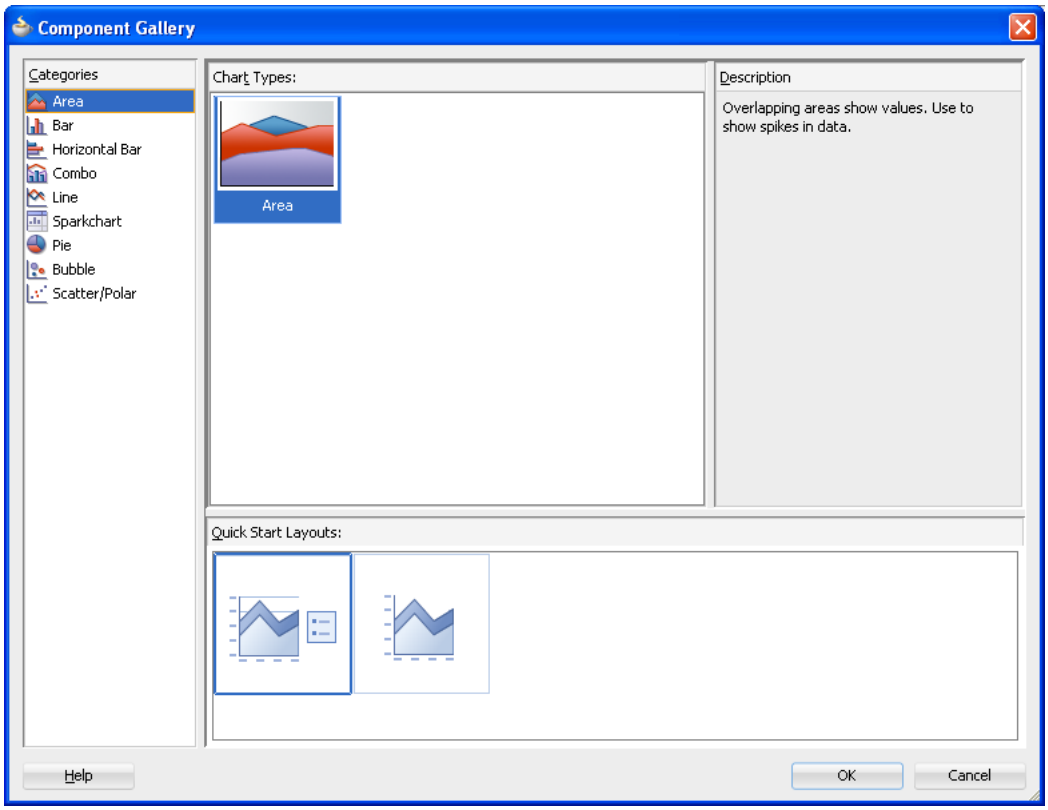
```
   CustomThematicMapStyle = {
      // selected area properties
      'areaSelected': {
         // selected area border color
         'borderColor': "#000000",
         // selected area border width
         'borderWidth': '1.5px'
      },

      // area properties on mouse hover
      'areaHover': {
         // area border color on hover
         'borderColor': "#FFFFFF",
         // area border width on hover
         'borderWidth': '2.0px'
      },

      // marker properties
      'marker': {
         // separator upper color
         'scaleX': 1.0,
         // separator lower color
         'scaleY': 1.0,
         // should display title separator
         'type': 'circle'
      },

      // thematic map legend properties
      'legend': {
         // legend position, such as none, auto, start, end, top, bottom
         'position': "auto"
      }
   };

})();
```

Note that you cannot change the name and the property names of the `CustomThematicMapStyle` object. Instead, you can modify specific property values to suit the needs of your application. For information on how to add custom CSS and JavaScript files to your application, see Section 5.9, "Defining the Content Types for an Application Feature."

8.5.16 How to Create Databound Data Visualization Components

You can declaratively create a databound chart, gauge, or map using a data collection inserted from the Data Controls panel (see Chapter 7, "Creating ADF Mobile AMX Pages"). The Component Gallery dialog that Figure 8–69 shows allows you to choose from a number of data visualization component categories, types, and layout options.

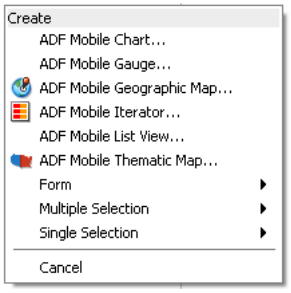
Figure 8–66 Component Gallery



Note: Some data visualization component types require very specific kinds of data. If you bind a component to a data collection that does not contain sufficient data to display the component type requested, then the component is not displayed and a message about insufficient data appears.

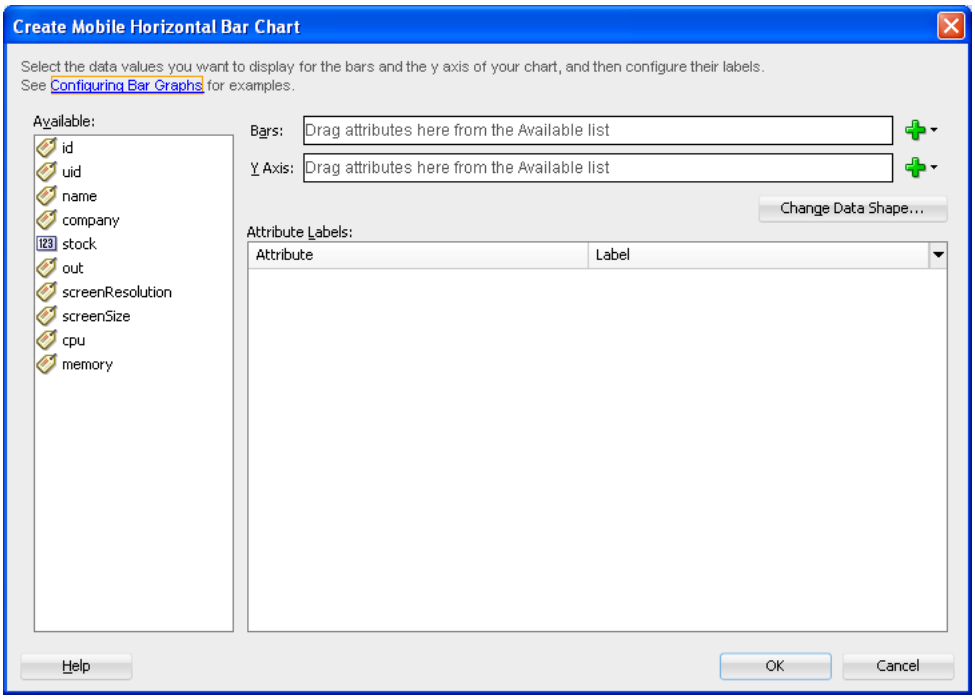
To trigger the display of the **Component Gallery**, you drag and drop a collection from the Data Controls panel onto the ADF Mobile AMX page, and then select either **ADF Mobile Chart**, **ADF Mobile Gauge**, **ADF Mobile Geographic Map**, or **ADF Mobile Thematic Map** from the context menu that appears (see Figure 8–67).

Figure 8–67 Creating Databound DVT Components



After you select the category, type, and layout for your new databound component from the Component Gallery and click **OK**, you can start binding the data collection attributes in the DVT component using data binding dialogs. The name of the dialog and the input field labels depend on the category and type of the DVT component that you selected. For example, if you select Horizontal Bar as the category and Bar as the type, then the name of the dialog that appears is Create Mobile Horizontal Bar Chart, and the input field is labeled Bars, as Figure 8–68 shows.

Figure 8–68 Specifying Data Values for Databound Chart



The attributes in a data collection can be data values or categories of data values. Data values are numbers represented by markers, like bar height, or points in a scatter chart. Categories of data values are members represented as axis labels. The role that an attribute plays in the bindings (either data values or identifiers) is determined by both its data type (chart requires numeric data values) and where it is mapped (for example, Bars or X Axis).

After completing the data binding dialog, you can use the Property Inspector to specify settings for the component attributes. You can also use the child elements associated with the component to further customize it (see Section 8.5.13, "How to Define Child Elements for Chart and Gauge Components").

For more information, see the following sections of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework:*

- "Creating Databound Graphs"
- "Creating Databound Gauges"
- Chapter on creating databound geographic and thematic maps

8.6 Styling UI Components

ADF Mobile enables you to employ CSS to apply style to UI components.

8.6.1 How to Use Component Attributes to Define Style

You style your UI components by setting the following attributes:

- `styleClass` attribute defines a CSS style class to use for your layout component:

  ```
  <amx:panelPage styleClass="#{pageFlowScope.pStyleClass}">
  ```

  You can define the style class for layout, command, and input components in an ADF Mobile AMX page or in a skinning CSS file, in which case a certain style is applied to all components within the ADF Mobile AMX application feature (see Section 8.6.2, "What You May Need to Know About Skinning"). Alternatively, you can use the public style classes provided by ADF Mobile.

  > **Note:** The CSS file is not accessible from JDeveloper. Instead, ADF Mobile injects this file into the package at build or deploy time, upon which the CSS file appears in the `css` directory under the `Web Content` root directory.

- `inlineStyle` attribute defines a CSS style to use for any UI component and represents a set of CSS styles that are applied to the root DOM element of the component:

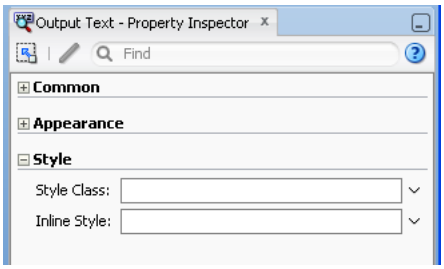
  ```
  <amx:outputText inlineStyle="color:red;">
  ```

  You should use this attribute when basic style changes are required.

  > **Note:** Some UI components are rendered with such subelements as HTML `div` elements and more complex markup. As a result, setting the `inlineStyle` attribute on the parent component may not produce the desired effect. In such cases, you should examine the generated markup and, instead of defining the `inlineStyle` attribute, apply a CSS class that would propagate the style to the subelement.
  >
  > For information on how to configure JavaScript debugging, see Section 18.3.4, "How to Enable Debugging of Java Code and JavaScript."

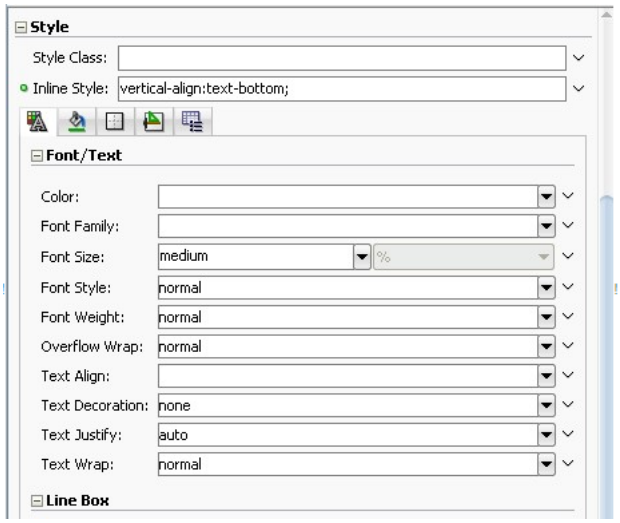
These attributes are displayed in the **Style** section in the Property Inspector, as Figure 8–69 shows.

Figure 8–69 Style Section of the Property Inspector



ADF Mobile AMX provides a drop-down editor that you can use to set various properties of the `inlineStyle` attribute (see Figure 8–70).

Figure 8–70 Inline Style Editor



For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

8.6.2 What You May Need to Know About Skinning

Skinning allows you to define and apply a uniform style to all UI components within an ADF Mobile AMX application feature to create a theme for the entire feature. For more information, see Section 5.11, "Skinning ADF Mobile Applications."

8.6.3 How to Style Data Visualization Components

Most of the style properties of ADF Mobile AMX data visualization components are defined in the `dvtm.css` file located in the `css` directory. You can override the default values by adding a custom CSS file with custom style definitions at the application feature level (see Section 5.11.4, "Overriding the Default Skin Styles").

Some of the style properties cannot be mapped to CSS and have to be defined in custom JavaScript files. These properties include the following:

- Background and needle images for the Dial Gauge component (see Section 8.5.12, "How to Create a Dial Gauge").
- Base maps for the Thematic Map component (see Section 8.5.15.4, "How to Define a Custom Base Map").

- Style properties of the Geographic Map component (see Section 8.5.14, "How to Create a Geographic Map Component").

You should specify these custom JavaScript files in the Includes section at the application feature level (see Section 5.9.1, "How to Define the Application Content"). By doing so, you override the default style values defined in the XML style template. Example 8–87 shows a JavaScript file similar to the one you would add to your ADF Mobile project that includes the ADF Mobile AMX application feature with data visualization components which require custom styling of properties that cannot be styled using CSS.

Example 8–87 Defining Custom Style Properties

my-custom.js:

```
CustomChartStyle = {

   // common chart properties
   'chart': {
      // text to be displayed, if no data is provided
      'emptyText': null,
      // animation effect when the data changes
      'animationOnDataChange': "none",
      // animation effect when the chart is displayed
      'animationOnDisplay': "none",
      // time axis type - disabled / enabled / mixedFrequency
      'timeAxisType': "disabled"
   },

   // chart title separator properties
   'titleSeparator': {
      // separator upper color
      'upperColor': "#74779A",
      // separator lower color
      'lowerColor': "#FFFFFF",
      // should display title separator
      'rendered': false
   },

   // chart legend properties
   'legend': {
      // legend position none / auto / start / end / top / bottom
      'position': "auto"
   },

   // default style values
   'styleDefaults': {
      // default color palette
      'colors': ["#003366", "#CC3300", "#666699", "#006666", "#FF9900",
                 "#993366", "#99CC33", "#624390", "#669933", "#FFCC33",
                 "#006699", "#EBEA79"],
      // default shapes palette
      'shapes': ["circle", "square", "plus", "diamond",
                 "triangleUp", "triangleDown", "human"],
      // series effect
      'seriesEffect': "gradient",
      // animation duration in ms
      'animationDuration': 1000,
      // animation indicators - all / none
      'animationIndicators': "all",
```

```
            // animation up color
            'animationUpColor': "#0099FF",
            // animation down color
            'animationDownColor': "#FF3300",
            // default line width (appllicable to line chart)
            'lineWidth': 3,
            // default line style (appllicable to line chart)
            // solid / dotted / dashed
            'lineStyle': "solid",
            // should markers be displayed (appllicable to line and area charts)
            'markerDisplayed': false,
            // default marker color
            'markerColor': null,
            // default marker shape
            'markerShape': "auto",
            // default marker size
            'markerSize': 8,
            // pie feeler color (appllicable to pie chart only)
            'pieFeelerColor': "#BAC5D6",
            // slice label position and text type (appllicable to pie chart only)
            'sliceLabel': {
               'position': "outside",
               'textType': "percent" }
         }
      };

      CustomGaugeStyle = {
         // default animation duration in milliseconds
         'animationDuration': 1000,
         // default animation effect on data change
         'animationOnDataChange': "none",
         // default animation effect on gauge display
         'animationOnDisplay': "none",
         // default visual effect
         'visualEffects': "auto"
      };
...
}
```

After the JavaScript file has been defined, you can uncomment and modify any values. You add this file as an included feature in the `adfmf-feature.xml` file, as Example 8–88 shows.

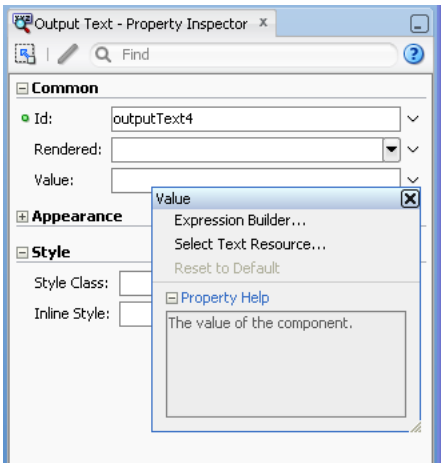
Example 8–88 Including Custom Style File in the Application Feature

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                          xmlns:adfmf="http://xmlns.oracle.com/adf/mf">
   <adfmf:feature id="feature1" name="feature1">
      <adfmf:content id="feature1.1">
         <adfmf:amx file="feature1/untitled1.amx">
            <adfmf:includes>
               <adfmf:include type="StyleSheet" file="css/custom.css"/>
               <adfmf:include type="JavaScript" file="feature1/js/my-custom.js"/>
            </adfmf:includes>
         </adfmf:amx>
      </adfmf:content>
   </adfmf:feature>
</adfmf:features>
```

8.7 Localizing UI Components

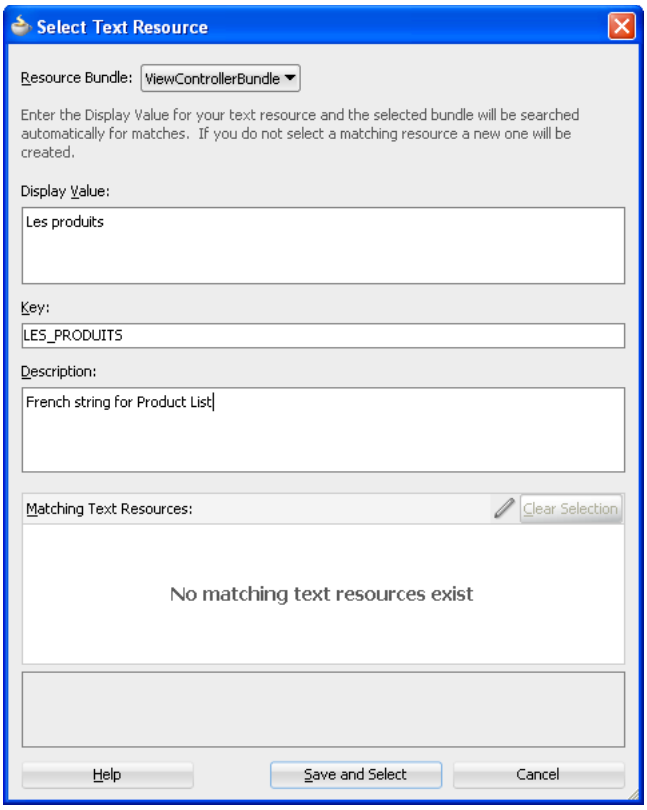
In your ADF Mobile AMX page, you can localize the text that UI components display by using the standard resource bundle provided by JDeveloper. You do so by selecting a component and one of its text-presenting properties whose value you intend to localize, and then choosing **Select Text Resource** in the **Value** box in the Property Inspector (see Figure 8–71).

Figure 8–71 Selecting Text Resource



This will display the standard ADF **Select Text Resource** dialog that Figure 8–72 shows. You use this dialog to enter or find a string reference for the property you are modifying.

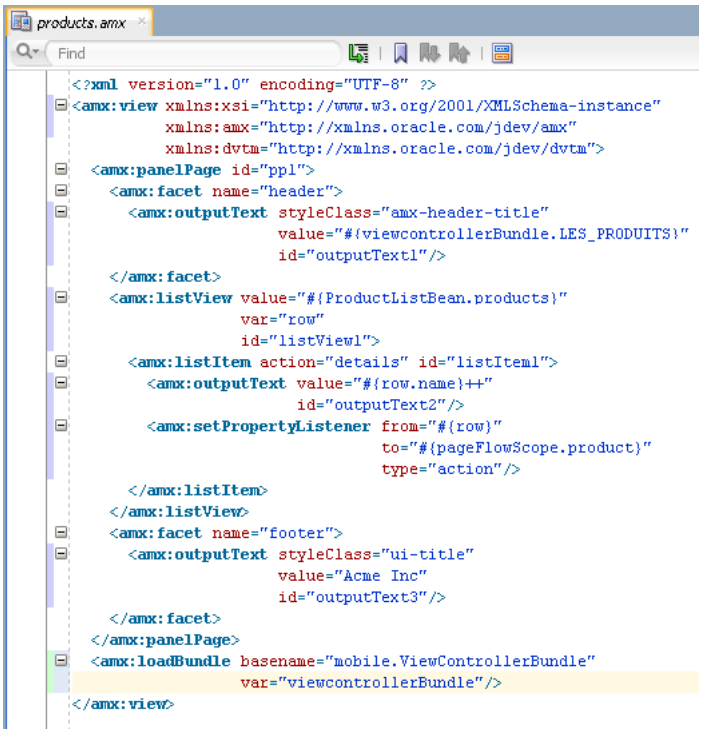
Figure 8–72 Select Text Resource Dialog



After you have defined a localized string resource, the EL for that reference is automatically placed in the property from which the Select Text Resource dialog was launched.

Figure 8–73 shows the changes in the ADF Mobile AMX file.

Figure 8–73 Localized String in ADF Mobile AMX File



For more information, see Section 5.10, "Working with Resource Bundles."

8.8 Understanding ADF Mobile Support for Accessibility

When developing ADF Mobile applications, you may need to accommodate visually and physically impaired users by addressing accessibility issues. User agents, such as web browsers rendering to nonvisual media (for example, a screen reader) can read text descriptions of UI components to provide useful information to impaired users. ADF Mobile AMX UI components are designed to be compliant with the following accessibility standards:

- The Accessible Rich Internet Applications (WAI-ARIA) 1.0 specification.

  For more information, see the following:

  - "Introduction" to WAI-ARIA 1.0 specification at `http://www.w3.org/TR/wai-aria/introduction`
  - "Using WAI-ARIA" at `http://www.w3.org/TR/wai-aria/usage`
  - Section 8.8.2, "What You May Need to Know About the Basic WAI-ARIA Terms"

- The Oracle Global HTML Accessibility Guidelines (OGHAG).

  For more information, see Section 8.8.3, "What You May Need to Know About the Oracle Global HTML Accessibility Guidelines."

- iOS Accessibility guidelines.

  For more information, see the *Accessibility Programming Guide for iOS* at `http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility/Introduction/Introduction.html`.

Accessible components do not change their appearance nor is the application logic affected by the introduction of such components.

To enable the proper functioning of the accessibility in your ADF Mobile AMX application feature, follow these guidelines:

- The navigation must not be more than three levels deep and it must be easy for the user to traverse back to the home screen.
- Keep scripting to a minimum.
- Do not provide direct interaction with the DOM.
- Do not use JavaScript time-outs.
- Avoid unnecessary focus changes
- Provide explicit popup triggers
- If needed, utilize the WAI-ARIA live region (see Section 8.8.2, "What You May Need to Know About the Basic WAI-ARIA Terms").
- Keep CSS use to a minimum.
- Try not to override the default component appearance.
- Choose scalable size units.
- Do not use CSS positioning.

For more information, see "Web Content Accessibility and Mobile Web: Making a Web Site Accessible Both for People with Disabilities and for Mobile Devices" at `http://www.w3.org/WAI/mobile`.

8.8.1 How to Configure UI Components for Accessibility

ADF Mobile AMX UI components have built-in accessibility support, with most components being subject to the accessibility audit (see Figure 8–75).

Table 8–8 lists UI components and their attributes that you can set through the Accessibility section of the Property Inspector.

Table 8–8 UI Components with Configurable Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
Button (`commandButton`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Select Button (`selectOneButton`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Link (`commandLink`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Link Go (`goLink`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Carousel (`carousel`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.

Table 8–8 (Cont.) UI Components with Configurable Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
CarouselItem (`carouselItem`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
List Item (`listItem`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Popup (`popup`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be present and describe the action that will take place.
Image (`image`)	Short Desc (`shortDesc`)	The `shortDesc` attribute should be specified. If the image is used for decorative purposes, it can be empty.
Input Text (`inputText`)	Hint Text (`hintText`)	The `hintText` attribute should be present and describe what the field should contain.
Panel Group Layout (`panelGroupLayout`)	Landmark (`landmark`)	NA [1]

[1] The landmark attribute has a default value (none) and is not subject to the accessibility audit.

You use the `shortDesc` attribute for different purposes for different components. For example, if you set the shortDesc attribute for the Image component, in the ADF Mobile AMX file it will appear as a value of the `alt` attribute of the `image` element.

The value of the `shortDesc` attribute can be localized.

For the Panel Group Layout component, you define the landmark role type (see Table 8–13, " Landmark Roles") that is applicable as per the context of the page. You can set one of the following values for the `landmark` attribute:

- default (none)
- application
- banner
- complementary
- contentinfo
- form
- main
- navigation
- search

Table 8–9 lists UI components whose accessible attributes defined by WAI-ARIA specification are automatically applied at run time and that you cannot modify.

Table 8–9 UI Components with Static Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
Input Date (`inputDate`)	Label (`label`)	`inputDate` is not labeled. The label attribute of `inputDate` should be specified

Table 8–9 (Cont.) UI Components with Static Accessibility Attributes

Component	Accessibility Attribute	Accessibility Audit Message
Input Number Slider (`inputNumberSlider`)	Label (`label`)	`inputNumberSlider` is not labeled. The label attribute of `inputNumberSlider` should be specified
Panel Label and Message (`panelLabelAndMessage`)	Label (`label`)	`panelLabelAndMessage` is not labeled. The label attribute of `panelLabelAndMessage` should be specified
Select Item (`selectItem`)	Label (`label`)	`selectItem` is not labeled. The label attribute of `selectItem` should be specified
Checkbox (`selectBooleanCheckbox`)	Label (`label`)	`selectBooleanCheckbox` is not labeled. The label attribute of `selectBooleanCheckbox` should be specified
Boolean Switch (`selectBooleanSwitch`)	Label (`label`)	`selectBooleanSwitch` is not labeled. The label attribute of `selectBooleanSwitch` should be specified
Radio Button (`selectOneRadio`)	Label (`label`)	`selectOneRadio` is not labeled. The label attribute of selectOneRadio should be specified
Select Many Checkbox (`selectManyCheckbox`)	Label (`label`)	`selectManyCheckbox` is not labeled. The label attribute of selectManyCheckbox should be specified
Choice (`selectOneChoice`)	Label (`label`)	`selectOneChoice` is not labeled. The label attribute of `selectOneChoice` should be specified
Output Text (`outputText`)	Value (`value`)	NA [1]

[1] The value attribute is not subject to the accessibility audit.

You can configure the accessibility audit rules using JDeveloper's Preferences dialog as follows:

1. In JDeveloper, select **Tools > Preferences** from the main menu.
2. From the list of preferences, select **Audit > Profiles**.
3. On the **Audit: Profiles** pane, expand the **Application Development Framework (ADF)** node from the tree of rules, and then select ADF Mobile Framework > Accessibility.
4. Select the accessibility audit rules to apply to your application, as Figure 8–74 shows.

Figure 8–74 Setting Accessibility Audit Rules

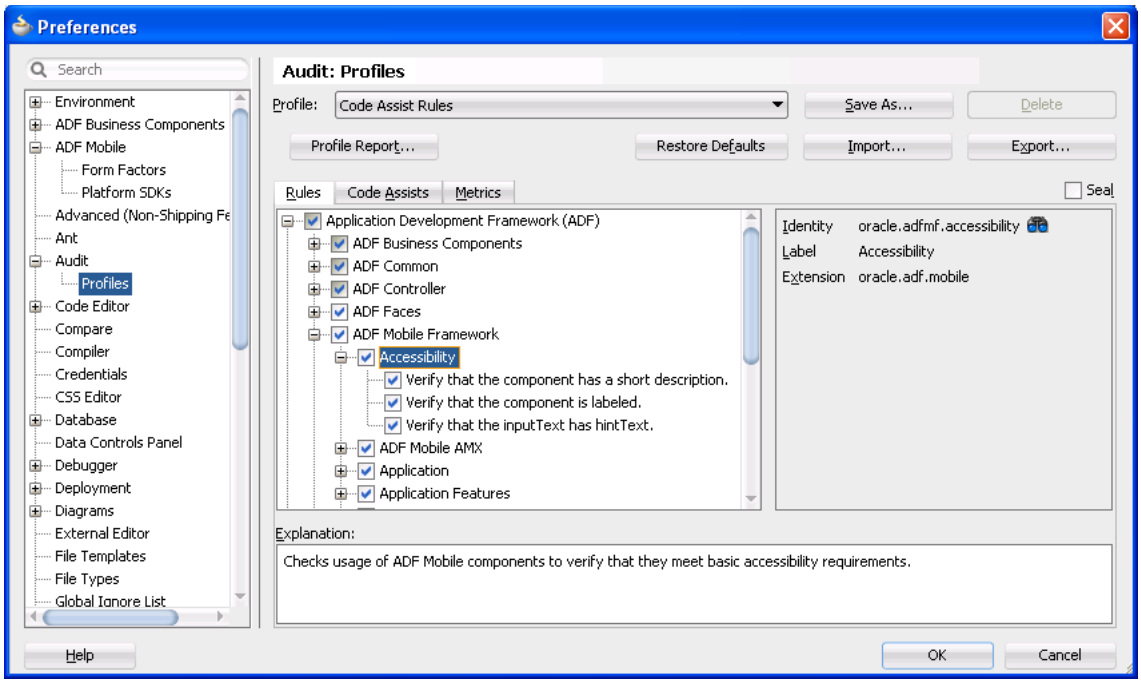
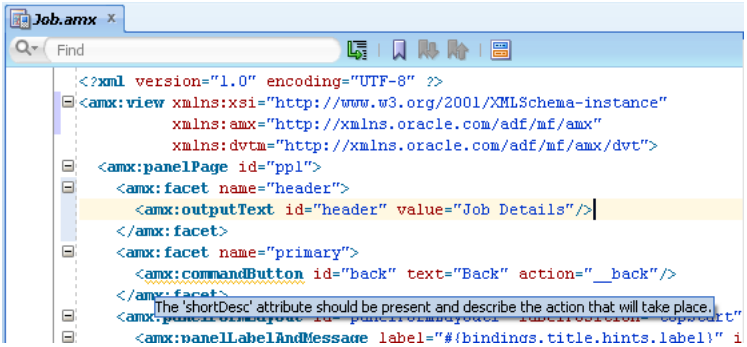


Figure 8–75 shows the accessibility audit warning displayed in JDeveloper.

Figure 8–75 Accessibility Audit Warning



For information on how to test your accessible ADF Mobile AMX application feature, see Section 18.2.1, "How to Perform Accessibility Testing on iOS-Powered Devices."

8.8.2 What You May Need to Know About the Basic WAI-ARIA Terms

As stated in the WAI-ARIA 1.0 specification, complex web applications become inaccessible when assistive technologies cannot determine the semantics behind portions of a document or when the user is unable to effectively navigate to all parts of it in a usable way. WAI-ARIA divides the semantics into roles (the type defining a user interface element), and states and properties supported by the roles. The following semantic associations form the base for the WAI-ARIA terms:

- Role
- Landmark
- Live region

For more information, see http://www.w3.org/TR/wai-aria/terms.

The following tables list role categories (as defined in the WAI-ARIA 1.0 specification) that are applicable to ADF Mobile.

Table 8–10 lists abstract roles that are used to support the WAI-ARIA role taxonomy for the purpose of defining general role concepts.

Table 8–10 Abstract Roles

Abstract Role	Description
input	A generic type of widget that allows the user input.
landmark	A region of the page intended as a navigational landmark.
select	A form widget that allows the user to make selections from a set of choices.
widget	An interactive component of a graphical user interface.

Table 8–11 lists widget roles that act as standalone user interface widgets or as part of larger, composite widgets.

Table 8–11 Widget Roles

Widget Role	Description	Widget Required States
alertdialog	A type of dialog that contains an alert message, where initial focus moves to an element within the dialog.	aria-labelledby, aria-describedby
button	An input that allows for user-triggered actions when clicked or pressed.	aria-expanded (state), aria-pressed (state)
checkbox	A checkable input that has three possible values: `true`, `false`, or `mixed`.	aria-checked (state)
dialog	A dialog represented by an application window that is designed to interrupt the current processing of an application in order to prompt the user to enter information or require a response.	aria-labelledby, aria-describedby
link	An interactive reference to an internal or external resource that, when activated, causes the user agent to navigate to that resource.	aria-disabled (state), aria-describedby
option	A selectable item in a select list.	aria-labelledby, aria-checked (state), aria-selected (state)
radio	A checkable input in a group of radio roles, only one of which can be checked at a time.	aria-checked (state), aria-disabled (state)
slider	A user input where the user selects a value from within a given range.	aria-valuemax, aria-valuemin, aria-valuenow, aria-disabled (state)
listbox	A widget that allows the user to select one or more items from a list of choices.	aria-live
radiogroup	A group of radio buttons.	aria-disabled (state)
listitem	A single item in a list or directory.	aria-describedby

Table 8–11 (Cont.) Widget Roles

Widget Role	Description	Widget Required States
textbox	Input that allows free-form text as its value.	aria-labelledby, aria-readonly, aria-required, aria-multiline, aria-disabled (state)

Table 8–12 lists document structure roles that describe structures that organize content in a page. Typically, document structures are not interactive.

Table 8–12 Document Structure Roles

Document Structure Role	Description
img	A container for a collection of elements that form an image.
list	A group of non-interactive list items.
listitem	A single item in a list or directory.

Table 8–13 lists landmark roles that represent regions of the page intended as navigational landmarks.

Table 8–13 Landmark Roles

Landmark Role	Description
application	A region declared as a web application (as opposed to a web document).
banner	A region that contains mostly site-oriented content (rather than page-specific content).
complementary	A supporting section of a document designed to be complementary to the main content at a similar level in the DOM hierarchy, but that remains meaningful when separated from the main content.
contentinfo	A large perceivable region that contains information about the parent document.
form	A region that contains a collection of items and objects that, as a whole, combine to create a form.
main	The main content of a document.
navigation	A collection of navigational elements (usually links) for navigating the document or related documents.
search	A region that contains a collection of items and objects that, as a whole, combine to create a search facility.

For the majority of ADF Mobile UI components, you cannot modify accessible WAI-ARIA attributes. For some components, you can set special accessible attributes at design time, and for the Panel Group Layout, you can use the WAI-ARIA landmark role type. For more information, see Section 8.8.1, "How to Configure UI Components for Accessibility."

8.8.3 What You May Need to Know About the Oracle Global HTML Accessibility Guidelines

The Oracle Global HTML Accessibility Guidelines (OGHAG) is a set of scripting standards for HTML that Oracle follows. These standards represent a combination of Section 508 (see http://www.section508.gov) and Web Content Accessibility Guidelines (WCAG) 1.0 level AA (see http://www.w3.org/TR/WCAG10), with improved wording and checkpoint measurements.

For more information, see *Oracle's Accessibility Philosophy and Policies* at http://www.oracle.com/us/corporate/accessibility/policies/index.html.

8.9 Validating Input

ADF Mobile allows you to inform the end user about data input errors and other conditions that occur during data input. Depending on their type (error or warning), validation messages have a different look and feel.

The user input validation is triggered when an input is submitted: Input Text components are automatically validated when the end user leaves the field; for selection components, such as a Checkbox or Choice, the validation occurs when the end user makes a selection. For validation purposes, UI components on an ADF Mobile AMX page are grouped together within a Validation Group operation (`validationGroup`) to define components whose input is to be validated when the submit operation takes place. A Validation Behavior (`validationBehavior`) component defines which Validation Group is to be validated before a command component's action is taken. A command component can have multiple child Validation Behavior components. Validation does not occur if a component does not have a Validation Behavior defined for it.

Note: You cannot define nested Validation Group operations.

The following is an invalid definition of a Validation Group:

```
<amx:view>
   <amx:panelPage>
      <amx:validationGroup>
         <amx:panelGroupLayout>
            <amx:validationGroup/>
         <amx:panelGroupLayout/>
      </amx:validationGroup>
   </amx:panelPage>
</amx:view>
```

The following is a valid definition:

```
<amx:view>
   <amx:panelPage>
      <amx:validationGroup>
   </amx:panelPage>
   <amx:popup>
      <amx:validationGroup>
   </amx:popup>
</amx:view>
```

If an ADF Mobile AMX page contains any validation error messages, the end user is prevented from navigating off the page using command components, such as List

Item, Link, and Button, unless those components have their `immediate` attribute set to `true`. Messages containing warnings do not halt the navigation.

Example 8–89 shows how to define validation elements, including multiple Validation Group and Validation Behavior operations, in an ADF Mobile AMX file.

Example 8–89 Defining Input Validation

```
<amx:panelPage id="pp1">
   <amx:facet name="header">
      <amx:outputText id="outputText1" value="Validate"/>
   </amx:facet>
   <amx:facet name="secondary">
      <amx:commandButton id="commandButton2" action="go" text="Save">
         <amx:validationBehavior disabled="#{pageFlowScope.myPanel ne 'panel1'}"
                                 group="group1"/>
         <amx:validationBehavior disabled="#{pageFlowScope.myPanel ne 'panel2'}"
                                 group="group2"/>
         <!-- invalid, should be caught by audit rule but for any reason
         if group not found at run time, this validate is ignored -->
         <amx:validationBehavior disabled="false" group="groupxxx"/>
         <!-- group is not found at run time, this validate is ignored -->
         <amx:validationBehavior disabled="false" group="group3"/>
      </amx:commandButton>
   </amx:facet>
   <amx:panelSplitter selectedItem="#{pageFlowScope.myPanel}">
      <amx:panel id="panel1">
         <amx:validationGroup id="group1">
            <amx:panelFormLayout id="pfl1">
               <amx:inputText value="#{bindings.first.inputValue}"
                              required="true"
                              label="#{bindings.first.hints.label}"
                              id="inputText1"/>
               <amx:inputText value="#{bindings.last.inputValue}"
                              label="#{bindings.last.hints.label}"
                              id="inputText2"/>
            </amx:panelFormLayout>
         </amx:validationGroup>
      </amx:panel>
      <amx:panel id="panel2">
         <amx:validationGroup id="group2">
            <amx:panelFormLayout id="pfl2">
               <amx:inputText value="#{bindings.salary.inputValue}"
                              label="#{bindings.first.hints.label}"
                              id="inputText3"/>
               <amx:inputText value="#{bindings.last.inputValue}"
                              label="#{bindings.last.hints.label}"
                              id="inputText4"/>
            </amx:panelFormLayout>
         </amx:validationGroup>
      </amx:panel>
   </amx:panelSplitter>
   <amx:panelGroupLayout id="pgl1" rendered="false">
      <amx:validationGroup id="group3">
         <amx:panelFormLayout id="pfl4">
            <amx:inputText value="#{bindings.salary.inputValue}"
                           label="#{bindings.first.hints.label}"
                           id="inputText5"/>
            <amx:inputText value="#{bindings.last.inputValue}"
                           label="#{bindings.last.hints.label}"
                           id="inputText6"/>
```

```
         </amx:panelFormLayout>
      </amx:validationGroup>
   </amx:panelGroupLayout>
</amx:panelPage>
```

Example 8–90 shows how to define a validation message displayed in a popup in an ADF Mobile AMX file.

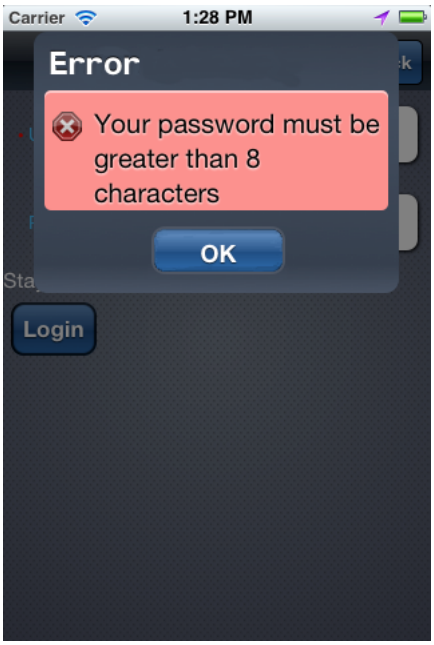
Example 8–90 Defining Input Validation with Popup Message

```
<amx:panelPage id="pp1">
   <amx:facet name="header">
      <amx:outputText id="outputText1" value="Login Demo"/>
   </amx:facet>
   <amx:facet name="secondary">
      <amx:commandButton id="btnBack" action="__back" text="Back"/>
   </amx:facet>
   <amx:panelGroupLayout id="panelGroupLayout1">
      <amx:validationGroup id="group1">
         <amx:panelGroupLayout id="panelGroupLayout2">
            <amx:inputText value="#{bindings.userName.inputValue}"
                           label="#{bindings.userName.hints.label}"
                           id="inputText1"
                           showRequired="true"
                           required="true"/>
            <amx:inputText value="#{bindings.password.inputValue}"
                           label="#{bindings.password.hints.label}"
                           id="inputText2"
                           required="true"
                           showRequired="true"
                           secret="true"/>
            <amx:outputText id="outputText2"
                            value="#{bindings.timeToStayLoggedIn.hints.label}:
                            #{bindings.timeToStayLoggedIn.inputValue} minutes"/>
        </amx:panelGroupLayout>
      </amx:validationGroup>
      <amx:commandButton id="commandButton2"
                         text="Login"
                         action="navigationSuccess">
         <amx:validationBehavior group="group1"/>
      </amx:commandButton>
   </amx:panelGroupLayout>
</amx:panelPage>
```

Validation messages are displayed in a Popup component (see Section 8.2.8, "How to Use a Popup Component"). You cannot configure the title of a validation popup, which is automatically determined by the relative message severity: the most severe of all of the current messages becomes the title of the validation popup. That is, if all validation messages are of type `WARNING`, then the title is "Warning"; if some of the messages are of type `WARNING` and others are of type `ERROR`, then the title is set to "Error".

Figure 8–76 shows a popup validation message produced at run time.

Figure 8–76 Validation Message on iPhone



8.10 Using Event Listeners

To invoke Java code from your ADF Mobile AMX pages and perform the application logic, you define listeners as attributes of UI components in one of the following ways:

- Manually in the source of your ADF Mobile AMX file.
- From the **Property** editor of the selected component. For more information, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*

You may use the following listeners to add awareness of the UI-triggered events to your ADF Mobile AMX page:

- `valueChangeListener`: listens to `ValueChangeEvent` that is constructed with the following parameters:
  - `java.lang.Object` representing an old value
  - `java.lang.Object` representing a new changed value
- `actionListener`: listens to `ActionEvent` that is constructed without parameters;
- `selectionListener`: listens to `SelectionEvent` that is constructed with the following parameters:
  - `java.lang.Object` representing an old row key
  - `java.lang.String[]` representing selected row keys
- `moveListener`: listens to `MoveEvent` that is constructed with the following parameters: of the `RowKey` type representing an old row key;
  - `java.lang.Object` representing the moved row key
  - `java.lang.String[]` representing the row key before which the moved row key was inserted
- `rangeChangeListener`: listens to `RangeChangeEvent` that is constructed with the following parameters:

- `int` representing an old start range
- `int` representing an old end range
- `int` representing a new start range
- `int` representing a new end range

The value for your listener must match the pattern `#{*}` and conform to the following requirements:

- Type name: EL Expression
- Base type: string
- Primitive type: string

For information on EL events, see Section 9.2.3, "EL Events."

Most ADF Mobile AMX event classes extend the `oracle.adfmf.amx.event.AMXEvent` class. When defining event listeners in your Java code, you need to pass the `oracle.adfmf.amx.event.AMXEvent` class.

For more information, see the following:

- *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile*
- *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*

ADF Mobile allows you to create managed bean methods for listeners so that your managed bean methods use ADF Mobile AMX-specific event classes. Example 8–91, Example 8–92, and Example 8–93 demonstrate a Button and a Link component calling the same managed bean method. The source value of the `AMXEvent` determines which object invoked the event by showing a message box with the component's ID.

Example 8–91 Calling a Bean Method from ADF Mobile AMX File

```
<amx:commandButton text="commandButton1" id="commandButton1"
   actionListener="#{applicationScope.Bean.actionListenerMethod}">
</amx:commandButton>
<amx:commandLink text="commandLink1" id="commandLink1"
   actionListener="#{applicationScope.Bean.actionListenerMethod}">
</amx:commandLink>
```

Example 8–92 Using AMXEvent

```
private void actionListenerMethod(AMXEvent amxEvent) {
   Component source = (Component) amxEvent.getSource();
   MessageBox.show("actionListener called for " + source.getId());
}
```

Example 8–93 Invoking the Event Method

```
public Object invokeMethod(String methodName, Object[] params) {
   if (methodName.equals("actionListenerMethod")) {
      actionListenerMethod((AMXEvent) params[0]);
   }
   return null;
}
```

For additional examples, see an ADF Mobile sample application called JavaDemo located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on

your development computer. This sample demonstrates how to call listeners from Java beans.

8.10.1 What You May Need to Know About Constrained Type Attributes for Event Listeners

You can define event listeners as children of some ADF Mobile AMX UI components. The listeners' `type` attribute identifies which event they are to be registered to handle. Since each parent UI component supports only a subset of the events (suitable for that particular component), these supported events are presented in a constrained list of types that you can select for a listener.

Table 8–14 lists parent UI components, event listeners they can have as children, and event types they support.

Table 8–14 Supported Event Listeners and Event Types

UI Component (parent)	Action Listener (child)	Set Property Listener (child)	Show Popup Behavior (child)	Close Popup Behavior (child)	actionListener attribute	valueChangeListener attribute	moveListener attribute	selectionListener attribute
Button	Supported	Supported	Supported	Supported	Supported	Not supported	Not supported	Not supported
Link	Supported	Supported	Supported	Supported	Supported	Not supported	Not supported	Not supported
List Item	Supported	Supported	Supported	Supported	Supported	Not supported	Not supported	Not supported
Input Date	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Input Number Slider	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Input Text	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
List View	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported
Checkbox	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Switch	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Checkbox (Select Many)	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Choice (Select Many)	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Choice	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Select Button	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Radio Button	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported

Table 8–14 (Cont.) Supported Event Listeners and Event Types

UI Component (parent)	Action Listener (child)	Set Property Listener (child)	Show Popup Behavior (child)	Close Popup Behavior (child)	actionListener attribute	valueChangeListener attribute	moveListener attribute	selectionListener attribute
Link (Go)	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Carousel	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Carousel Item	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Image	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Area Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Bar Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Bubble Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Combo Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Horizontal Bar Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Led Gauge	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Dial Gauge	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Line Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Pie Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Scatter Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Supported
Spark Chart	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported	Not supported
Status Meter Gauge	Not supported	Not supported	Not supported	Not supported	Not supported	Supported	Not supported	Not supported
Geographic Map	Not supported	Supported[1]	Not supported	Not supported	Not supported	Not supported	Not supported	Supported[2]
Thematic Map	Not supported	Supported[3]	Not supported	Not supported	Not supported	Not supported	Not supported	Supported[4]

[1] The Set Property Listener can be specified as a child of the Geographic Map's Marker of Area.

[2] The selectionListener attribute can be set on the Geographic Map's Area Data Layer or Point Data Layer.

[3] The Set Property Listener can be specified as a child of the Thematic Map's Marker of Area.

[4] The selectionListener attribute can be set on the Thematic Map's Area Data Layer or Point Data Layer.

The `type` attribute (see Figure 8–77) of each of the child event listeners has a base set of values that match the listener events. These values are filtered based on the

information presented in Table 8–14 such that when the child event listener is within the context of the identified parent UI component, only the events that the parent supports are shown. For example, under a Button component, the Action Listener or Set Property Listener child would show only the `action` Type value, as well as gestures.

Figure 8–77 shows values available in the constrained Type list of the Set Property Listener for a parent List Item component.

Figure 8–77 Selecting Event Type

9

Using Bindings and Creating Data Controls

This chapter describes how to use data bindings, data controls, and the ADF data binding expression language (EL) in ADF Mobile applications. This chapter also covers validation and data change events.

This chapter includes the following sections:

9.1 Introduction to Binding Layer Components and Data Controls

ADF Model implements two concepts that enable the decoupling of the user interface technology from the business service implementation: *data controls* and *declarative bindings*. Data controls abstract the implementation technology of a business service by using standard metadata interfaces to describe the service's operations and data collections, including information about the properties, methods, and types involved. Using JDeveloper, you can view that information as icons that you can drag and drop onto a page. Declarative bindings abstract the details of accessing data from data collections in a data control and invoking its operations. At runtime, the ADF Model layer reads the information describing the data controls and bindings from the appropriate XML files and then implements the two-way connection between the user interface and the business service.

The group of bindings supporting the UI components on a page are described in a page-specific XML file called the page definition file. The ADF Model layer uses this file at runtime to instantiate the page's bindings. These bindings are held in a request-scoped map called the binding container, accessible during each page request using the EL expression `#{bindings}`. This expression always evaluates to the binding container for the current page. You can design a databound user interface by dragging an item from the Data Controls panel and dropping it on a page as a specific UI component. When you use data controls to create a UI component, JDeveloper automatically creates the code and objects needed to bind the component to the data control you selected.

In ADF Mobile, data controls behave similarly to the way they work in Oracle ADF. The DeviceFeatures data control appears within the Data Controls panel in JDeveloper, allowing you to drag and drop the primary data attributes of data controls to your application as (text) fields, and the operations of data controls as command objects (buttons). These drag and drop actions will generate EL bindings in your application in the appropriate properties for the controls that are created. The normal ADF bindings for those actions (represented by a general `DataControls.dcx` file, to point at the data control source, and the page bindings, to link the specific page's reference to the data control) will be present, allowing the runtime to process the bindings when your application executes.

For more information on data binding and data controls, see the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* and the *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

9.2 Understanding EL Support

ADF Mobile provides support for the use of the Expression Language (EL) in its ADF Mobile AMX application feature.

You use the EL to enable data binding. For an overview of the use of EL with Oracle ADF, see the following:

- "Using ADF Model in a Fusion Web Application" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- "Creating ADF Data Binding EL Expressions" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

ADF objects exist within different scopes, such as the application scope, session scope, page flow scope, and so on (see "About Object Scope Lifecycles" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*). ADF Mobile, on the other hand, supports the following scopes:

- Application scope
- Page flow scope
- Preference scope
- View scope
- Device scope

EL expressions defined in the application scope namespace are available for the life of the application. With ADF Mobile, you can define an application scope in one view of an application, and then reference it in another.

9.2.1 ADF Mobile AMX EL Implementation

The ADF Mobile AMX EL implementation is based on the Java Unified Expression Language (JUEL) project and follows *Expression Language Specification Version 2.1* (available from the JUEL project page at `http://juel.sourceforge.net/`, and referred to hereinafter as "the specification"), with the following exceptions:

- Immediate and Deferred Evaluation
- Enumerated Types

9.2.1.1 Immediate and Deferred Evaluation

As described in "1.2.1: Eval-expression" in the specification, expressions may be evaluated immediately or deferred. In the ADF Mobile AMX EL implementation, expressions are parsed when the page metadata is loaded, at which point the owning component holds on to a reference to the parsed object. The expression is not actually evaluated until the component needs it for rendering a value. Because ADF Mobile AMX supports only the deferred semantics, an expression using the immediate construction expression (`"${}"`) still parses, but behaves the same as a deferred expression (`"#{}"`).

9.2.1.2 Enumerated Types

As described in "1.17: Enums" in the specification, using a literal string to coerce to the value of an enum type is not supported, because the required underlying enum operations are not supported on J2ME.

9.2.2 How to Reference Binding Containers

The active screen's binding container can be referenced by the root EL expression `"#{bindings}"`. Another screen's binding container can be referenced through the expression `"#{data.PageDefName}"`. The ADF Mobile AMX binding objects are referenced by name from the binding container `"#{bindings.Name}"`.

Table 9–1 lists the properties that you can use in EL expressions to access values of the ADF Mobile AMX binding objects at run time. The table lists these properties in alphabetical order.

Table 9–1 Runtime Properties

Runtime Property	Description	Iterator	Action	Attribute	Tree
`class`	Returns the Java class object for the runtime binding.	Yes	Yes	Yes	Yes
`collectionModel`	Exposes a collection of data. EL expressions used within a component that is bound to a `collectionModel` can be referenced with a `row` variable [1], which will resolve the expression for each element in the collection.	No	No	No	Yes
`currentRow`	Provides access to the current row pointed to by an iterator.	Yes	No	No	No
`dataControl`	Returns the iterator's associated data provider.	Yes	No	No	No
`dataProvider`	Available as a child of `currentRow`. Provides access to the data structure used in the iterator.	Yes	No	No	No
`enabled`	Returns `true` or `false`, depending on the state of the action binding. For example, the action binding may be enabled (`true`) or disabled (`false`) based on the currency (as determined, for example, when the user clicks the First, Next, Previous, Last navigation buttons).	No	Yes	No	No
`execute`	Invokes the named action or `methodAction` binding when resolved.	No	Yes	No	No

Table 9–1 (Cont.) Runtime Properties

Runtime Property	Description	Iterator	Action	Attribute	Tree
`hints`	Returns a list of name-value pairs for UI hints for all display attributes to which the binding is associated. The following named values are supported: ■ `format`: The format to be used for the current attribute. ■ `label`: The label to display for the current attribute. ■ `updateable`: Returns true if the current attribute can be written to.	No	No	Yes	Yes
`inputValue`	Returns or sets the value of the current attribute.	No	No	Yes	No
`items`	Returns the list of values associated with the current list enabled attribute.	Yes	No	No	No
`label`	Available as a child of `hints` or direct child of an attribute. Returns the label (if supplied by control hints) for the first attribute of the binding.	No	No	Yes	Yes
`rangeSize`	Returns the range size of the iterator binding's row set.	Yes	No	No	Yes
`rangeStart`	Returns the absolute index in a collection of the first row in range.	Yes	No	No	No
`rowCount`	Returns the total number of rows in the collection.	Yes	No	No	No
`updateable`	Available as a child of `hints` or direct child of an attribute. Returns `true` if the current attribute is updateable.	No	No	Yes	Yes
`viewable`	Available as a child of Tree. Resolves at runtime whether this binding and the associated component should be rendered or not.	No	No	No	Yes

[1] The EL term `row` is used within the context of a collection component; `row` simply acts as an iteration variable over each element in the collection whose attributes can be accessed by an ADF Mobile AMX binding object when the collection is rendered. Attribute and list bindings can be accessed through the `row` variable. The syntax for such expressions will be the same as those used for accessing binding objects outside of a collection, with the `row` variable prepended as the first term: `#{row.bindings.Name.property}`.

9.2.3 EL Events

EL events play a significant role in the functioning of the ADF Mobile AMX UI, as it enables expressions with common terms to update in sync with each other.

EL expressions can refer to values in various contexts. Example 9–1 shows the creation of two Input Number Slider components, with each component tied to an `applicationScope` value. The output text then uses EL to display a simple addition equation along with the calculated results. When the framework parses the EL expression in the output text labels, it determines that the expression contains references to two values and creates event listeners (see Section 8.10, "Using Event

Listeners") for the output text on those two values. When the value of the underlying expression changes, an event is generated to all listeners for that value.

> **Note:** If you are referencing properties on a managed bean (as opposed to scope objects) you have to add the listeners. For more information, see Section 9.2.4.2, "ADF Managed Beans."

Example 9–1 Generating EL Events with Two Components

```
<amx:inputNumberSlider id="slider1" label="X" value="#{applicationScope.X}"/>
<amx:inputNumberSlider id="slider2" label="Y" value="#{applicationScope.Y}"/>
<amx:outputText id="ot1" value="#{applicationScope.X} +
       #{applicationScope.Y} = #{applicationScope.X + applicationScope.Y}"/>
```

In Example 9–1 two components are updating one value each, and one component is consuming both values. Example 9–2 shows that the behavior would be identical if a third Input Number Slider component is added that references one of the existing values.

Example 9–2 Generating EL Events with Three Components

```
<amx:inputNumberSlider id="slider1" label="X" value="#{applicationScope.X}"/>
<amx:inputNumberSlider id="slider2" label="Y" value="#{applicationScope.Y}"/>
<amx:outputText id="ot1" value="#{applicationScope.X} +
       #{applicationScope.Y} = #{applicationScope.X + applicationScope.Y}"/>
<amx:inputNumberSlider id="slider3" label="X" value="#{applicationScope.X}"/>
```

In Example 9–2, when either Input Number Slider component updates `#{applicationScope.X}`, the other is automatically updated along with the Output Text.

9.2.3.1 Configuration Properties

Any `<adf-property>` elements in the application's `adf-config.xml` file will be exposed through EL as children of the `#{applicationScope.configuration}` node. So, for example, if `adf-config.xml` looks like the following example, evaluating `#{applicationScope.configuration.key1}` will yield `value1`, and evaluating `#{applicationScope.configuration.key2}` will yield `vaue2`.

```
<?xml version="1.0" encoding="windows-1252" ?>
<adf-config xmlns="http://xmlns.oracle.com/adf/config"
xmlns:config="http://xmlns.oracle.com/bc4j/configuration"
            xmlns:sec="http://xmlns.oracle.com/adf/security/config">
  <adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/properties">
    <adf-property name="key1" value="value1"/>
    <adf-property name="key2" value="value2"/>
  </adf-properties-child>
</adf-config>
```

In addition, `#{applicationScope.configuration.accessibilityEnabled}` evaluates to `true` if accessibility mode is enabled on the device. On iOS devices, this means that `VoiceOver` is activated. On Android devices, this means `TalkBack` is activated.

9.2.4 EL Expression Builder

You can use the JDeveloper Expression Builder to create EL expressions by selecting values from variables and operators. The Expression Builder can be invoked from the Property Inspector for any EL-enabled property. For more information about using the

Expression Builder, see "How to Create an ADF Data Binding EL Expression" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

There are two EL types in Oracle ADF:

- Dynamic EL: `${EL expression}`
- Deferred EL: `#{EL expression}`

Since ADF Mobile does not support dynamic EL type and treats `${EL expression}` as `#{EL expression}`, you should use the hash sign (#) prefix when defining expressions. For more information, see Section 9.2.1.1, "Immediate and Deferred Evaluation."

The following categories are available in the Expression Builder for the ADF Mobile AMX pages:

- ADF Bindings
- ADF Managed Beans
- ADF Mobile Objects

9.2.4.1 ADF Bindings

This section lists the options available under the ADF Bindings category. The bindings and data nodes display the same set of supported bindings and properties. Table 9–2 lists available binding types along with the properties that are supported for each binding type.

- `bindings`

  Table 9–2 lists the available binding types along with the properties that are supported for each binding type.

- `data`

  Table 9–2 lists the available binding types along with the properties that are supported for each binding type.

- `securityContext`

  Supported properties:

  - `authenticated`
  - `userGrantedPrivilege`
  - `userInRole`
  - `userName`

Table 9–2 Supported Binding Types

Binding Type	Properties

`accessorIterator`	`class`
	`currentRow: dataProvider`
	`name`
	`rangeSize`

Table 9–2 (Cont.) Supported Binding Types

Binding Type	Properties
action	class
	enabled
	execute
	name
attributeValues	class
	format
	hints: format, label, updateable
	inputValue
	items
	iteratorBinding
	label
	name
	updateable
button	class
	format
	hints: format, label, updateable
	inputValue
	items
	label
	name
	updateable
invokeAction	always
	deferred
iterator	class
	currentRow: dataProvider
	name
	rangeSize
list	class
	format
	hints: format, label, updateable
	inputValue
	items
	label
	name
	updateable

Table 9–2 (Cont.) Supported Binding Types

Binding Type	Properties
methodAction	class
	enabled
	execute
	name
	operationInfo
	paramsMap
	result
methodIterator	class
	currentRow: dataProvider
	name
	rangeSize
tree	class
	collectionModel: <AttrName>
	hints: format, label, updateable, <AttrName>
	iteratorBinding
	name
	rangeSize
	viewable
variableIterator	class
	currentRow: dataProvider
	name

9.2.4.2 ADF Managed Beans

You can create and use managed beans in an ADF Mobile application to store additional data or to execute custom code. Adding a managed bean to an ADF Mobile application is done the same way as adding one to a Fusion Web Application.

For more information, see the following:

- "Using a Managed Bean in a Fusion Web Application" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
- JavaDemo, an ADF Mobile sample application located in the *jdev_install*/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples directory. This sample application shows how to use Java beans and invoke container utility methods. For more information, see Appendix E, "ADF Mobile Sample Applications."

> **Note:** Carefully consider the binding styles you use when configuring components. More specifically, combining standard bindings with managed bean bindings will frequently result in misunderstood behaviors because the class instances are unlikely to be the same between the binding infrastructure and the managed bean infrastructure. If you mix bindings, you may end up calling behavior on an instance that isn't directly linked to the UI.

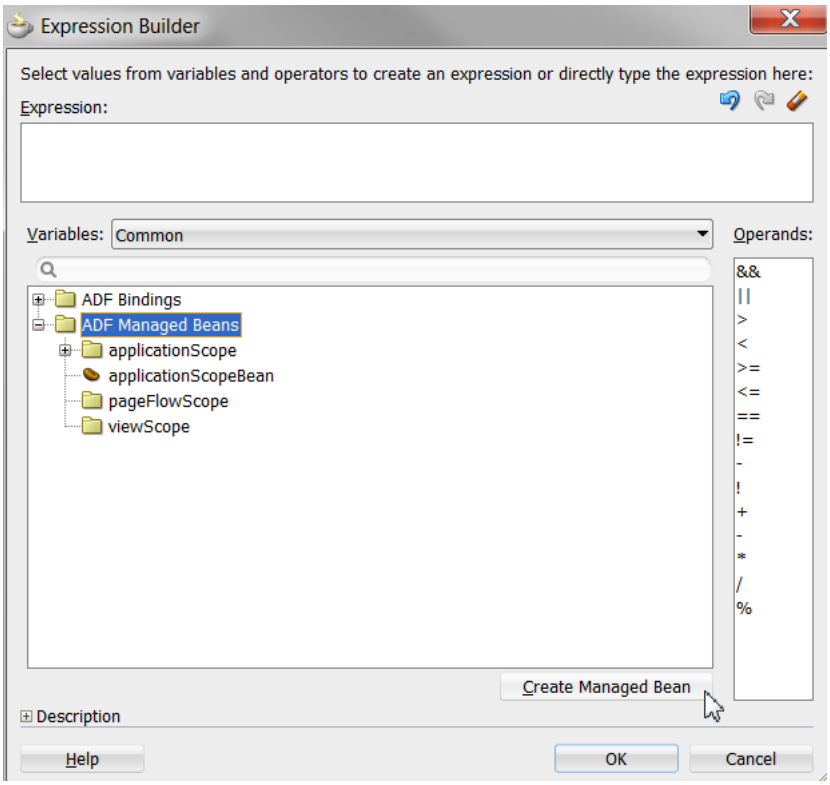
Supported ADF Mobile managed beans scopes are as follows:

- `applicationScope`: ADF Managed Beans > `applicationScope` node contains everything that is defined at the application level (for example, application-scoped managed beans).
- `pageFlowScope`: ADF Managed Beans > `pageFlowScope` node contains everything that is defined at the page flow level (for example, page flow-scoped managed beans).
- `viewScope`: ADF Managed Beans > `viewScope` node contains everything that is defined at the view level (for example, view-scoped managed beans).

When the ADF Managed Beans category is selected in the Expression Builder, the Create Managed Bean button appears. Click this button to create Managed Beans directly from the Expression Builder.

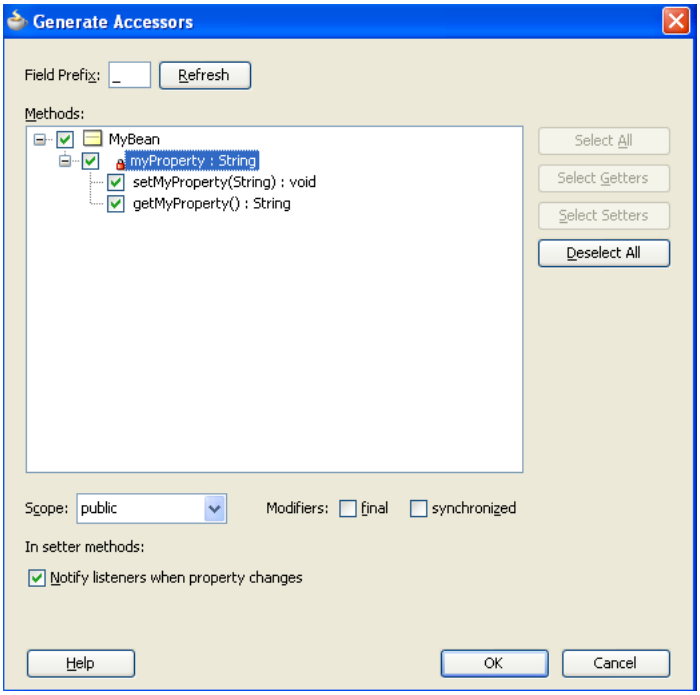
Figure 9–1 shows an example of the ADF Mobile Managed Beans contents and the Create Managed Bean button.

Figure 9–1 ADF Managed Bean Contents



The ADF Mobile runtime will register itself as a listener on managed bean property change notifications so that EL expressions bound to UI components that reference bean properties will update automatically if the value of the property changes. Sourcing these notifications requires some additional code in the beans' property accessors. To automatically generate the necessary code to source notifications from your beans' property accessors, select the **Notify listeners when property changes** checkbox in the Generate Accessors dialog (see Figure 9–2).

Figure 9–2 Notify Listeners When Property Changes



It is not necessary to add this code to simply reference bean methods or properties through EL, but it is necessary to keep the rendering of any EL expressions in the active form that depend on values stored in the bean current if those values change, especially if the change is indirect, such as a side effect of executing a bean method that changes one or more property values. For information about property changes and the `PropertyChangeSupport` class, see Section 9.7, "Data Change Events."

Note: If you declare a managed bean within the `applicationScope` of a feature but then try to reference that bean through EL in another feature at design time, you will see a warning in the design time about invalid EL. This warning is due to the fact that the design time cannot find a reference in the current project for that bean. You can reference that bean at runtime only if you first visit the initial feature where you declared the bean and the bean is instantiated before you access it through EL in another feature. This is not the case for the `PreferenceValue` element as it uses the `Name` attribute value as the node label.

9.2.4.3 ADF Mobile Objects

The following options are available under the ADF Mobile Objects category:

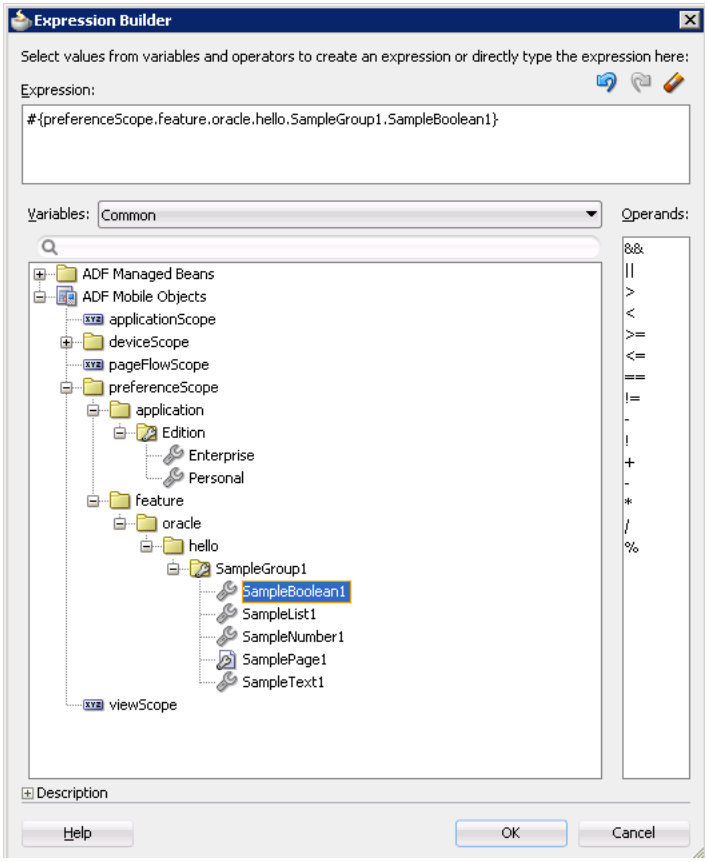
- `applicationScope`: The `applicationScope` node contains everything that is defined at the application level (for example, application-scoped managed beans).
- `deviceScope`: The `deviceScope` node contains everything that is defined at the device level (for example, device-scoped managed beans). ADF Mobile supports the following `deviceScope` properties:
  - `device`
    - `model`

    - name
    - os
    - phonegap
    - platform
    - version
  - hardware
    - hasAccelerometer
    - hasCamera
    - hasCompass
    - hasContacts
    - hasFileAccess
    - hasGeolocation
    - hasLocalStorage
    - hasMediaPlayer
    - hasMediaRecorder
    - hasTouchScreen
    - networkStatus
    - screen (availableHeight, availableWidth, height, width)

- pageFlowScope: The pageFlowScope node contains everything that is defined at the page flow level (for example, page flow-scoped managed beans).
- preferenceScope: The preferenceScope node contains all the application and feature preferences.
  - Application preferences are available under ADF Mobile Objects > preferenceScope > application.
  - Feature preferences are available under ADF Mobile Objects > preferenceScope > feature > *featureId*

  Figure 9–3 shows an example of preference elements under the preferenceScope node.

Figure 9–3 Preference Elements Under the preferenceScope Node



Preference elements use the `Id` attribute value as the node label in the Expression Builder, except for the `PreferenceValue` element. The `PreferenceValue` element uses the `Name` attribute value as the node label in the Expression Builder.

Note: Where string tokens in EL expressions contain a dot (".") or any special character, or a reserved word like `default`, the Expression Builder surrounds such string tokens with a single quote and bracket. When the feature ID or preference component ID contains a dot, the Expression Builder displays each part of the ID that is separated by a dot as a separate property in the `preferenceScope` hierarchy. The expression generated also takes each part of the ID separated by a dot as a separate property.

Following are some sample `preferenceScope` EL expressions:

Example 9–3 Feature ID Containing "."

```
"#{preferenceScope.feature.oracle.hello.SampleGroup1.label}"
```

Example 9–4 Attribute Name Is a Reserved Word

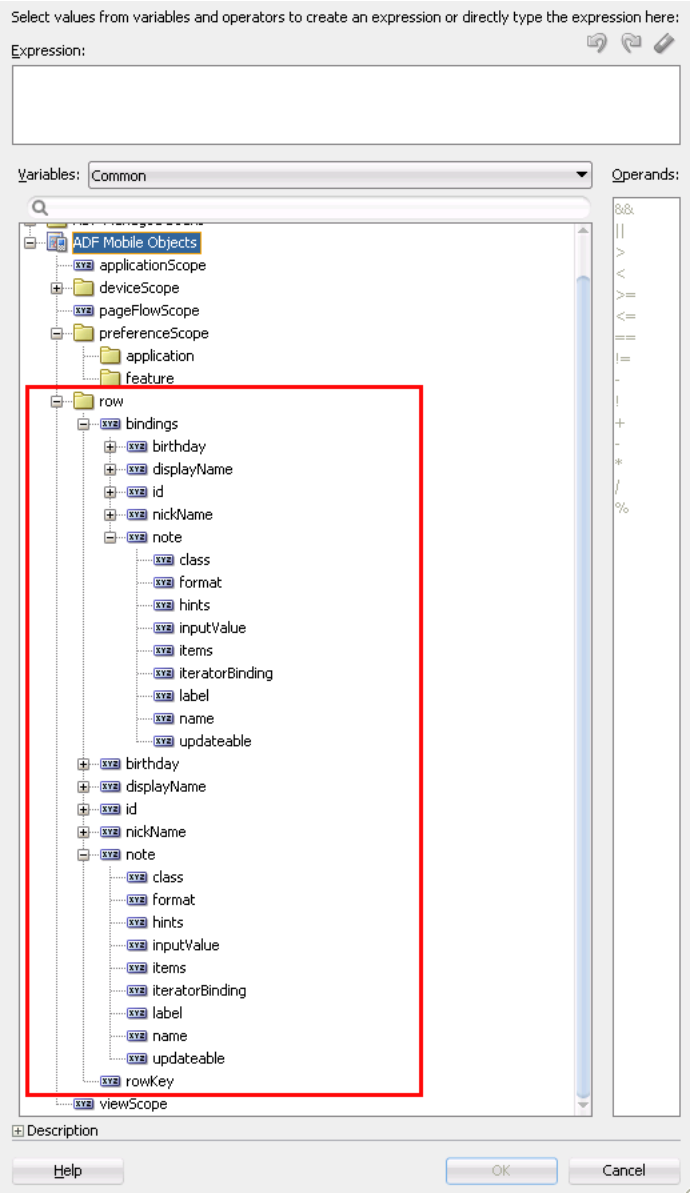
```
"#{preferenceScope.application.OracleMobileApp.Edition['default']}"
```

- `viewScope`: This node contains everything that is defined at the view level (for example, view-scoped managed beans).

- `row`: The `row` variable node is a shortcut to the `collectionModel`. Its name is the value of the `var` attribute of the parent Iterator, List View, or Carousel component. This node supports the following directory structure:
  - bindings
    - *AttrName*
    - `class`
    - `format`
    - `hints`
    - `inputValue`
    - `items`
    - `iteratorBinding`
    - `label`
    - `name`
    - `updateable`
  - *AttrName*
    - `class`
    - `format`
    - `hints`
    - `inputValue`
    - `items`
    - `iteratorBinding`
    - `label`
    - `name`
    - `updateable`
  - `rowKey`

  Figure 9–4 shows an example of the `row` variable node contents.

Figure 9–4 Row Variable Contents



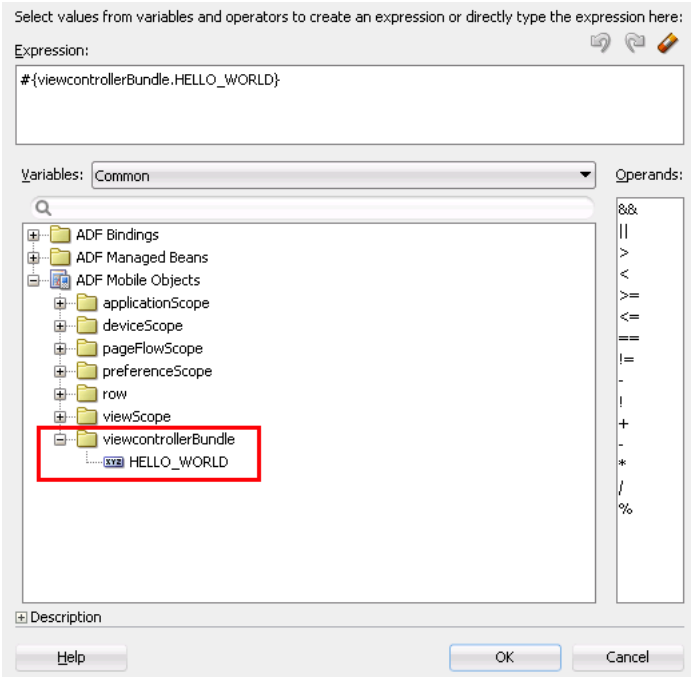
- `viewControllerBundle`

  This is the name of the resource bundle variable that points to a resource bundle defined at the project level.

  This node is shown only after the `amx:loadBundle` element has been dropped and a resource bundle has been created. The name of this node will vary as it depends on the variable name of amx:loadBundle. This node will display all strings declared in the bundle.

  Figure 9–5 shows an example of the contents of the `viewControllerBundle` node. Example 9–5 shows an example of AMX code for `viewControllerBundle`.

Figure 9–5 Contents of the viewControllerBundle Node



Example 9–5 AMX Code Sample of the loadBundle Element

```
<amx:loadBundle basename="mobile.ViewControllerBundle"
var="viewcontrollerBundle"/>
```

9.2.4.4 Method Expression Builder

Table 9–3 shows properties that have the Method Expression Builder option available in the Property Inspector instead of the Expression Builder option. The only difference between them is that the Method Expression Builder filters out the managed beans depending on the selected property.

Table 9–3 Editable Attributes

Property	Element
binding	actionListener
action	commandButton
actionListener	commandButton
action	commandLink
actionListener	commandLink
action	listItem
actionListener	listItem
valueChangeListener	inputDate
valueChangeListener	inputNumberSlider
valueChangeListener	inputText
valueChangeListener	selectBooleanCheckbox
valueChangeListener	selectBooleanSwitch

Table 9–3 (Cont.) Editable Attributes

Property	Element
valueChangeListener	selectManyCheckbox
valueChangeListener	selectManyChoice
valueChangeListener	selectOneButton
valueChangeListener	selectOneChoice
valueChangeListener	selectOneRadio
moveListener	listView

9.2.4.5 Non EL-Properties

Table 9–4 shows the properties that do not have the EL Expression Builder option available in the Property Inspector, because they are not EL-enabled.

Table 9–4 Non EL-Properties

Property	Element
name	facet
var	iterator
var	listView
var	loadBundle
group	validationBehavior

9.3 Understanding Binding Layer Components

The ADF Mobile runtime uses the `default` behavior (illustrated in Example 9–6) to invoke the action for a particular operation, because it must be called on repeat showings of the page after it has been loaded (not only for the initial load).

Example 9–6 invokeAction in ADF Mobile AMX

```
<invokeAction id="callSetCurrentRowWithKeyValue"
              Binds="setCurrentRowWithKeyValue"
              Refresh="default"
              RefreshCondition="#{requestScope.partyId != null}"/>
```

For information on the recognized values of the `Refresh` attribute and their meaning, see Table 9–5, " Refresh Values and the Corresponding Conditions to Invoke".

The ADF Mobile runtime can detect the following conditions for issuing notifications to invoke executables:

- A: Each time a page is shown.
- B: On any action that changes the state of the model (like setting an attribute value or navigating to a new row).

The runtime executes an `invokeAction` on the above conditions according to values of its `Refresh` and `RefreshCondition` attributes, as listed in Table 9–5, " Refresh Values and the Corresponding Conditions to Invoke". For more information, see "What You May Need to Know About Using the Refresh Property Correctly" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Table 9–5 Refresh Values and the Corresponding Conditions to Invoke

Refresh Value	Condition to Invoke
`always` - Fire the `invokeAction` every time a page is navigated to regardless of the state of the binding container. In terms of the runtime lifecycle, this will be on every `setContext` message.	A and B
`deferred` - Fire the `invokeAction` when a page is navigated to and its bindings are loaded and initialized. In terms of the runtime lifecycle, this will be on every `setContext` message that initializes a binding container. If subsequent navigations to the page do not reload the bindings, then the action will not fire. This is the default behavior.	A
`default` - Always set to `deferred` by default.	A

If a `RefreshCondition` expression is supplied, it will be evaluated at each potential execution of the `invokeAction`. If it evaluates to `false`, execution is skipped on that occurrence.

Note: For iterator executables, `deferred` is the standard behavior: the iterator does not perform its initial `Refresh` until an EL expression that is dependent on it is evaluated. Any `invokeAction` executable in a page definition file must have a value other than the default (`deferred`) for its `refresh` attribute, or it will not be refreshed and invoked.

For more information on the use of bindings in ADF Mobile, see the following:

- Section 7.3.2.4.4, "What You May Need to Know About Generated Bindings"
- Section 7.3.2.4.6, "Using the ADF Mobile AMX Editor Bindings Tab"
- Section 7.3.2.4.7, "What You May Need to Know About Removal of Unused Bindings"

9.3.1 Sequencing

By default, `invokeActions` declared with the same `Refresh` value execute in the order they are declared in the `pagedef`.

Note: The predecessor still executes or not based on its own `Refresh` and `RefreshCondition` values.

9.3.2 Validation of EL Bindings for ADF Mobile AMX Pages

In ADF Mobile, validation of EL bindings for the ADF Mobile AMX pages is done using the same mechanism for validating EL for JSF pages, behaving similarly to Oracle ADF. The EL constructs are validated against the page bindings and method references are validated against the managed beans.

For more information on validation, see Section 9.6, "Performing Validation."

9.4 Creating and Using the Bean Data Control

Java bean data controls obtain their data structure from POJOs (plain old Java objects). To create a Java bean data control, right-click a Java class file (in the Applications

window), and choose Create Data Control. Creating bean data controls is very similar to creating EJB data controls.

> **Note:** If the Java bean is using a background thread to update data in the UI, you need to manually call `oracle.adfmf.framework.api.AdfmfJavaUtilities.flushDataChangeEvent`. For information about the `flushDataChangeEvent` method, see Section 9.7, "Data Change Events."

For more information, see the "Data Controls in Fusion Web Applications" section in the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* and the "Exposing Business Services with ADF Data Controls" section in the *Oracle Fusion Middleware Java EE Developer's Guide for Oracle Application Development Framework*.

9.4.1 What You May Need to Know About Serialization of Bean Class Variables

ADF Mobile does not serialize to JavaScript Object Notation (JSON) data bean class variables that are declared as transient. If you want to avoid serialization of a chain of nested objects, you should define them as transient. This strategy also helps to prevent the creation of cyclic objects due to object nesting.

Consider the following scenario: you have an `Employee` object that, in turn, has a child `Employee` object representing the employee's manager. If you do not declare the child object transient, a chain of serialized nested objects will be created when you attempt to calculate the child `Employee` object at runtime.

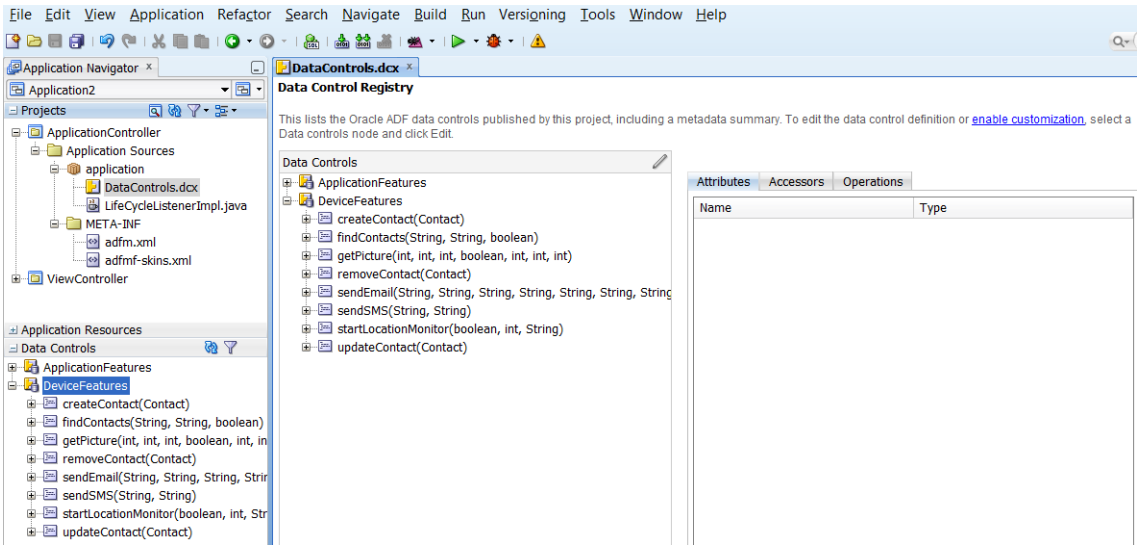
ADF Mobile does not support serializing objects of the `GregorianCalendar` class. The `JSONBeanSerializationHelper` class cannot serialize objects of the `GregorianCalendar` class because the `GregorianCalendar` class has cyclical references in it. Instead, use `java.util.Date or java.sql.Date` for date manipulation. The following example shows how to convert a `GregorianCalendar` object using `java.util.Date`:

```
Calendar calDate = new GregorianCalendar();
calDate.set(1985, 12, 1); // "January 1, 1986"
Date date = calDate.getTime();
```

9.5 Using the DeviceFeatures Data Control

ADF Mobile exposes device-specific features that you can use in your application through the DeviceFeaturers data control, a component that appears in the Data Controls panel when you create a new ADF Mobile application. The PhoneGap Java API is abstracted through this data control, enabling the application features implemented as ADF Mobile AMX to access various services embedded on the device. By dragging and dropping the operations provided by the DeviceFeatures data control into an ADF Mobile AMX page, you can add functions to manage the user contacts stored on the device, create and send both email and SMS text messages, ascertain the location of the device, use the device's camera, and retrieve images stored in the device's file system. The following sections describe each of these operations in detail, including how to use them declaratively and how to implement them with Java code and JavaScript.

Figure 9–6 ADF Mobile DeviceFeatures Data Control in the Overview Editor



The DeviceFeatures data control appears in the Data Controls panel automatically when you create an application using the ADF Mobile application template. Figure 9–6 shows the DeviceFeatures data control appears in the overview editor. The following methods are available:

- `createContact`
- `findContacts`
- `getPicture`
- `removeContact`
- `sendEmail`
- `sendSMS`
- `startLocationMonitor`
- `updateContact`

After you create a page, you can drag DeviceFeatures data control methods (or other objects nested within those methods) from the Data Controls panel to an ADF Mobile AMX view to create command buttons and other components that are bound to the associated functionality. You can accept the default bindings or modify the bindings using EL. You can also use JavaScript or Java to implement or configure functionality.

The `DeviceManager` is the object that enables you to access device functionality. You can get a handle on this object by calling `DeviceManagerFactory.getDeviceManager`. The following sections describe how you can invoke methods like `getPicture` or `createContact` using the `DeviceManager` object.

For information on how to include data controls in your ADF Mobile application, see Section 7.3.2.4, "Adding Data Controls to the View."

9.5.1 How to Use the getPicture Method

The DeviceFeatures data control includes the `getPicture` method, which enables ADF Mobile applications to leverage the device's camera and photo library so end users can take a photo or retrieve an existing image. Example 9–7 shows JavaScript code that allows an end user to take a picture with the device's camera. Example 9–8 and

Example 9–9 show Java code that will allow an end user to take a picture or retrieve a saved image. For information about the `getPicture` method, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap documentation (see `http://www.phonegap.com/home`).

The following parameters control where the image is taken from and how it is returned:

- `quality`: Set the quality of the saved image. Range is 0 to 100, inclusive. A higher number indicates higher quality, but also increases the file size. Only applicable to JPEG images (specified by `encodingType`).
- `destinationType`: Choose the format of the return value:
  - `DeviceManager.CAMERA_DESTINATIONTYPE_DATA_URL (0)`—Returns the image as a Base64-encoded string. This value is also specified as an enum using `DeviceManager.CAMERA_DESTINATION_DATA_URL` when used programmatically.
  - `DeviceManager.CAMERA_DESTINATIONTYPE_FILE_URI (1)`—Returns the image file path. This value is also specified as an enum using DeviceManager.`CAMERA_DESTINATION_FILE_URI` when used programmatically.
- `sourceType`: Set the source of the picture:
  - `DeviceManager.CAMERA_SOURCETYPE_PHOTOLIBRARY (0)`—Allows the user to choose from a previously saved image. This value is also specified as an enum using `DeviceManager.CAMERA_SOURCETYPE_PHOTOLIBRARY` when used programmatically.
  - `DeviceManager.CAMERA_SOURCETYPE_CAMERA (1)`—Allows the user to take a picture with device's camera. This value is also specified as an enum using `DeviceManager.CAMERA_SOURCETYPE_CAMERA` when used programmatically.
  - `DeviceManager.CAMERA_SOURCETYPE_SAVEDPHOTOALBUM (2)`—Allows the user to choose from an existing photo album. This value is also specified as an enum using `DeviceManager.CAMERA_SOURCETYPE_SAVEDPHOTOALBUM` when used programmatically.
- `allowEdit`: Choose whether to allow simple editing of the image before selection (boolean).
- `encodingType`: Choose the encoding of the returned image file:
  - `DeviceManager.CAMERA_ENCODINGTYPE_JPEG` (0)—Encodes the returned image as a JPEG file. This value is also specified as an enum using `DeviceManager.CAMERA_ENCODINGTYPE_JPEG` when used programmatically.
  - `DeviceManager.CAMERA_ENCODINGTYPE_PNG` (1)—Encodes the returned image as a PNG file. This value is also specified as an enum using `DeviceManager.CAMERA_ENCODINGTYPE_PNG` when used programmatically.
- `targetWidth`: Set the width in pixels to scale the image. Aspect ratio is maintained. A negative or zero value indicates that the original dimensions of the image will be used.
- `targetHeight`: Set the height in pixels to scale the image. Aspect ratio is maintained. A negative or zero value indicates that the original dimensions of the image will be used.

To customize a getPicture operation using the DeviceFeatures data control:

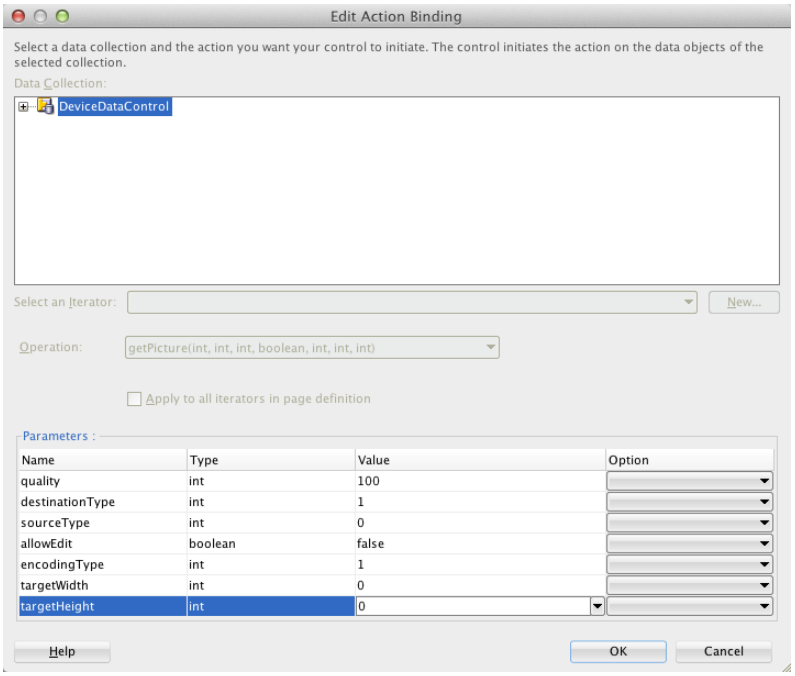
1. Drag the **`getPicture`** operation from the DeviceFeatures data control in the Data Controls panel and drop it on the page as an ADF Mobile **Button**.

If you want to provide more control to the user, drop the **`getPicture`** operation as an ADF Mobile **Parameter Form**. This allows the end user to specify settings before taking a picture or choosing an existing image.

2. In the Edit Action dialog, set the values for all parameters described above. Be sure to specify `destinationType = 1` so that the image is returned as a filename.
3. Drag the return value of **`getPicture`** and drop it on the page as an **Output Text**.
4. From the Common Components panel, drag an **Image** from the Component Palette and drop it on the page.
5. Set the `source` attribute of the Image to the return value of the `getPicture` operation. The bindings expression should be: `#{bindings.Return.inputValue}`.

Figure 9–7 shows the bindings for displaying an image from the end user's photo library:

Figure 9–7 Bindings for Displaying an Image from the Photo Library at Design Time



When this application is run, the image chooser will automatically be displayed and the end user can select an image to display. The image chooser is displayed automatically because the Image control is bound to the return value of the `getPicture` operation, which in turn causes the `getPicture` operation to be invoked.

Keep in mind the following platform-specific issues:

- iOS
  - Set quality below 50 to avoid memory error on some devices.
  - When `destinationType FILE_URI` is used, photos are saved in the application's temporary directory.
  - The contents of the application's temporary directory are deleted when the application ends. You may also delete the contents of this directory using the `navigator.fileMgr` APIs if storage space is a concern.

  - `targetWidth` and `targetHeight` must both be specified to be used. If one or both parameters have a negative or zero value, the original dimensions of the image will be used.

- Android
  - Ignores the `allowEdit` parameter.
  - `Camera.PictureSourceType.PHOTOLIBRARY` and `Camera.PictureSourceType.SAVEDPHOTOALBUM` both display the same photo album.
  - `Camera.EncodingType` is not supported. The parameter is ignored, and will always produce JPEG images.
  - `targetWidth` and `targetHeight` can be specified independently. If one parameter has a positive value and the other uses a negative or zero value to represent the original size, the positive value will be used for that dimension, and the other dimension will be scaled to maintain the original aspect ratio.
  - When `destinationType DATA_URL` is used, large images can exhaust available memory, producing an out-of-memory error, and will typically do so if the default image size is used. Set the `targetWidth` and `targetHeight` to constrain the image size.

Example 9–7 shows JavaScript code that allows the user to take a picture with the device's camera. The result will be the full path to the saved image.

Example 9–7 JavaScript Code Example for getPicture

```
// The camera, like many other device-specific features, is accessed
// from the global 'navigator' object in JavaScript.
// Note that in the PhoneGap JavaScript APIs, the parameters are passed
// in as a dictionary, so it is only necessary to provide key-value pairs
// for the parameters you want to specify.

navigator.camera.getPicture(onSuccess, onFail, { quality: 50,

function onSuccess(imageURI) {
    var image = document.getElementById('myImage');
    image.src = imageURI;
}
function onFail(message) {
    alert('Failed because: ' + message);
}
```

Example 9–8 shows Java code that allows the user to take a picture with the device's camera. The result will be the full path to the saved image.

Example 9–8 Java Code Example for Taking a Picture with getPicture

```
import oracle.adf.model.datacontrols.device;

// Access device features in Java code by acquiring an instance of the
// DeviceManager from the DeviceManagerFactory.
// Take a picture with the device's camera.
// The result will be the full path to the saved PNG image.
String imageFilename = DeviceManagerFactory.getDeviceManager().getPicture(100,
    DeviceManager.CAMERA_DESTINATIONTYPE_FILE_URI,
    DeviceManager.CAMERA_SOURCETYPE_CAMERA, false,
    DeviceManager.CAMERA_ENCODINGTYPE_PNG, 0, 0);
```

Example 9–9 shows Java code that allows the user to retrieve a previously-saved image. The result will be a base64-encoded JPEG.

Example 9–9 Java Code Example for Retrieving an Image with getPicture

```
import oracle.adf.model.datacontrols.device;

// Retrieve a previously-saved image. The result will be a base64-encoded JPEG.
String imageData = DeviceManagerFactory.getDeviceManager().getPicture(100,
  DeviceManager.CAMERA_DESTINATIONTYPE_FILE_URL,
  DeviceManager.CAMERA_SOURCETYPE__PHOTOLIBRARY, false,
  DeviceManager.CAMERA_ENCODINGTYPE_JPEG, 0, 0);
```

9.5.2 How to Use the SendSMS Method

The DeviceFeatures data control includes the `sendSMS` method, which enables ADF Mobile applications to leverage the device's Short Message Service (SMS) text messaging interface so end users can send and receive SMS messages. ADF Mobile enables you to display the device's SMS interface and optionally pre-populate the following fields:

- `to`: List recipients (comma-separated).
- `body`: Add message body.

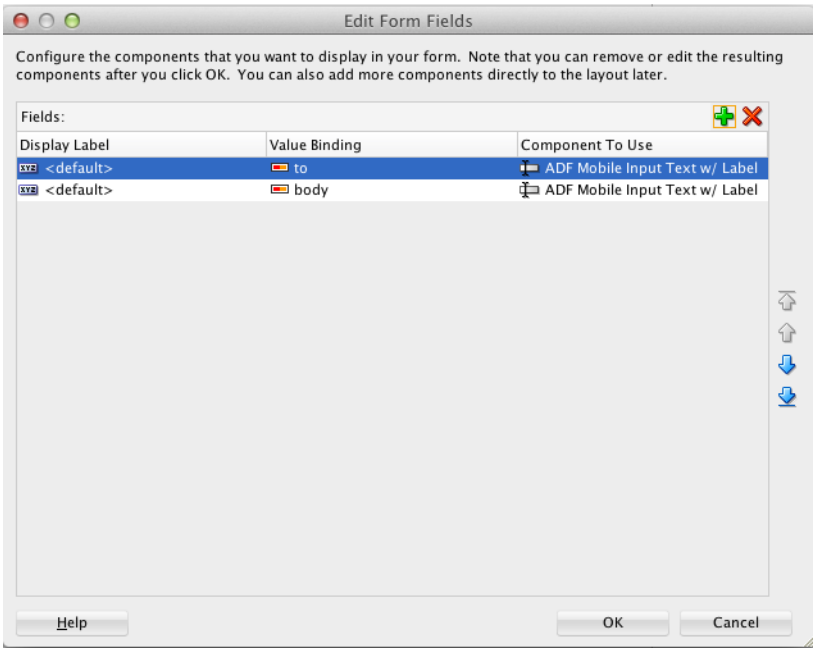
After the SMS text messaging interface is displayed, the end user can choose to either send the SMS or discard it. It is not possible to automatically send the SMS due to device and carrier restrictions; only the end user can actually send the SMS.

To customize a sendSMS operation using the DeviceFeatures data control:

To display an interactive form on the page for sending SMS, drag the `sendSMS` operation from the DeviceFeatures data control in the Data Controls panel and drop it on the page designer as an ADF Mobile **Parameter Form**. You can then customize the form in the **Edit Form Fields** dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various fields described above. Below this form will be a button to display the device's SMS interface, which will display an SMS that is ready to send with all of the specified fields pre-populated.

Figure 9–8 shows the bindings for sending an SMS using an editable form on the page.

Figure 9–8 Bindings for Sending an SMS Using an Editable Form at Design Time



Example 9–10 and Example 9–11 show code examples that allow the end user to send an SMS message with the device's text messaging interface.

For information about the `sendSMS` method, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap documentation (see `http://www.phonegap.com/home`).

Example 9–10 JavaScript Code Example for sendSMS

```
adf.mf.api.sendSMS({to: "5551234567", body: "This is a test message"});
```

Example 9–11 Java Code Example for sendSMS

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

// Access device features in Java code by acquiring an instance of the
// DeviceManager from the DeviceManagerFactory.
// Send an SMS to the phone number "5551234567"
DeviceManagerFactory.getDeviceManager().sendSMS("5551234567", "This is a test message");
```

9.5.3 How to Use the sendEmail Method

The DeviceFeatures data control includes the `sendEmail` method, which enables ADF Mobile applications to leverage the device's email messaging interface so end users can send and receive email messages. ADF Mobile enables you to display the device's email interface and optionally pre-populate the following fields:

- `to`: List recipients (comma-separated).
- `cc`: List CC recipients (comma-separated).
- `subject`: Add message subject.
- `body`: Add message body.

- `bcc`: List BCC recipients (comma-separated).
- `attachments`: List file names to attach to the email (comma-separated).
- `mimeTypes`: List MIME types to use for the attachments (comma-separated). Specify null to let ADF Mobile automatically determine the MIME types. It is also possible to specify only the MIME types for selected attachments as shown in Example 9–12 and Example 9–13.

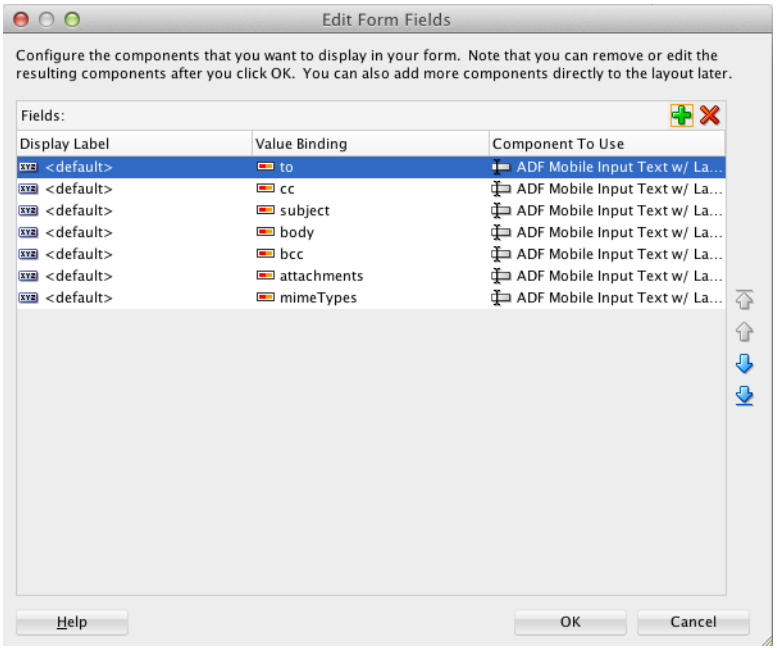
After the device's email interface is displayed, the user can choose to either send the email or discard it. It is not possible to automatically send the email due to device and carrier restrictions; only the end user can actually send the email. The device must also have at least one email account configured to send email; otherwise, an error will be displayed indicating that no email accounts could be found.

To customize a sendEmail operation using the DeviceFeatures data control:

In JDeveloper, drag the `sendEmail` operation from the DeviceFeatures data control in the Data Controls panel to the page designer and drop it as an ADF Mobile **Parameter Form**. You can then customize the form in the **Edit Form Fields** dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various fields described above. Below this form will be a button to display the device's email interface, which will display an email ready to send with all of the specified fields pre-populated.

Figure 9–9 shows the bindings for sending an email using an editable form on the page.

Figure 9–9 Bindings for Sending an Email Using an Editable Form at Design Time



Example 9–12 and Example 9–13 show code examples that allow the end user to send an email message with the device's email interface.

For information about the `sendEmail` method, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap documentation (see `http://www.phonegap.com/home`).

Example 9–12 JavaScript Code Example for sendEmail

```
// Populate an email to 'ann.li@corp.example.com',
// copy 'joe.jones@corp.example.com', with the
// subject 'Test message', and the body 'This is a test message'
// No BCC recipients or attachments
adf.mf.api.sendEmail({to: "ann.li@corp.example.com",
                          cc: "joe.jones@corp.example.com",
                          subject: "Test message",
                          body: "This is a test message"});

// Populate the same email as before, but this time, also BCC
// 'john.smith@corp.example.com' & 'jane.smith@corp.example.com' and attach two files.
// By not specifying a value for the mimeTypes parameter, you are telling
// ADFMobile to automatically determine the MIME type for each of the attachments.
adf.mf.api.sendEmail({to: "ann.li@corp.example.com",
                          cc: "joe.jones@corp.example.com",
                          subject: "Test message",
                          body: "This is a test message"});
                          bcc: "john.smith@corp.example.com,jane.smith@corp.example.com",
                          attachments: "path/to/file1.txt,path/to/file2.png"});

// For iOS only: Same as previous email, but this time, explicitly specify
// all the MIME types.
adf.mf.api.sendEmail({to: "ann.li@corp.example.com",
                          cc: "joe.jones@corp.example.com",
                          subject: "Test message",
                          body: "This is a test message"});
                          bcc: "john.smith@corp.example.com,jane.smith@corp.example.com",
                          attachments: "path/to/file1.txt,path/to/file2.png"});
                          mimeTypes: "text/plain,image/png"});

// For iOS only: Same as previous email, but this time, only specify
// the MIME type for the second attachment and let the system determine
// the MIME type for the first one.
adf.mf.api.sendEmail({to: "ann.li@corp.example.com",
                          cc: "joe.jones@corp.example.com",
                          subject: "Test message",
                          body: "This is a test message"});
                          bcc: "john.smith@corp.example.com,jane.smith@corp.example.com",
                          attachments: "path/to/file1.txt,path/to/file2.png"});
                          mimeTypes: ",image/png"});

// For Android only: Same as previous e-mail, but this time, explicitly specify
// the MIME type.
adf.mf.api.sendEmail({to: "ann.li@corp.example.com",
                          cc: "joe.jones@corp.example.com",
                          subject: "Test message",
                          body: "This is a test message"});
                          bcc: "john.smith@corp.example.com,jane.smith@corp.example.com",
                          attachments: "path/to/file1.txt,path/to/file2.png"});
                          mimeTypes: "image/*"});
// You can also use "plain/text" as the MIME type as it just determines the type
// of applications to be filtered in the application chooser dialog.
```

Example 9–13 Java Code Example for sendEmail

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

// Access device features in Java code by acquiring an instance of the
```

```
// DeviceManager from the DeviceManagerFactory.
// Populate an email to 'ann.li@corp.example.com', copy 'joe.jones@corp.example.com', with the
// subject 'Test message', and the body 'This is a test message'.
// No BCC recipients or attachments.
DeviceManagerFactory.getDeviceManager().sendEmail(
                                        "ann.li@corp.example.com",
                                        "joe.jones@corp.example.com",
                                        "Test message",
                                        "This is a test message",
                                        null,
                                        null,
                                        null);

// Populate the same email as before, but this time, also BCC
// 'john.smith@corp.example.com' & 'jane.smith@corp.example.com' and attach two files.
// By specifying null for the mimeTypes parameter, you are telling
// ADFMobile to automatically determine the MIME type for each of the attachments.
DeviceManagerFactory.getDeviceManager().sendEmail(
                                        "ann.li@corp.example.com",
                                        "joe.jones@corp.example.com",
                                        "Test message",
                                        "This is a test message",
                                        "john.smith@corp.example.com,jane.smith@corp.example.com",
                                        "path/to/file1.txt,path/to/file2.png",
                                        null);

// Same as previous email, but this time, explicitly specify all the MIME types.
DeviceManagerFactory.getDeviceManager().sendEmail(
                                        "ann.li@corp.example.com",
                                        "joe.jones@corp.example.com",
                                        "Test message",
                                        "This is a test message",
                                        "john.smith@corp.example.com,jane.smith@corp.example.com",
                                        "path/to/file1.txt,path/to/file2.png",
                                        "text/plain,image/png");


// Same as previous email, but this time, only specify the MIME type for the
// second attachment and let the system determine the MIME type for the first one.
DeviceManagerFactory.getDeviceManager().sendEmail(
                                        "ann.li@corp.example.com",
                                        "joe.jones@corp.example.com",
                                        "Test message",
                                        "This is a test message",
                                        "john.smith@corp.example.com,jane.smith@corp.example.com",
                                        "path/to/file1.txt,path/to/file2.png",
                                        ",image/png");
```

9.5.4 How to Use the createContact Method

The DeviceFeatures data control includes the `createContact` method, which enables ADF Mobile applications to leverage the device's interface and file system for managing contacts so end users can create new contacts to save in the device's address book. ADF Mobile enables you to display the device's interface and optionally pre-populate the `Contact` fields. The `createContact` method takes in a `Contact` object as a parameter and returns the created `Contact` object, as shown in Example 9–14. For information about the `createContact` method and the `Contact` object, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap

documentation (see http://www.phonegap.com/home). Also see Section 9.5.5, "How to Use the findContacts Method" for a description of Contact fields.

To customize a createContact operation using the DeviceFeatures data control:

1. In JDeveloper, drag the createContact operation from the DeviceFeatures data control in the Data Controls panel and drop it on the page designer as an ADF Mobile **Link**, **Button**, or **Parameter Form**.

   Link or Button: You will be prompted with the **Edit Action Binding** dialog to enter values for arguments to the createContact operation. At runtime, a button or link will be displayed on the page, which will use the entered values to perform a createContact operation when pressed.

   Parameter Form: Customize the form in the **Edit Form Fields** dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various Contact fields. Below this form will be a button, which will use the entered values to perform a createContact operation when pressed.

2. You can also drag a Contact return object from under the createContact operation in the Data Controls panel and drop it on to the page as an **ADF Form**. You can then customize the form in the **Edit Form Fields** dialog. When the createContact operation is performed, the results will be displayed in this form.

Example 9–14 and Example 9–15 show code examples that allow the end user to create contacts on devices.

Example 9–14 JavaScript Code Example for createContact

```
// Contacts, like many other device-specific features, are accessed from the
global 'navigator' object in JavaScript.
var contact = navigator.contacts.create();

var name = new ContactName();
name.givenName = "GivenName";
name.familyName = "FamilyName";

contact.name = name;

// Store contact phone numbers in ContactField[]
var phoneNumbers = [1];
phoneNumbers[0] = new ContactField('home', '650-111-2222', true);

contact.phoneNumbers = phoneNumbers;

// Store contact phone numbers in ContactField[]
var emails = [1];
emails[0] = new ContactField('work', 'first.given@oracle.com');

contact.emails = emails;

// Save
contact.save(onSuccess, onFailure);

function onSuccess()
{
  alert("Create Contact successful.");
}

function onFailure(Error)
```

```
{
  alert("Create Contact failed: " + Error.code);
}
```

Example 9–15 Java Code Example for createContact

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

import oracle.adfmf.framework.contract.adf.ContactAddresses;
import oracle.adfmf.framework.contract.adf.ContactField;
import oracle.adfmf.framework.contract.adf.ContactName;

String givenName = "GivenName";
String familyName = "FamilyName";
String note = "Just a Note";
String phoneNumberType = "mobile";
String phoneNumberValue = "650-000-1234";
String phoneNumberNewValue = "650-111-1234";
String emailType = "home";
String emailTypeNew = "work";
String emailValue = "first.given@oracle.com";
String addressType = "home";
String addressStreet = "500 Oracle Pkwy";
String addressLocality = "Redwood Shores";
String addressCountry = "USA";
String addressPostalCode = "94065";
ContactField[] phoneNumbers = null;
ContactField[] emails = null;
ContactAddresses[] addresses = null;
ContactField[] emails = null;

/*
* Create contact
*/
Contact aContact = new Contact();

ContactName name = new ContactName();
name.setFamilyName(familyName);
name.setGivenName(givenName);
aContact.setName(name);

ContactField phoneNumber = new ContactField();
phoneNumber.setType(phoneNumberType);
phoneNumber.setValue(phoneNumberValue);

phoneNumbers = new ContactField[] { phoneNumber };

ContactField email = new ContactField();
email.setType(emailType);
email.setValue(emailValue);

emails = new ContactField[] { email };

ContactAddresses address = new ContactAddresses();
address.setType(addressType);
address.setStreetAddress(addressStreet);
address.setLocality(addressLocality);
address.setCountry(addressCountry);
```

```
addresses = new ContactAddresses[] { address };

aContact.setNote(note);
aContact.setPhoneNumbers(phoneNumbers);
aContact.setEmails(emails);
aContact.setAddresses(addresses);

// Access device features in Java code by acquiring an instance of the
// DeviceManager from the DeviceManagerFactory.
// Invoking the createContact method, using the DeviceDataControl object.
Contact createdContact = DeviceManagerFactory.getDeviceManager()
    .findContacts.createContact(aContact);
```

9.5.5 How to Use the findContacts Method

The DeviceFeatures data control includes the `findContacts` method, which enables ADF Mobile applications to leverage the device's interface and file system for managing contacts so end users can find one or more contacts from the device's address book. ADF Mobile enables you to display the device's interface and optionally pre-populate the `findContacts` fields. The `findContacts` method takes in a filter string and a list of field names to look through (and return as part of the found contacts). The filter string can be anything to look for in the contacts. For more information about the `findContacts` method, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap documentation (see `http://www.phonegap.com/home`).

The `findContacts` operation takes the following arguments:

- `contactFields`: *Required parameter*. Use this parameter to specify which fields should be included in the `Contact` objects resulting from a `findContacts` operation. Separate fields with a comma (spacing does not matter).
- `filter`: The search string used to filter contacts. (String) (Default: `""`)
- `multiple`: Determines if the `findContacts` operation should return multiple contacts. (Boolean) (Default: `false`)

> **Note:** Passing in a field name that is not in the following list may result in a `null` return value for the `findContacts` operation. Also, only the fields specified in the `Contact` fields argument will be returned as part of the `Contact` object.

The following list shows the possible `Contact` fields that can be passed in to look through and be returned as part of the found contacts:

- `id`: A globally unique identifier
- `displayName`: The name of this contact, suitable for display to end-users
- `name`: An object containing all components of a person's name
- `nickname`: A casual name for the contact
- `phoneNumbers`: An array of all the contact's phone numbers
- `emails`: An array of all the contact's email addresses
- `addresses`: An array of all the contact's addresses
- `ims`: An array of all the contact's Instant messaging (IM) addresses
- `organizations`: An array of all the contact's organizations

- `revision`: The last date the contact was revised
- `birthday`: The birthday of the contact
- `gender`: The gender of the contact
- `note`: A note about the contact
- `photos`: An array of the contact's photos
- `urls`: An array of web pages associated to the contact

To customize a findContacts operation using the DeviceFeatures data control:

1. In JDeveloper, drag the `findContacts` operation from the DeviceFeatures data control in the Data Controls panel and drop it on the page designer as an ADF Mobile **Link**, **Button**, or **Parameter Form**.

   Link or Button: You will be prompted with the Edit Action Binding dialog to enter values for arguments to the `findContacts` operation. At runtime, a button or link will be displayed on the page, which will use the entered values to perform a `findContacts` operation when pressed.

   Parameter Form: Customize the form in the Edit Form Fields dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various Contact fields described above. Below this form will be a button, which will use the entered values to perform a `findContacts` operation when pressed.

2. You can also drag a `Contact` return object from under the `findContacts` operation in the Data Controls panel and drop it on to the page as an **ADF Form**. You can then customize the form in the Edit Form Fields dialog. When the `findContacts` operation is performed, the results will be displayed in this form.

Example 9–16 shows possible argument values for the `findContacts` method. Example 9–17 and Example 9–18 show how to find a contact by family name and get the contact's name, phone numbers, email, addresses, and note.

Example 9–16 Possible Argument Values for findContacts

```
// This will return just one contact with only the ID field:
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts("", "", false);

// This will return all contacts with only ID fields:
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts("", "", true);

// This will return just one contact with all fields:
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts("*", "", false);

// This will return all contacts with all fields:
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts("*", "", true);

// These will throw an exception as contactFields is a required argument and cannot be null:
DeviceManagerFactory.getDeviceManager().findContacts(null, "", false);
DeviceManagerFactory.getDeviceManager().findContacts(null, "", true);

// These will throw an exception as the filter argument cannot be null:
DeviceManagerFactory.getDeviceManager().findContacts("", null, false);
DeviceManagerFactory.getDeviceManager().findContacts("", null, true);
```

Note: The `Contact` fields passed are strings (containing the comma-delimited fields). If any arguments are passed as `null` to the method, an ADF exception is thrown.

Example 9–17 JavaScript Code Example for findContacts

```
var filter = ["name", "phoneNumbers", "emails", "addresses", "note"];

var options = new ContactFindOptions();
options.filter="FamilyName";

// Contacts, like many other device-specific features, are accessed from
// the global 'navigator' object in JavaScript.
navigator.contacts.find(filter, onSuccess, onFail, options);

function onSuccess(contacts)
{
  alert ("Find Contact call succeeded! Number of contacts found = " + contacts.length);
}

function onFail(Error)
{
  alert("Find Contact failed: " + Error.code);
}
```

Example 9–18 Java Code Example for findContacts

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

/*
 * Find Contact - Find contact by family name.
 *
 * See if we can find the contact that we just created.
 */

String familyName = "FamilyName"

// Access device features in Java code by acquiring an instance of the
// DeviceManager from the DeviceManagerFactory.
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts(
    "name,phoneNumbers,emails,addresses,note", familyName, true);
```

9.5.6 How to Use the updateContact Method

The DeviceFeatures data control includes the `updateContact` method, which enables ADF Mobile applications to leverage the device's interface and file system for managing contacts so end users can update contacts in the device's address book. ADF Mobile enables you to display the device's interface and optionally pre-populate the `updateContact` fields. The `updateContact` method takes in a `Contact` object as a parameter and returns the updated `Contact` object, as shown in Example 9–19.

Note: The `Contact` object that is needed as the input parameter can be found using the `findContacts` method as described in Section 9.5.5, "How to Use the findContacts Method." If a null `Contact` object is passed in to the method, an ADF exception is thrown.

To customize an updateContact operation using the DeviceFeatures data control:

1. In JDeveloper, drag the **`updateContact`** operation from the DeviceFeatures data control in the Data Controls panel and drop it on the page designer as an ADF Mobile **Link**, **Button**, or **Parameter Form**.

   Link or Button: You will be prompted with the Edit Action Binding dialog to enter values for arguments to the `updateContact` operation. At runtime, a button or link will be displayed on the page, which will use the entered values to perform an `updateContact` operation when pressed.

   Parameter Form: Customize the form in the Edit Form Fields dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various Contact fields. Below this form will be a button, which will use the entered values to perform an `updateContact` operation when pressed.

2. You can also drag a **`Contact`** return object from under the `updateContact` operation in the Data Controls panel and drop it on to the page as an **ADF Form**. You can then customize the form in the Edit Form Fields dialog. When the `updateContact` operation is performed, the results will be displayed in this form.

Example 9–19 and Example 9–21 show how to update a contact's phone number. Example 9–20 and Example 9–22 show how to add another phone number to a contact.

For information about the `updateContact` method and the `Contact` object, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap documentation (see `http://www.phonegap.com/home`). Also see Section 9.5.5, "How to Use the findContacts Method" for a description of `Contact` fields.

Example 9–19 JavaScript Code Example for updateContact

```
function updateContact(contact)
{
  try
  {
    if (null != contact.phoneNumbers)
    {
      alert("Number of phone numbers = " + contact.phoneNumbers.length);
      var numPhoneNumbers = contact.phoneNumbers.length;
      for (var j = 0; j < numPhoneNumbers; j++)
      {
        alert("Type: " + contact.phoneNumbers[j].type + "\n" +
              "Value: "  + contact.phoneNumbers[j].value + "\n" +
              "Preferred: "  + contact.phoneNumbers[j].pref);

        contact.phoneNumbers[j].type = "mobile";
        contact.phoneNumbers[j].value = "408-123-4567";
      }

      // save
      contact.save(onSuccess, onFailure);
    }
    else
```

```
    {
      //alert ("No phone numbers found in the contact.");
    }
  }
  catch(e)
  {
    alert("updateContact - ERROR: " + e.description);
  }
}

function onSuccess()
{
  alert("Update Contact successful.");
}

function onFailure(Error)
{
  alert("Update Contact failed: " + Error.code);
```

Example 9–20 shows you how to add another phone number to the already existing phone numbers.

Example 9–20 JavaScript Code Example for Adding a Phone Number with updateContact

```
function updateContact(contact)
{
  try
  {
    var phoneNumbers = [1];
    phoneNumbers[0] = new ContactField('home', '650-111-2222', true);
    contact.phoneNumbers = phoneNumbers;

    // save
    contact.save(onSuccess, onFailure);
  }
  catch(e)
  {
    alert("updateContact - ERROR: " + e.description);
  }
}

function onSuccess()
{
  alert("Update Contact successful.");
}

function onFailure(Error)
{
  alert("Update Contact failed: " + Error.code);
}
```

Example 9–21 shows how to update a contact's phone number, email type, and postal code.

Example 9–21 Java Code Example for updateContact

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

/*
```

```
 * Update Contact - Updating phone number, email type, and adding address postal code
 */
String familyName = "FamilyName";
String phoneNumberNewValue = "650-111-1234";
String emailTypeNew = "work";
String addressPostalCode = "94065";

Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts(
    "name,phoneNumbers,emails,addresses,note", familyName, true);

// Assuming there was only one contact returned, we can use the first contact in the array.
// If more than one contact is returned then we have to filter more to find the exact contact
// we need to update.

foundContacts[0].getPhoneNumbers()[0].setValue(phoneNumberNewValue);
foundContacts[0].getEmails()[0].setType(emailTypeNew);
foundContacts[0].getAddresses()[0].setPostalCode(addressPostalCode);

Contact updatedContact = DeviceManagerFactory.getDeviceManager().updateContact(foundContacts[0]);
```

Example 9–22 shows you how to add another phone number to the already existing phone numbers.

Example 9–22 Java Code Example for Adding a Phone Number with updateContact

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

String additionalPhoneNumberValue = "999-888-0000";
String additionalPhoneNumberType = "mobile";
// Create a new phoneNumber that will be appended to the previous one.
ContactField additionalPhoneNumber = new ContactField();
additionalPhoneNumber.setType(additionalPhoneNumberType);
additionalPhoneNumber.setValue(additionalPhoneNumberValue);

foundContacts[0].setPhoneNumbers(new ContactField[] { additionalPhoneNumber });

// Access device features in Java code by acquiring an instance of the DeviceManager
// from the DeviceManagerFactory.
Contact updatedContact = DeviceManagerFactory.getDeviceManager().updateContact(foundContacts[0]);
```

9.5.7 How to Use the removeContact Method

The DeviceFeatures data control includes the `removeContact` method, which enables ADF Mobile applications to leverage the device's interface and file system for managing contacts so end users can remove contacts from the device's address book. ADF Mobile enables you to display the device's interface and optionally pre-populate the `removeContact` fields. The `removeContact` method takes in a `Contact` object as a parameter, as shown in Example 9–23.

> **Note:** The `Contact` object that is needed as the input parameter can be found using the `findContacts` method as described in Section 9.5.5, "How to Use the findContacts Method."

To customize a removeContact operation using the DeviceFeatures data control:

1. In JDeveloper, drag the **`removeContact`** operation from the DeviceFeatures data control in the Data Controls panel and drop it on the page designer as an ADF Mobile **Link**, **Button**, or **Parameter Form**.

   Link or Button: You will be prompted with the Edit Action Binding dialog to enter values for arguments to the `removeContact` operation. At runtime, a button or link will be displayed on the page, which will use the entered values to perform a `removeContact` operation when pressed.

   Parameter Form: Customize the form in the Edit Form Fields dialog. At runtime, an editable form will be displayed on the page, which enables the application user to enter values for the various Contact fields. Below this form will be a button, which will use the entered values to perform a `removeContact` operation when pressed.

2. You can also drag a `Contact` return object from under the `removeContact` operation in the Data Controls panel and drop it on to the page as an **ADF Form**. You can then customize the form in the Edit Form Fields dialog. When the `removeContact` operation is performed, the results will be displayed in this form.

Example 9–23 and Example 9–24 show you how to delete a contact that you found using `findContacts`. For information about the `removeContact` method and the Contact object, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap documentation (see `http://www.phonegap.com/home`).

Example 9–23 JavaScript Code Example for removeContact

```
// Remove the contact from the device
contact.remove(onSuccess,onError);

function onSuccess()
{
  alert("Removal Success");
}

function onError(contactError)'
{
  alert("Error = " + contactError.code);
}
```

Example 9–24 Java Code Example for removeContact

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;

/*
 * Remove the contact from the device
 */
Contact[] foundContacts = DeviceManagerFactory.getDeviceManager().findContacts(
    "name,phoneNumbers,emails,addresses", familyName, true);

// Assuming there is only one contact returned, we can use the first contact in the array.
// If more than one contact is returned we will have to filter more to find the
// exact contact we want to remove.

// Access device features in Java code by acquiring an instance of the DeviceManager
// from the DeviceManagerFactory.
DeviceManagerFactory.getDeviceManager().removeContact(foundContacts[0]);
```

9.5.8 How to Use the startLocationMonitor Method

The DeviceFeatures data control includes the `startLocationMonitor` method, which enables ADF Mobile applications to provide a geolocation feature so end users can obtain the device's location. ADF Mobile enables you to display the device's interface and optionally pre-populate the `startLocationMonitor` fields.

ADF Mobile exposes APIs that enable you to acquire the device's current position, allowing you to retrieve the device's current location for one instant in time or to subscribe to it on a periodic basis. Example 9–25 and Example 9–26 show code examples that will allow your application to obtain the device's location. For information about the `startLocationMonitor` method, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap documentation (see `http://www.phonegap.com/home`).

To listen for changes in the device's location using the DeviceFeatures data control:

In JDeveloper, drag the `startLocationMonitor` operation from the DeviceFeatures data control in the Data Controls panel to the page designer and drop it as an ADF Mobile **Link** or **Button**. When prompted by the **Edit Action Dialog**, populate the fields as follows:

- `enableHighAccuracy`: If `true`, use the most accurate possible method of obtaining a location fix. This is just a hint; the operating system may not respect it. Devices often have several different mechanisms for obtaining a location fix, including cell tower triangulation, Wi-Fi hotspot lookup, and true GPS. Specifying `false` indicates that you are willing to accept a less accurate location, which may result in a faster response or consume less power.
- `updateInterval`: Defines how often, in milliseconds, to receive updates. Location updates may not be delivered as frequently as specified; the operating system may wait until a significant change in the device's position has been detected before triggering another location update.
- `locationListener`: EL expression that resolves to a bean method with the following signature:

  ```
  void methodName(Location newLocation)
  ```

  This EL expression will be evaluated every time a location update is received. For example, enter `viewScope.LocationListenerBean.locationUpdated` (without the surrounding`#{}`), then define a bean named `LocationListenerBean` in `viewScope` and implement a method with the following signature:

  ```
  public void locationUpdated(Location currentLocation)
  {
    System.out.println(currentLocation);
    // To stop subscribing to location updates, invoke the following:
    // DeviceManagerFactory.getDeviceManager().clearWatchPosition(
    //     currentLocation.getWatchId());
  }
  ```

> **Note:** Do not use the EL syntax `#{LocationListenerBean.locationUpdate}` to specify the `locationListener`, unless you truly want the result of evaluating that expression to be the name of the `locationListener`.

Example 9–25 shows how to subscribe to changes in the device's location periodically. The example uses the `DeviceManager.startUpdatingPosition` method, which takes the following parameters:

- `int updateInterval`: Defines how often to deliver location updates, in milliseconds. Location updates may not be delivered as frequently as specified; the operating system may wait until a significant change in the device's position has been detected before triggering another location update. Conversely, location updates may also be delivered at the specified frequency, but may be identical until the device's position has changed significantly.
- `boolean enableHighAccuracy`: If set to `true`, use the most accurate possible method of obtaining a location fix.
- `String watchID`: Defines a unique ID that can be subsequently used to stop subscribing to location updates
- `GeolocationCallback`: An implementation of the GeolocationCallback interface. This implementation's `locationUpdated` method is invoked each time the location is updated, as shown in Example 9–25.

For an example of how to subscribe to changes in the device's position using JavaScript, refer to the PhoneGap documentation (see `http://www.phonegap.com/home`).

Parameters returned in the callback function specified by the `locationListener` are as follows:

- `double getAccuracy`—Accuracy level of the latitude and longitude coordinates in meters
- `double getAltitude`—Height of the position in meters above the ellipsoid
- `double getLatitude`—Latitude in decimal degrees
- `double getLongitude`—Longitude in decimal degrees
- `double getAltitudeAccuracy`—Accuracy level of the altitude coordinate in meters
- `double getHeading`—Direction of travel, specified in degrees counting clockwise relative to the true north
- `double getSpeed`—Current ground speed of the device, specified in meters per second
- `long getTimestamp`—Creation of a timestamp in milliseconds since the Unix epoch
- `String getWatchId`—Only used when subscribing to periodic location updates. A unique ID that can be subsequently used to stop subscribing to location updates

For more information about the `startLocationMonitor` and `startHeadingMonitor` methods, see the `DeviceDataControl` class in the ADF Mobile Javadoc and refer to the PhoneGap documentation (see `http://www.phonegap.com/home`).

Example 9–25 Using Geolocation to Subscribe to Changes in the Device's Location

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;
import oracle.adf.model.datacontrols.device.GeolocationCallback;
import oracle.adfmf.framework.contract.adf.Location;

// Subscribe to location updates that will be delivered every 20 seconds, with high accuracy.
// As we can have multiple subscribers, let's identify this one as 'MyGPSSubscriptionID'.
```

```
// Notice that this call returns the watchID, which is usually the same as the watchID passed in.
// However, it may be different if the specified watchID conflicts with an existing watchID,
// so be sure to always use the returned watchID.
String watchID = DeviceManagerFactory.getDeviceManager().startUpdatingPosition(20000, true, "
       "MyGPSSubscriptionID", new GeolocationCallback() {
    public void locationUpdated(Location position) {
        System.out.println("Location updated to: " + position);
    }
});

// The previous call returns immediately so that you can continue processing.
// When the device's location changes, the locationUpdated() method specified in
// the previous call will be invoked in the context of the current feature.

// When you wish to stop being notified of location changes, call the following method:
DeviceManagerFactory().getDeviceManager().clearWatchPosition(watchID);
```

To obtain the device's location using the DeviceFeatures data control:

In JDeveloper, drag the **`startLocationMonitor`** operation from the DeviceFeatures data control in the Data Controls panel to the page designer and drop it as an ADF Mobile **Link** or **Button**. Follow Example 9–25, but stop listening after the first location update is received.

Example 9–26 shows how to get a device's location one time. The example uses `DeviceManager.getCurrentPosition`, which takes the following parameters:

- `int maximumAge:` Accept a cached value no older than this value, in milliseconds. If a location fix has been obtained within this window of time, then it will be returned immediately; otherwise, the call will block until a new location fix can be determined.
- `boolean: enableHighAccuracy` If set to true, use the most accurate possible method of obtaining a location fix.

Example 9–26 Using Geolocation to Get the Device's Current Location (One Time)

```
import oracle.adf.model.datacontrols.device.DeviceManagerFactory;
import oracle.adfmf.framework.contract.adf.Location;

// Get the device's current position, with highest accuracy, and accept a cached location that is
// no older than 60 seconds.
Location currentPosition = DeviceManagerFactory.getDeviceManager().getCurrentPosition(60000, true);
System.out.println("The device's current location is: latitude=" + currentPosition.getLatitude() +
    ", longitude=" + currentPosition.getLongitude());
```

9.5.9 Device Properties

There may be features of your application that rely on specific device characteristics or capabilities. For example, you may want to present a different user interface depending on the device's screen orientation, or there may be a mapping feature that you want to enable only if the device supports geolocation. ADF Mobile provides a number of properties that you can access from Java and EL in order to support this type of dynamic behavior. Table 9–6 lists these properties, along with information about how to query them, what values to expect in return, and whether the property can change during the application's lifecycle.

Table 9–6 Device Properties and Corresponding EL Expressions

Property	Static/ Dynamic	EL Expression	Sample Value	Java API
device.name	Static	#{deviceScope.device.name}	"iPhone Simulator", "Joe Smith's iPhone"	DeviceManager.getName()
device.platform	Static	#{deviceScope.device.platform}	"iPhone Simulator", "iPhone"	DeviceManager.getPlatform()
device.version	Static	#{deviceScope.device.version}	"4.3.2", "5.0.1"	DeviceManager.getVersion()
device.os	Static	#{deviceScope.device.os}	"iOS"	DeviceManager.getOs()
device.model	Static	#{deviceScope.device.model}	"x86_64", "i386", "iPhone3,1"	DeviceManager.getModel()
device.phonegap	Static	#{deviceScope.device.phonegap}	"1.0.0"	DeviceManager.getPhonegap()
hardware.hasCamera	Static	#{deviceScope.hardware.hasCamera}	"true", "false"	DeviceManager.hasCamera()
hardware.hasContacts	Static	#{deviceScope.hardware.hasContacts}	"true", "false"	DeviceManager.hasContacts()
hardware.hasTouchScreen	Static	#{deviceScope.hardware.hasTouchScreen}	"true", "false"	DeviceManager.hasTouchScreen()
hardware.hasGeolocation	Static	#{deviceScope.hardware.hasGeolocation}	"true", "false"	DeviceManager.hasGeolocation()
hardware.hasAccelerometer	Static	#{deviceScope.hardware.hasAccelerometer}	"true", "false"	DeviceManager.hasAccelerometer()
hardware.hasCompass	Static	#{deviceScope.hardware.hasCompass}	"true", "false"	DeviceManager.hasCompass()
hardware.hasFileAccess	Static	#{deviceScope.hardware.hasFileAccess}	"true", "false"	DeviceManager.hasFileAccess()
hardware.hasLocalStorage	Static	#{deviceScope.hardware.hasLocalStorage}	"true", "false"	DeviceManager.hasLocalStorage()
hardware.hasMediaPlayer	Static	#{deviceScope.hardware.hasMediaPlayer}	"true", "false"	DeviceManager.hasMediaPlayer()
hardware.hasMediaRecorder	Static	#{deviceScope.hardware.hasMediaRecorder}	"true", "false"	DeviceManager.hasMediaRecorder()
hardware.networkStatus	Dynamic	#{deviceScope.hardware.networkStatus}	"wifi", "2g", "unknown", "none"[1]	DeviceManager.getNetworkStatus()
hardware.screen.width	Dynamic	#{deviceScope.hardware.screen.width}	320, 480	DeviceManager.getScreenWidth()
hardware.screen.height	Dynamic	#{deviceScope.hardware.screen.height}	480, 320	DeviceManager.getScreenHeight()
hardware.availableWidth	Dynamic	#{deviceScope.hardware.screen.availableWidth}	<= 320, <= 480	DeviceManager.getAvailableScreenWidth()
hardware.availableHeight	Dynamic	#{deviceScope.hardware.screen.availableHeight}	<= 480, <= 320	DeviceManager.getAvailableScreenHeight()

Table 9–6 (Cont.) Device Properties and Corresponding EL Expressions

Property	Static/ Dynamic	EL Expression	Sample Value	Java API
hardware.screen.dpi	Static	#{deviceScope.hardware.screen.dpi}	160, 326	DeviceManager.getScreenDpi()
hardware.screen.diagonalSize	Static	#{deviceScope.hardware.screen.diagonalSize}	9.7, 6.78	DeviceManager.getScreenDiagonalSize()
hardware.screen.scaleFactor	Static	#{deviceScope.hardware.screen.scaleFactor}	1.0, 2.0	DeviceManager.getScreenScaleFactor()

[1] If both wifi and 2G are turned on, network status will be wifi, as wifi takes precedence over 2G.

9.6 Performing Validation

In ADF Mobile, validation occurs in the data control layer, with validation rules set on binding attributes behaving similarly to those in Oracle ADF. The attribute validation takes place at a single point in the system, during the setValue operation on the bindings.

You can define the following validators for attributes exposed by the data controls:

- Compare validator
- Length validator
- List validator
- Range validator

All validators for a given attribute are executed, and nested exceptions are thrown for every validator that does not pass. You can define a validation message for attributes, which is displayed when a validation rule is fired at runtime. For more information, see Section 8.9, "Validating Input" and Section 9.6.1, "How to Add Validation Rules."

> **Note:** Due to a JSON limitation, the value that a BigDecimal can hold is within the range of a Double, and the value that a BigInteger can hold is within the range of a Long. If you want to use numbers greater than those allowed, you can call toString on BigDecimal or BigInteger to serialize values as String.

Table 9–7 lists supported validation combinations for the length validator.

Table 9–7 Length Validation

Compare type	Byte	Character
Equals	Supported	Supported
Not Equals	Supported	Supported
Less Than	Supported	Supported
Greater Than	Supported	Supported

Table 9–7 (Cont.) Length Validation

Compare type	Byte	Character
`Less Than Equal To`	Supported	Supported
`Greater Than Equal To`	Supported	Supported
`Between`	Supported	Supported

Table 9–8 and Table 9–9 list supported validation combinations for the range validator.

Table 9–8 Range Validation

Compare type	Byte	Char	Double	Float	Integer	Long	Short
`Between`	Supported	Supported	Supported	Supported	Supported	Supported	Supported
`Not Between`	Supported	Supported	Supported	Supported	Supported	Supported	Supported

Table 9–9 Range Validation - math, sql, and util Packages

Compare type	java.math.BigDecimal	java.math. BigInteger	java.sql.Date	java.sql.Time	java.sql.Time stamp	java.util.Date
`Between`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Not Between`	Supported	Supported	Not supported	Not supported	Not supported	Not supported

Table 9–10 lists supported validation combinations for the list validator.

Table 9–10 List Validation

Compare type	String
`In`	Supported
`Not In`	Supported

Table 9–11 and Table 9–12 lists supported validation combinations for the compare validator.

Table 9–11 Compare Validation

Compare type	Byte	Char	Double	Float	Integer	Long	Short	String
`Equals`	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
`Not Equals`	Supported	Supported	Supported	Supported	Supported	Supported	Supported	Supported
`Less Than`	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported
`Greater Than`	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported
`Less Than Equal To`	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported
`Greater Than Equal To`	Not supported	Supported	Supported	Supported	Supported	Supported	Supported	Not supported

Table 9–12 Compare Validation - java.math, java.sql, and java.util Packages

Compare type	java.math. BigDecimal	java.math. BigInteger	java.sql.Date	java.sql.Time	java.sql.Timestamp	java.util.Date
`Equals`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Not Equals`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Less Than`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Greater Than`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Less Than Equal To`	Supported	Supported	Not supported	Not supported	Not supported	Not supported
`Greater Than Equal To`	Supported	Supported	Not supported	Not supported	Not supported	Not supported

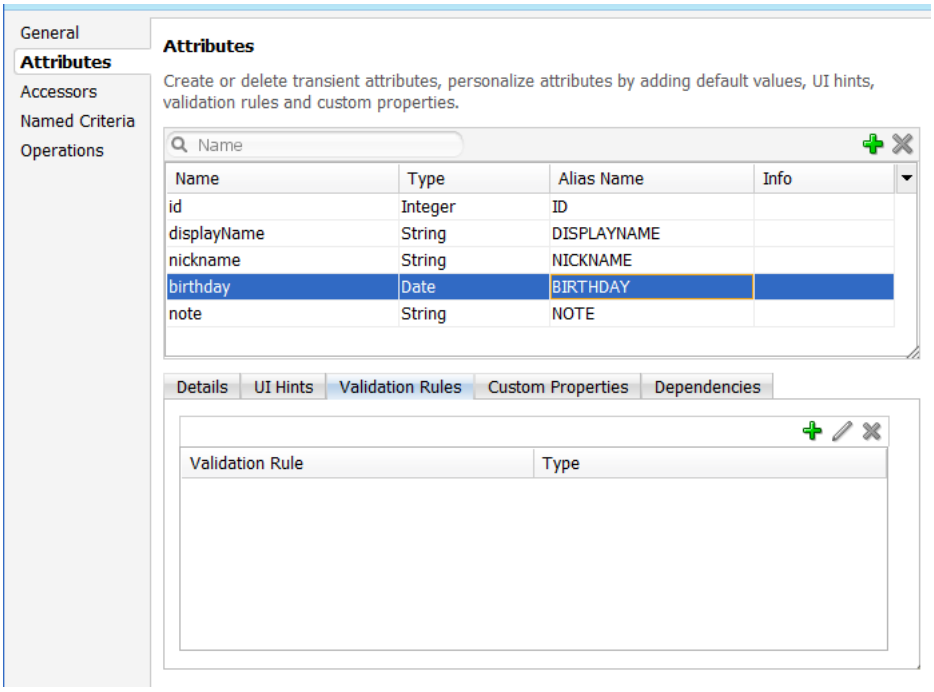
9.6.1 How to Add Validation Rules

You can define validation rules for a variety of use cases. To add a declarative validation rule to an entity object, use the Overview Editor for Data Control Structure Files - Attributes Page.

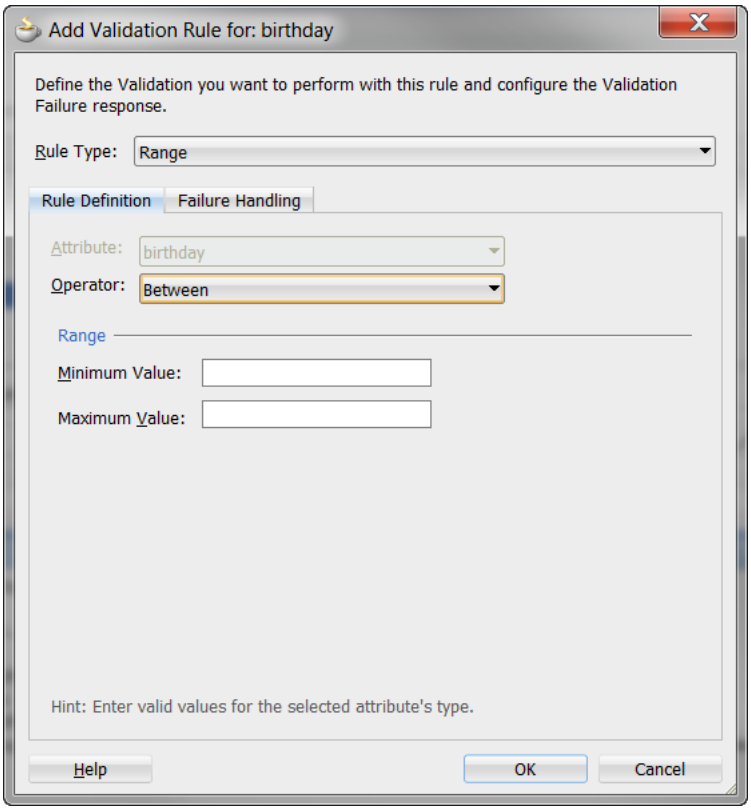
For more information about adding validation rules, see the "Adding Validation Rules to Entity Objects and Attributes" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To add a validation rule:

1. From the Data Controls panel, right-click on a data controls object and choose **Edit Definition**.
2. In the Overview Editor for Data Control Structure Files, select the **Attributes** page.



3. Select the **Validation Rules** tab in the lower part of the page and then click **Add**. In the resulting **Add Validation Rule** dialog, define the validation rule and the failure handling.



For more information, see the following:

- Section 7.3.2.4, "Adding Data Controls to the View"
- "Using the Built-in Declarative Validation Rules" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

9.6.2 What You May Need to Know About the Validator Metadata

The validator metadata is placed into the data control structure metadata XML files at design time. Example 9–27 shows a sample length validator.

Example 9–27 Length Validator Declared in Metadata File

```
<?xml version="1.0" encoding="windows-1252" ?>
<!DOCTYPE PDefViewObject SYSTEM "jbo_03_01.dtd">
<PDefViewObject
   xmlns="http://xmlns.oracle.com/bc4j"
   Name="Product"
   Version="12.1.1.61.36"
   xmlns:validation="http://xmlns.oracle.com/adfm/validation">
   <DesignTime>
      <Attr Name="_DCName" Value="DataControls.ProductListBean"/>
      <Attr Name="_SDName" Value="mobile.Product"/>
   </DesignTime>
   <PDefAttribute
      Name="name">
      <validation:LengthValidationBean
```

```
            Name="nameRule0"
            OnAttribute="name"
            CompareType="GREATERTHAN"
            DataType="BYTE"
            CompareLength="5"
            Inverse="false"/>
      </PDefAttribute>
</PDefViewObject>
```

9.7 Data Change Events

To simplify data change events, JDeveloper uses the property change listener pattern. In most cases you can use JDeveloper to generate the necessary code to source notifications from your beans' property accessors by selecting the Notify listeners when property changes checkbox in the Generate Accessors dialog (see Section 9.2.4.2, "ADF Managed Beans" for details). The `PropertyChangeSupport` object is generated automatically, with the calls to `firePropertyChange` in the newly-generated setter method. Additionally, the `addPropertyChangeListener` and `removePropertyChangeListener` methods are added so property change listeners can register and unregister themselves with this object. This is what the framework uses to capture changes to be pushed to the client cache and to notify the user interface layer that data has been changed.

Property changes alone will not solve all the data change notifications, as in the case where you have a bean wrapped by a data control and you want to expose a collection of items. While a property change is sufficient for individual items of the list changing, it is not sufficient for cardinality changes. In this case, rather than fire a property change for the entire collection, which would cause a degradation of performance, you can instead refresh just the collection delta. To do this you need to expose more data than is required for a simple property change, which you can do using the `ProviderChangeSupport` class.

Since the provider change is required only when you have a dynamic collection exposed by a data control wrapped bean, there are only a few types of provider change events to fire:

- `fireProviderCreate`—when a new element is added to the collection
- `fireProviderDelete`—when an element is removed from the collection
- `fireProviderRefresh`—when multiple changes are done to the collection at one time and you decide it is better to simply ask for the client to refresh the entire collection (this should only be used in bulk operations)

The `ProviderChangeSupport` class is used for sending notifications relating to collection elements, so that components update properly when a change occurs in a Java bean data control. It follows a similar pattern to the automatically-generated `PropertyChangeSupport` class, but the event objects used with `ProviderChangeSupport` send more information, including the type of operation as well as the key and position of the element that changed. `ProviderChangeSupport` captures structural changes to a collection, such as adding or removing an element (or provider) from a collection. `PropertyChangeSupport` captures changes to the individual items in the collection.

Example 9–28 shows how to use `ProviderChangeSupport` for sending notifications relating to structural changes to collection elements (such as when adding or removing a child). For more information on the `ProviderChangeListener` interface and the `ProviderChangeEvent` class, see the ADF Mobile Javadoc.

Example 9–28 ProviderChangeSupport Code Example

```
public class NotePad {
   private   static    List
                       s_notes              = null;

   protected transient PropertyChangeSupport
                       propertyChangeSupport = new PropertyChangeSupport(this);
   protected transient ProviderChangeSupport
                       providerChangeSupport = new ProviderChangeSupport(this);

    public NotePad() {
        …
    }

    public  mobile.Note[] getNotes() {
        mobile.Note n[] = null;

        synchronized (this)
        {
            if(s_notes.size() > 0) {
                n = (mobile.Note[])
                    s_notes.toArray(new mobile.Note[s_notes.size()]);
            }
            else {
                n = new mobile.Note[0];
            }
        }

        return n;
    }

    public void addNote() {
        System.out.println("Adding a note ....");
        Note  n = new Note();
        int   s = 0;

        synchronized (this)
        {
            s_notes.add(n);
            s = s_notes.size();
        }

        System.out.println("firing the events");
        providerChangeSupport.fireProviderCreate("notes", n.getUid(), n);
    }

    public void removeNote() {
        System.out.println("Removng a note ....");
        if(s_notes.size() > 0) {
            int     end = -1;
            Note    n   = null;

            synchronized (this)
            {
                end   = s_notes.size() - 1;
                n     = (Note)s_notes.remove(end);
            }

            System.out.println("firing the events");
        providerChangeSupport.fireProviderDelete("notes", n.getUid());
```

```
        }
    }

    public void RefreshNotes() {
        System.out.println("Refreshing the notes ....");

        providerChangeSupport.fireProviderRefresh("notes");
    }

    public void addProviderChangeListener(ProviderChangeListener l) {
        providerChangeSupport.addProviderChangeListener(l);
    }

    public void removeProviderChangeListener(ProviderChangeListener l) {
        providerChangeSupport.removeProviderChangeListener(l);
    }

    protected String   status;

    /* --- JDeveloper generated accessors --- */

    public void addPropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public void setStatus(String status) {
        String oldStatus = this.status;
        this.status = status;
        propertyChangeSupport.firePropertyChange("status", oldStatus, status);
    }

    public String getStatus() {
        return status;
    }
}
```

Data changes are passed back to the client (to be cached) with any response message or return value from the JVM layer. This allows JDeveloper to compress and reduce the number of events and updates to refresh to the user interface, allowing the framework to be as efficient as possible.

However, there are times where you may need to have a background thread handle a long-running process (such as web-service interactions, database interactions, or expensive computations) and notify the user interface independent of a user action. To update data on an AMX page to reflect the current values of data fields whose values have changed, you can avoid the performance hit associated with reloading the whole AMX page by calling `AdfmfJavaUtilities.flushDataChangeEvent` to force the currently queued data changes to the client.

Note: The `flushDataChangeEvent` method can only be executed from a background thread.

Example 9–29 shows how the `flushDataChangeEvent` method can be used to force pending data changes to the client. For more information about

`oracle.adfmf.framework.api.AdfmfJavaUtilities.flushDataChangeEvent`, see *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile*.

Example 9–29 Data Change Event Example

```
/* Note - Simple POJO used by the NotePad managed bean or data control wrapped bean */

package mobile;

import oracle.adfmf.amx.event.ActionEvent;
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
import oracle.adfmf.java.beans.PropertyChangeListener;
import oracle.adfmf.java.beans.PropertyChangeSupport;


/**
 * Simple note object
 * uid   - unique id - generated and not mutable
 * title - title for the note - mutable
 * note  - note comment - mutable
 */
public class Note {
    /* standard jdev generated property change support */
    protected transient PropertyChangeSupport
                      propertyChangeSupport = new PropertyChangeSupport(this);


    private static boolean s_backgroundFlushTestRunning = false;


    public Note() {
        this("" + (System.currentTimeMillis() % 10000));
    }

    public Note(String id) {
        this("UID-"+id, "Title-"+id, "");
    }

    public Note(String uid, String title, String note) {
        this.uid     = uid;
        this.title   = title;
        this.note    = note;
    }


    /* update the current note with the values passed in */
    public void updateNote(Note n) {
        if (this.getUid().compareTo(n.getUid()) == 0) {
            this.setTitle(n.getTitle());
            this.setNote(n.getNote());
        } else {
            throw new IllegalArgumentException("note");
        }
    }


    /* background thread to simulate some background process that make changes */
    public void startNodeBackgroundThread(ActionEvent actionEvent) {
        Thread backgroundThread   = new Thread() {
```

```
        public void run() {
            System.out.println("startBackgroundThread enter - " +
                                                s_backgroundFlushTestRunning);

            s_backgroundFlushTestRunning = true;
            for(int i = 0; i <= iterations; ++i) {
                try
                {
                    System.out.println("executing " + i + " of " + iterations + "
                            " iterations.");

                    /* update a property value */
                    if(i == 0) {
                        setNote("thread starting");
                    }
                    else if( i == iterations) {
                        setNote("thread complete");
                        s_backgroundFlushTestRunning = false;
                    }
                    else {
                        setNote("executing " + i + " of " + iterations + " iterations.");
                    }
                    setVersion(getVersion() + 1);
                    setTitle("Thread Test v" + getVersion());
                    AdfmfJavaUtilities.flushDataChangeEvent();  /* key line */
                }
                catch(Throwable t)
                {
                    System.err.println("Error in the background thread: " + t);
                }

                try {
                    Thread.sleep(delay);  /* sleep for 6 seconds */
                } catch (InterruptedException ex) {
                    // TODO Auto-generated catch block
                    ex.printStackTrace();
                }
            }
        }
    };

    backgroundThread.start();
}

protected String uid;
protected String title;
protected String note;
protected int    version;

protected int    iterations =  10;
protected int    delay      = 500;


/* --- jdev generated accessors --- */

public void addPropertyChangeListener(PropertyChangeListener l) {
    propertyChangeSupport.addPropertyChangeListener(l);
}

public void removePropertyChangeListener(PropertyChangeListener l) {
```

```
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public String getUid() {
        return uid;
    }

    public void setTitle(String title) {
        String oldTitle = this.title;
        this.title = title;
        propertyChangeSupport.firePropertyChange("title", oldTitle, title);
    }

    public String getTitle() {
        return title;
    }

    public void setNote(String note) {
        String oldNote = this.note;
        this.note = note;
        propertyChangeSupport.firePropertyChange("note", oldNote, note);
    }

    public String getNote() {
        return note;
    }

    public void setVersion(int version) {
        int oldVersion = this.version;
        this.version = version;
        propertyChangeSupport.firePropertyChange("version", oldVersion, version);
    }

    public int getVersion() {
        return version;
    }

    public void setIterations(int iterations) {
        int oldIterations = this.iterations;
        this.iterations = iterations;
        propertyChangeSupport.
                 firePropertyChange("iterations", oldIterations, iterations);
    }

    public int getIterations() {
        return iterations;
    }

    public void setDelay(int delay) {
        int oldDelay = this.delay;
        this.delay = delay;
        propertyChangeSupport.
                firePropertyChange("delay", oldDelay, delay);
    }

    public int getDelay() {
        return delay;
    }
}
```

```
/* NotePad - Can be used as a managed bean or wrapped as a data control */

package mobile;

import java.util.ArrayList;
import java.util.List;

import oracle.adfmf.amx.event.ActionEvent;
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
import oracle.adfmf.java.beans.PropertyChangeListener;
import oracle.adfmf.java.beans.PropertyChangeSupport;
import oracle.adfmf.java.beans.ProviderChangeListener;
import oracle.adfmf.java.beans.ProviderChangeSupport;


public class NotePad {
    private static List     s_notes                      = null;
    private static boolean  s_backgroundFlushTestRunning = false;

    protected transient     PropertyChangeSupport
        propertyChangeSupport = new PropertyChangeSupport(this);

    protected transient     ProviderChangeSupport
        providerChangeSupport = new ProviderChangeSupport(this);

    public NotePad() {
        if (s_notes == null) {
            s_notes = new ArrayList();

            for(int i = 1000; i < 1003; ++i) {
                s_notes.add(new Note(""+i));
            }
        }
    }

    public  mobile.Note[] getNotes() {
        mobile.Note n[] = null;

        synchronized (this)
        {
            if(s_notes.size() > 0) {
                n = (mobile.Note[])s_notes.
                     toArray(new mobile.Note[s_notes.size()]);
            }
            else {
                n = new mobile.Note[0];
            }
        }

        return n;
    }

    public void addNote() {
        System.out.println("Adding a note ....");
        Note  n = new Note();
        int   s = 0;
```

```
        synchronized (this)
        {
            s_notes.add(n);
            s = s_notes.size();
        }

        System.out.println("firing the events");

        /* update the note count property on the screen */
        propertyChangeSupport.
             firePropertyChange("noteCount", s-1, s);

        /* update the notes collection model with the new note */
        providerChangeSupport.
             fireProviderCreate("notes", n.getUid(), n);

        /* to update the client side model layer */
        AdfmfJavaUtilities.flushDataChangeEvent();
    }

    public void removeNote() {
        System.out.println("Removing a note ....");
        if(s_notes.size() > 0) {
            int     end = -1;
            Note    n   = null;

            synchronized (this)
            {
                end   = s_notes.size() - 1;
                n     = (Note)s_notes.remove(end);
            }

            System.out.println("firing the events");

            /* update the client side model layer */
            providerChangeSupport.
                fireProviderDelete("notes", n.getUid());

           /* update the note count property on the screen */
           propertyChangeSupport.
                firePropertyChange("noteCount", -1, end);
        }
    }

    public void RefreshNotes() {
        System.out.println("Refreshing the notes ....");

        /* update the entire notes collection on the client */
        providerChangeSupport.fireProviderRefresh("notes");
    }

    public int getNoteCount() {
        int size = 0;

        synchronized (this)
        {
            size = s_notes.size();
        }
       return size;
    }
```

```
public void
addProviderChangeListener(ProviderChangeListener l) {
    providerChangeSupport.addProviderChangeListener(l);
}

public void
removeProviderChangeListener(ProviderChangeListener l) {
    providerChangeSupport.removeProviderChangeListener(l);
}

public void
startListBackgroundThread(ActionEvent actionEvent) {
    for(int i = 0; i < 10; ++i) {
        _startListBackgroundThread(actionEvent);
        try {
            Thread.currentThread().sleep(i * 1234);
        } catch (InterruptedException e) {
        }
    }
}

public void
_startListBackgroundThread(ActionEvent actionEvent) {
    Thread backgroundThread   = new Thread() {
        public void run() {
            s_backgroundFlushTestRunning = true;

            for(int i = 0; i <= iterations; ++i) {
                System.out.println("executing " + i +
                      " of " + iterations + " iterations.");

                try
                {
                    /* update a property value */
                    if(i == 0) {
                        setStatus("thread starting");
                        addNote();  // add a note
                    }
                    else if( i == iterations) {
                        setStatus("thread complete");
                        removeNote();  // remove a note
                        s_backgroundFlushTestRunning = false;
                    }
                    else {
                        setStatus("executing " + i + " of " +
                                iterations + " iterations.");

                        synchronized (this)
                        {
                            if(s_notes.size() > 0) {
                                Note n =(Note)s_notes.get(0);

                                n.setTitle("Updated-" +
                                      n.getUid() + " v" + i);
                            }
                        }
                    }
                    AdfmfJavaUtilities.
                             flushDataChangeEvent();
```

```
                }
                catch(Throwable t)
                {
                    System.err.
                    println("Error in bg thread - " + t);
                }

                try {
                     Thread.sleep(delay);
                } catch (InterruptedException ex) {
                    // TODO Auto-generated catch block
                    setStatus("inturrpted " + ex);
                    ex.printStackTrace();
                }
            }
        }
    };

    backgroundThread.start();
}


protected int iterations = 100;
protected int delay      = 750;

protected String   status;

/* --- jdev generated accessors --- */

public void
addPropertyChangeListener(PropertyChangeListener l) {
    propertyChangeSupport.addPropertyChangeListener(l);
}

public void
removePropertyChangeListener(PropertyChangeListener l) {
    propertyChangeSupport.removePropertyChangeListener(l);
}

public void setStatus(String status) {
    String oldStatus = this.status;
    this.status = status;
    propertyChangeSupport.
            firePropertyChange("status", oldStatus, status);
}

public String getStatus() {
    return status;
}

public void setIterations(int iterations) {
    int oldIterations = this.iterations;
    this.iterations = iterations;
    propertyChangeSupport.
            firePropertyChange("iterations",
                               oldIterations, iterations);
}

public int getIterations() {
    return iterations;
```

```
    }

    public void setDelay(int delay) {
        int oldDelay = this.delay;
        this.delay = delay;
        propertyChangeSupport.
                firePropertyChange("delay", oldDelay, delay);
    }

    public int getDelay() {
        return delay;
    }
}
```

The StockTracker sample application provides an example of how data change events use Java to enable data changes to be reflected in the user interface. This sample application is in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

jdev_install`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples`

For more information about sample applications, see Appendix E, "ADF Mobile Sample Applications."

10 Using Web Services

This chapter describes how to integrate a third-party web service into the ADF Mobile AMX application feature implementation.

This chapter includes the following sections:

10.1 Introduction to Using Web Services

Web services allow applications and their features to exchange data and information through defined application programming interfaces. Using web services you can expose business functionality irrespective of the platform or language of the originating application. For more information, see the "Introduction to Web Services in Fusion Web Applications" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

ADF Mobile supports both SOAP and REST web services and allows you to integrate a third-party web service into your ADF Mobile AMX application feature.

10.2 Creating a Web Service Data Control

The most common way of using web services in an application feature developed with ADF Mobile is to create a data control for a web service. Typically, this is done for the following reasons:

- To add functionality that is readily available as a web service, but which would be time-consuming to develop within the application.
- To provide access to an application that runs on a different architecture.
- To enable reuse of components created by ADF Mobile to make them available as web services for other applications.

For more information about web service data controls and their usage, see the following:

- "What You May Need to Know About Web Service Data Controls" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- Section on REST web service data control in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- "Data Controls in Oracle ADF Fusion Web Applications" appendix in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- Section 7.3.2.4, "Adding Data Controls to the View"
- "Using the Data Controls Panel" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

10.2.1 How to Create a Web Service Data Control Using SOAP

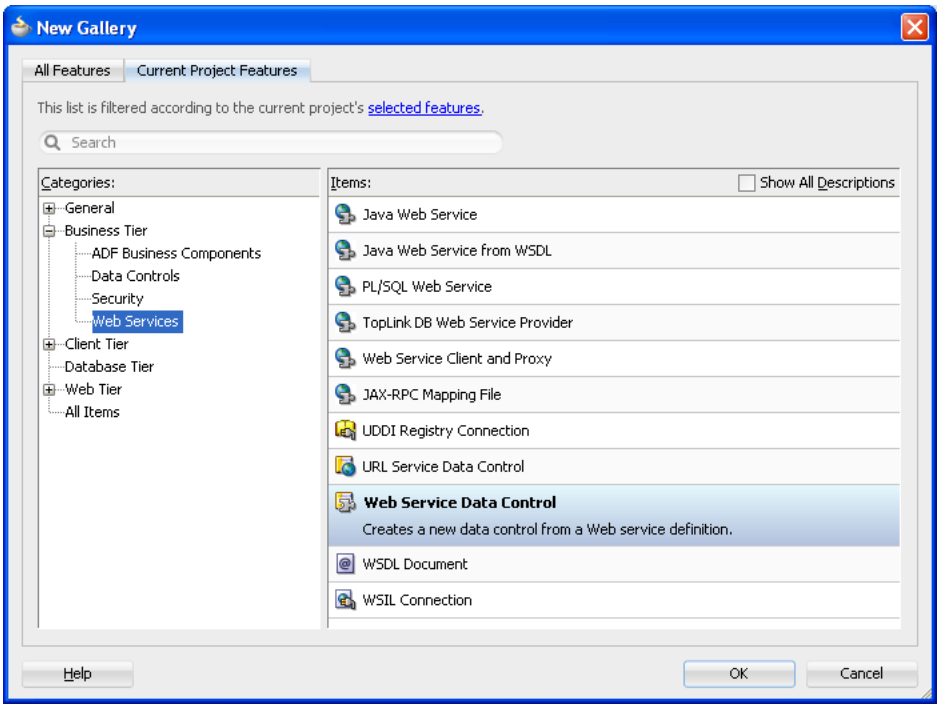
JDeveloper lets you create a data control for an existing SOAP web service using only the Web Services Description Language (WSDL) file for the service. You can either browse to a WSDL file on the local file system, locate one in a Universal Description, Discovery and Integration (UDDI) registry, or enter the WSDL URL directly.

Note: If you are working behind a firewall and you want to use a web service that is outside the firewall, you must configure the Web Browser and Proxy settings in JDeveloper. For more information, see the "Setting Browser Proxy Information" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To create a SOAP web service data control:

1. In the Application navigator, right-click the application name, and then select **File > New** from the main JDeveloper menu.
2. In the **New Gallery** dialog, expand the **Business Tier** node on the left and select **Web Services**. From the **Items** list on the right select **Web Service Data Control** (see Figure 10–1), and then click **OK**.

Figure 10–1 Creating a New SOAP Web Service Data Control



3. Follow the wizard instructions to complete creation of the data control.

Note: ADF Mobile supports the following encoding styles for both SOAP 1.1 and 1.2 versions:

- Document/literal
- Document/wrapped
- RPC

After the web service data control has been created, the web service operations and return values of the operations are displayed in the Data Control palette allowing you to drag and drop the objects returned by the web service operations as appropriate ADF Mobile AMX UI components in the page. For more information, see Section 7.3.2.4, "Adding Data Controls to the View." When data returned from a web service operation is displayed, the following object types are handled:

- Collections.
- Complex objects returned by a web service operation.
- Nested complex objects returned by a web service operation.

Using a web service operation, data can be updated (including both the standard data types and complex objects) and deleted.

10.2.2 How to Create a Web Service Data Control Using REST

JDeveloper lets you create a data control for an existing REST web service.

Note: If you are working behind a firewall and you want to use a web service that is outside the firewall, you must configure the Web Browser and Proxy settings in JDeveloper. For more information, see the "Setting Browser Proxy Information" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

You can associate a REST web service data control (also known as URL service data control) with one or more HTTP methods using the same connection. You should be able to access custom operations exposed by a REST service beyond the standard HTTP methods. These custom operations map to one of the HTTP methods and allow you to create a data control to expose these custom operations on the client.

To use security and notifications functionality on mobile devices, you can add custom headers and custom values to standard HTTP headers for use with specific operations exposed by the REST data control.

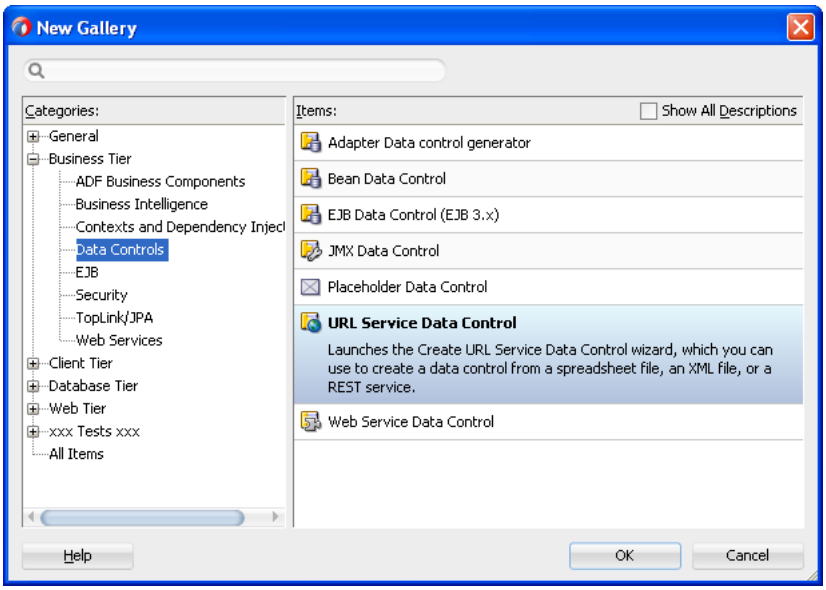
Before you begin:

Ensure that you have access to the REST web service that the data control is to access.

To create a REST web service data control:

1. In the Application navigator, right-click the application name, and then select **File > New** from the main JDeveloper menu.
2. In the **New Gallery** dialog, expand the **Business Tier** node on the left and select **Data Controls**. From the **Items** list on the right, select **URL Service Data Control** (see Figure 10–2), and then click **OK**.

Figure 10–2 Creating a New REST Web Service Data Control



3. Follow the Create URL Service Data Control wizard instructions to complete creation of the data control, keeping in mind the following:

- ADF Mobile supports only basic authentication for web services (see Section 10.5, "Accessing Secure Web Services"). When completing the **Connection** page, select **Basic** in the **Authentication Type** field.
- ADF Mobile supports all HTTP method types: `GET`, `POST`, `PUT`, and `DELETE`. You can select any of these method types when completing the **Http Method** field on the **Connection** page.

> **Note:** You can include all four methods using the same connection and the same REST web service data control.

- ADF Mobile supports all settings listed on the **Parameters** page.
- ADF Mobile does not support the delimiter separated values in the spread sheet data and the XSL for the XML data formats. When completing the **Data Format** page, specify either **XML** or **XSD for the XML** format.
- By specifying a local XSD file for the URL Service data control definition, you create a file URI reference for the **Schema** field.

> **Note:** Since ADF Mobile creates internal definitions for the XSD structures at compile time, the XSD should not change after the application has been compiled. Therefore, it is recommended to reference the XSD file locally. Using the remote XSD negatively affects performance because ADF Mobile retrieves the XSD with each run of the application.

After the REST web services data control has been created by following the preceding steps, it behaves identically to its Oracle ADF counterpart. For more information, see the section on URL services in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For information on how to use REST web services through Java bypassing data controls, see Section 10.6.1, "How to Use REST Web Services Adapter."

10.3 Creating a New Web Service Connection

The connection information for the web service is stored in the `connections.xml` file along with the other connections in your application. You do not need to explicitly create this file, as it is generated in the `.adf/META-INF` directory by the New Web Service Data Control wizard at the time when the web service data control is created (see Section 10.2, "Creating a Web Service Data Control").

You modify the connection settings by editing the `connections.xml` file.

10.4 Adjusting the Endpoint for a Web Service Data Control

After creating a web service data control, you can modify the endpoint of the URI. This is useful in such cases as when you migrate the feature from a test to production environment.

You modify the endpoint by editing the `connections.xml` file.

10.5 Accessing Secure Web Services

ADF Mobile supports both secured and unsecured web services, as well as basic authentication (`BASIC_AUTH`) over HTTP and HTTPS.

For more information, see the following:

- Section 17.3, "Introduction to Authentication"
- Chapter 17, "ADF Mobile Application Security"
- Section about configuring policies in *Oracle Fusion Middleware Administering Web Services*

To access secured web services from your ADF Mobile application, you may need to configure web service data controls included in the application.

10.5.1 How to Enable Access to SOAP-Based Web Services

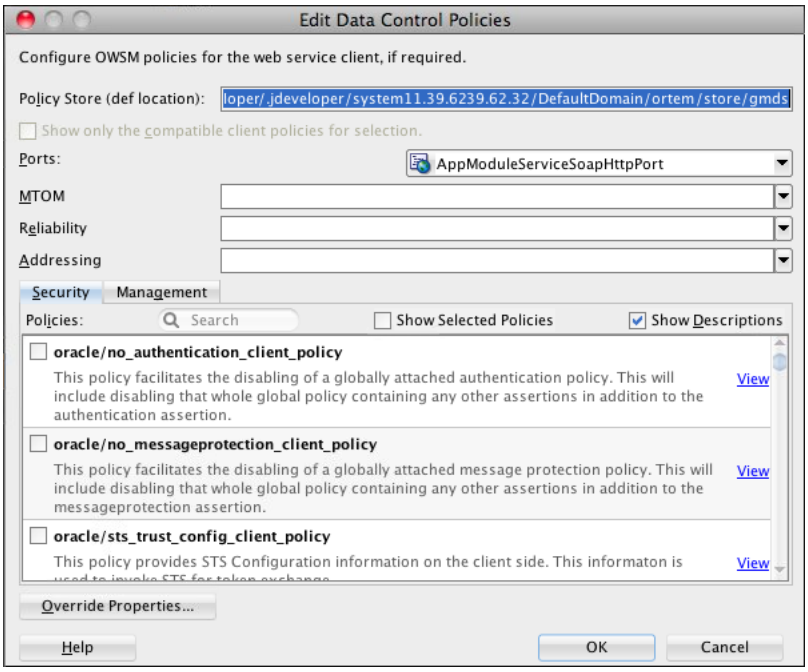
The following security policies are supported for SOAP-based web services:

- `oracle/wss_http_token_client_policy`
- `oracle/wss_http_token_over_ssl_client_policy`
- `oracle/http_basic_auth_over_ssl_client_policy`
- `oracle/wss_username_token_client_policy`
- `oracle/wss_username_token_over_ssl_client_policy`

If a SOAP web service is secured, you can access it by configuring the web service data control with either `oracle/wss_http_token_over_ssl_client_policy` or `oracle/wss_http_token_client_policy`. To do so, use the Edit Data Control Policies dialog that Figure 10.3 shows. You can open this dialog as follows:

- In the Navigator, select the `.dcx` file.
- In the Structure pane, right-click the web service data control that you would like to configure, and then select Define Web Service Security from the context menu.

Figure 10–3 Editing Web Service Data Control Policies



10.5.2 How to Enable Access to REST-Based Web Services

Since only one security policy is supported for REST-based web services, ADF Mobile automatically adds `oracle/wss_http_token_over_ssl_client_policy` for REST web service over HTTPS, or `oracle/wss_http_token_client_policy` for REST web service over HTTP protocol to enable Oracle Web Services Manager (OWSM) Lite Mobile ADF Application Agent to inject a proper security header.

10.5.3 What You May Need to Know About Credential Injection

For secured web services, the user credentials are dynamically injected into a web service request at the time when the request is invoked. This process is similar for SOAP and REST web services.

ADF Mobile uses Oracle Web Services Manager (OWSM) Lite Mobile ADF Application Agent to create and configure proxies, as well as to request services through the proxies. The user credentials are injected into the OWSM enforcement context when proxies are configured. The credential injection is handled by OWSM proxies. For more information, see *Oracle Fusion Middleware Java API Reference for Oracle Web Services Manager*.

Before web services are invoked, the user must respond to an authentication prompt triggered by the user trying to invoke a secured ADF Mobile application feature or to start the application controlled by the access control service (ACS). In the latter case, the application must define a default login server with ACS URL, as well as to have at least one feature with a constraint that depends on the `user.role` setting. The user credentials are stored in a credential store called IDM Mobile credential store—a device native and local repository used for storing credentials associated with the authentication provider's server URL and the user. At run time, ADF Mobile assumes that all credentials have already been stored in the IDM Mobile credential store before the time of their usage.

ADF Mobile supports authentication against the web service endpoint URL only. In the `connections.xml` file, you have to specify the login server connection name in the `credentialStoreKey` attribute of the web service connection reference in order to associate the login server to the web service security (see Example 10–1 and Example 10–2).

> **Note:** To edit the `connections.xml` file, use the Source editor as this configuration is not exposed in the JDeveloper's Designer.

Example 10–1 shows the definition of the web service connection referenced as `credentialStoreKey="MyAuth"`, where `MyAuth` is the name of the login connection reference.

Example 10–1 Defining the Web Service Connection

```
<Reference name="URLConnection1"
           className="oracle.adf.model.connection.url.HttpURLConnection"
           credentialStoreKey="MyAuth"
           xmlns="">
   <Factory className="oracle.adf.model.connection.url.URLConnectionFactory"/>
   <RefAddresses>
      <XmlRefAddr addrType="URLConnection1">
         <Contents>
            <urlconnection name="URLConnection1"
                           url="http://myhost.us.example.com:7777/
                                SecureRESTWebService1/Echo">
               <authentication style="challange">
                  <type>basic</type>
                  <realm>myrealm</realm>
               </authentication>
            </urlconnection>
         </Contents>
      </XmlRefAddr>
      <SecureRefAddr addrType="username"/>
      <SecureRefAddr addrType="password"/>
   </RefAddresses>
</Reference>
```

Example 10–2 shows the definition of the login connection, where `MyAuth` is used as the credential store key value in the login server connection.

Example 10–2 Defining the Login Connection

```
<Reference name="MyAuth"
           className="oracle.adf.model.connection.adfmf.LoginConnection"
           adfCredentialStoreKey="MyAuth"
           partial="false"
           manageInOracleEnterpriseManager="true"
           deployable="true"
           xmlns="">
   <Factory className="oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
   <RefAddresses>
      <XmlRefAddr addrType="adfmfLogin">
         <Contents>
            <login url="http://172.31.255.255:7777/
                        SecuredWeb1-ViewController-context-root/faces/view1.jsf"/>
            <logout url="http://172.31.255.255:7777/
                    /SecuredWeb1-ViewController-context-root/faces/view1.jsf"/>
```

```
            <accessControl url="http://myhost.us.example.com:7777/
                               UserObjects/jersey/getUserObjects" />
            <idleTimeout value="10"/>
            <sessionTimeout value="36000"/>
            <cookieNames>
               <cookie name="JSESSIONID"/>
            </cookieNames>
            <userObjectFilter>
               <role name="testuser1_role1"/>
               <role name="testuser2_role1"/>
               <privilege name="testuser1_priv1"/>
               <privilege name="testuser2_priv1"/>
               <privilege name="testuser2_priv2"/>
            </userObjectFilter>
         </Contents>
      </XmlRefAddr>
   </RefAddresses>
</Reference>
```

If a web service request is rejected due to the authentication failure, ADF Mobile returns an appropriate exception and invokes an appropriate action (see Section 18.5, "Using and Configuring Logging"). If none of the existing exceptions correctly represent the condition, a new exception is added.

The `connections.xml` file is deployed and managed under the Configuration Service. For more information, see Section 10.7, "Administering Web Services."

`connection.xml` files in FARs are aggregated when the ADF Mobile application is deployed. The credentials represent deployment-specific data and are not expected to be stored in FARs.

10.5.4 Limitations of Secure WSDL File Usage

Since an ADF Mobile application must make a WSDL file accessible at run time without authentication, you cannot use a secure WSDL file with a SOAP web service secured by the basic authentication.

If your intention is to secure the WSDL, consider the following: since the WSDL file is fetched by the `GET` method of the web service, if you secure each web service method, except the `GET` method, you can use a secure WSDL. If you secure the `GET` method, you should not secure the WSDL.

10.6 Invoking Web Services From Java

In your ADF Mobile application, you can invoke the web services layer (both REST and SOAP) from the Java code and use the results in Java methods.

ADF Mobile provides the `GenericTypeBeanSerializationHandler` utility class that you can use to perform conversions between POJOs (JavaBeans objects) and ADF Mobile's `GenericType` objects based on the following set of conversion rules:

1. When converting from POJO to `GenericType` objects:
   - Standard JavaBeans reflection rules are used for determining properties.
   - Transient properties are ignored in the conversion process.
   - Readable properties are converted into a `GenericType` attribute.
   - Array properties are represented as repeated attributes in the `GenericType`.

- Map properties are represented as individual attributes in the `GenericType`.
- Non-primitive properties are represented as nested `GenericType` objects.

2. When converting from `GenericType` objects to POJO:
   - Standard JavaBeans reflection rules are used for determining properties.
   - Transient properties are ignored in the conversion process.
   - Writable properties are converted from `GenericType` attributes.
   - Repeated attributes in the `GenericType` are converted into an array object.
   - If the POJO implements the `Map` interface, then any properties that cannot be set through standard accessors are set in the POJO through the `set` method of the `Map`.
   - Non-primitive attributes are represented as nested POJO objects.

The advantage of using this helper API is that it allows you to take the response received from a web service and convert it to a JavaBean in a single call.

For example, a web service passes back and forth an `Employee` object that needs to be reused throughout the business logic. This object has the following set of properties:

- `name` of type `String`
- `address`: a complex object with `street`, `city`, `state`, and `zipcode` attributes
- `id` of type `long`
- `salary` of type `float`
- `phone` of type `String`, and there could be more than one phone
- `password` of type `String`, and the password should never be transmitted to the back-end web service

Example 10–3 shows a potential code for the `Employee` object.

Example 10–3 Employee Object

```
public class Employee {

   protected String name;
   protected Address address;
   protected long id;
   protected float salary;
   protected String[] phone;
   protected transient String password;

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public Address getAddress() {
      return address;
   }

   public void setAddress(Address address) {
      this.address = address;
```

```
    }

    public long getId() {
       return id;
    }

    public void setId(long id) {
       this.id = id;
    }

    public float getSalary() {
       return salary;
    }

    public void setSalary(float salary) {
       this.salary = salary;
    }

    public String getPassword() {
       return password;
    }

    public void setPassword(String password) {
       this.password = password;
    }

    public void setPassword(String password) {
       this.password = password;
    }

    public String[] getPhone() {
       return phone;
    }

    public void setPhone(String phone) {
       this.phone = phone;
    }
```

Example 10–4 shows the potential code for the `Address` object of the `Employee` class.

Example 10–4 Address Object

```
public class Address {

    protected String street;
    protected String city;
    protected String state;
    protected String zipcode;

    public String getStreet() {
       return street;
    }

    public void setStreet(String street) {
       this.street = street;
    }

    public String getCity() {
       return city;
    }
```

```
public void setCity(String city)  {
   this.city = city;
}

public String getState() {
   return state;
}

public void setState(String state) {
   this.state = state;
}

public String getZipcode() {
   return zipcode;
}

public void setZipcode(String zipcode)  {
   this.zipcode = zipcode;
}
```

Keeping in mind the conversion rules, note the following:

1. Since the `password` is defined as transient, it is ignored with respect to the conversion algorithm.
2. Since `name`, `address`, `id`, and `salary` all have `get` and `set` methods, they will all be converted to and from the `GenericType`.
3. Based on the property type, properties can be coerced between types, as defined in the `coerceToType(Object, Class)` method of the `oracle.adfmf.misc.Converter` class.
4. Complex objects, such as `address`, are recursed by the conversion algorithm to either build the child `GenericType` or to create and populate the POJO complex object depending on the direction of the conversion.
5. Since `phone` is an array of `String` objects each representing a unique phone number and since the cardinality of this element is greater than one, the conversion algorithm will find all matches of the `phone` attribute in the `GenericType` object, present them as an array, and invoke the `setPhone` method on the bean. The `toGenericType` method of the `GenericTypeBeanSerializationHandler` will take each array element and append it to the `toGenericType` as an individual `phone` attribute.

With the following defined:

```
final String EMPLOYEE_VIRTUAL_BEAN_NAME = "EmployeeDC.Types.Employee";
Employee emp = getEmployee();
GenericType gt = null;
```

- The Employee object is converted to the `GenericType` as:

```
gt = GenericTypeBeanSerializationHelper.toGenericType
                     (EMPLOYEE_VIRTUAL_BEAN_NAME, emp);
```

- The `GenericType` is converted to the Employee object as:

```
emp = GenericTypeBeanSerializationHelper.fromGenericType
                      (Employee.class, gt, null);
```

For successful conversion, consider the following:

- Typically, POJOs closely follow their associated `GenericType` structure.
- When deviating from the `GenericType` structure, one of the following strategies should be followed:
  1. Additional bean properties should be declared transient.
  2. Optional properties can be excluded from the POJO, provided that the backing service can handle the missing data if used as an operation parameter.
- The `GenericType` is only exposed in SOAP data controls. The virtual types have an associated virtual bean name that is passed into the `toGenericType` method. You can access the virtual bean name by hovering over the virtual type in the Data Controls window of JDeveloper. The typical name format is *`<DCName>.Types.<methodName>.<argName>`*.

For more information, see *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile*.

10.6.1 How to Use REST Web Services Adapter

You can use the `RestServiceAdapter` interface to access data (that could be presented as JavaScript Object Notation, for example) sent across a REST call. The `RestServiceAdapter` interface lets you trigger execution of web service operations without the need to create a web service data control or interact with it directly.

To use the `RestServiceAdapter` interface in your ADF Mobile application, ensure that the connection exists in the `connections.xml` file (see Section 10.3, "Creating a New Web Service Connection"), and then add your code to the bean class, as the following examples show.

Example 10–5 demonstrates the use of the `RestServiceAdapter` for the `GET` request.

Example 10–5 Using RestServiceAdapter for GET Request

```
RestServiceAdapter restServiceAdapter = Model.createRestServiceAdapter();

// Clear any previously set request properties, if any
restServiceAdapter.clearRequestProperties();

// Set the connection name
restServiceAdapter.setConnectionName("RestServerEndpoint");

// Specify the type of request
restServiceAdapter.setRequestType(RestServiceAdapter.REQUEST_TYPE_GET);

// Specify the number of retries
restServiceAdapter.setRetryLimit(0);

// Set the URI which is defined after the endpoint in the connections.xml.
// The request is the endpoint + the URI being set
restServiceAdapter.setRequestURI("/WebService/Departments/100");

String response = "";

// Execute SEND and RECEIVE operation
try {
   // For GET request, there is no payload
   response = restServiceAdapter.send("");
}
```

```
catch (Exception e) {
   e.printStackTrace();
}
```

Example 10–6 demonstrates the use of the `RestServiceAdapter` for the `POST` request.

Example 10–6 Using RestServiceAdapter for POST Request

```
String id = "111";
String name = "TestName111";
String location = "TestLocation111";
RestServiceAdapter restServiceAdapter = Model.createRestServiceAdapter();

restServiceAdapter.clearRequestProperties();
restServiceAdapter.setConnectionName("RestServerEndpoint");
restServiceAdapter.setRequestType(RestServiceAdapter.REQUEST_TYPE_POST);
restServiceAdapter.setRetryLimit(0);
restServiceAdapter.setRequestURI("/WebService/Departments");

String response = "";

// Execute SEND and RECEIVE operation
try {
   String postData = makeDepartmentPost("DEPT", id, name, location);
   response = restServiceAdapter.send(postData);
}
catch (Exception e) {
   e.printStackTrace();
}
System.out.println("The response is:  " + response);

private String makeDepartmentPost(String rootName, String id,
                                  String name, String location) {
   String ret = "<" + rootName + ">";
   ret += "<DEPTID>" + id + "</DEPTID>";
   ret += "<NAME>" + name + "</NAME>";
   ret += "<LOCATION>" + location + "</LOCATION>";
   ret += "</" + rootName + ">";
   return ret;
}
```

Example 10–7 demonstrates the use of the `RestServiceAdapter` for the `PUT` request.

Example 10–7 Using RestServiceAdapter for PUT Request

```
String id = "111";
String name = "TestName111";
String location = "TestLocation111";
RestServiceAdapter restServiceAdapter = Model.createRestServiceAdapter();

restServiceAdapter.clearRequestProperties();
restServiceAdapter.setConnectionName("RestServerEndpoint");
restServiceAdapter.setRequestType(RestServiceAdapter.REQUEST_TYPE_PUT);
restServiceAdapter.setRetryLimit(0);
restServiceAdapter.setRequestURI("/WebService/Departments");

String response = "";
```

```
// Execute SEND and RECEIVE operation
try {
   String putData = makeDepartmentPut("DEPT", id, name, location);
   response = restServiceAdapter.send(putData);
}
catch (Exception e) {
   e.printStackTrace();
}
System.out.println("The response is:  " + response);

private String makeDepartmentPut(String rootName, String id,
                                 String name, String location) {
   String ret = "<" + rootName + ">";
   ret += "<DEPTID>" + id + "</DEPTID>";
   ret += "<NAME>" + name + "</NAME>";
   ret += "<LOCATION>" + location + "</LOCATION>";
   ret += "</" + rootName + ">";
   return ret;
}
```

Example 10–8 demonstrates the use of the `RestServiceAdapter` for the `DELETE` request.

Example 10–8 Using RestServiceAdapter for DELETE Request

```
RestServiceAdapter restServiceAdapter = Model.createRestServiceAdapter();

restServiceAdapter.clearRequestProperties();
restServiceAdapter.setConnectionName("RestServerEndpoint");
restServiceAdapter.setRequestType(RestServiceAdapter.REQUEST_TYPE_DELETE);
restServiceAdapter.setRetryLimit(0);
restServiceAdapter.setRequestURI("/WebService/Departments/44");

String response = "";

// Execute SEND and RECEIVE operation
try {
   // For DELETE request, there is no payload
   response = restServiceAdapter.send("");
}
catch (Exception e) {
   e.printStackTrace();
}

System.out.println("The response is:  " + response);
```

When you use the `RestServiceAdapter`, you should set the `Accept` and `Content-Type` headers to ensure that your request and response payloads are not deemed invalid due to mismatched MIME type.

> **Note:** The REST web service adapter only supports UTF-8 character set on mobile applications. UTF-8 is embedded in the adapter program.

10.6.1.1 Support for Non-Text Responses

You can use the `RestServiceAdapter` to handle binary (non-text) responses received from web service calls. These responses can include any type of binary data,

such as PDF or video files. The `RestServiceAdapter` method to use is `sendReceive`.

Example 10–9 shows how to send a request for a file to a REST server, and then save the file to a disk (see Example 10–10).

Example 10–9 Sending Request for File

```
RestServiceAdapter restServiceAdapter = Model.createRestServiceAdapter();

restServiceAdapter.clearRequestProperties();
restServiceAdapter.setConnectionName("JagRestServerEndpoint");
restServiceAdapter.setRequestType(RestServiceAdapter.REQUEST_TYPE_GET);
restServiceAdapter.setRetryLimit(0);
restServiceAdapter.setRequestURI("/ftaServer/v1/kpis/123/related/1");

// Set credentials needed to access the REST server
String theUsername = "hr_spec_all";
String thePassword = "Welcome1";
String userPassword = theUsername + ":" + thePassword;
String encoding = new sun.misc.BASE64Encoder().encode(userPassword.getBytes());

restServiceAdapter.addRequestProperty("Authorization", "Basic " + encoding);

// Execute the SEND / RECEIVE operation.
// Since it is a GET request, there is no payload.
try {
   this.responseRaw = restServiceAdapter.sendReceive("");
}
catch (Exception e) {
   e.printStackTrace();
}
System.out.println("The response is:  " + this.responseRaw);

// Write the response to a file
writeByteArrayToFile(this.responseRaw);
```

Example 10–10 demonstrates a method that is called by the code from Example 10–9. This method saves a `byte[]` response to a file on disk:

Example 10–10 Saving File to Disk

```
public void writeByteArrayToFile(byte[] fileContent) {
   BufferedOutputStream bos = null;
try {
   FileOutputStream fos = new FileOutputStream(new File(fileToSavePath));
   bos = new BufferedOutputStream(fos);

   // Write the byte [] to a file
   System.out.println("Writing byte array to file");
   bos.write(fileContent);
   System.out.println("File written");
}
catch(FileNotFoundException fnfe) {
   System.out.println("Specified file not found" + fnfe);
}
catch (IOException ioe) {
   System.out.println("Error while writing file" + ioe);
}
finally {
   if(bos != null) {
```

```
      try {
         // Flush the BufferedOutputStream
         bos.flush();

         // Close the BufferedOutputStream
         bos.close();
      }
      catch (Exception e) {
      }
   }
}
```

10.6.2 What You May Need to Know About Invoking Data Control Operations

You can use the `invokeDataControlMethod` method of the `AdfmfJavaUtilities` to invoke a data control operation which does not have to be explicitly added as a `methodAction` in a `PageDef` file.

For more information and examples, see *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile*.

10.7 Administering Web Services

Using the Configuration Service provided by ADF Mobile, administrators can update web service connection endpoints for different ADF Mobile applications from a server, resulting in users obtaining a new connection the first time the ADF Mobile application is started. This is done by hosting and updating multiple `connections.xml` files for different applications. Subsequent application starts require you (the ADF Mobile application developer) to use the container utilities API to check for a new connection (see Section 6.2.3, "checkforNewConfiguration").

Administrators should designate `connections.xml` files as resources protected by basic authentication (see Chapter 17, "ADF Mobile Application Security").

A client application can use the Configuration Service to check for updates.

The Configuration Service dialog allows you to do the following:

- Update the Configuration Service URL and save the update for this application.

  You can seed the Configuration Service URL with a URL value that assists in completing the Configuration Service dialog (see Example 10–11). If that preference ID exists, it is seeded into the Configuration Service dialog.

  The `connections.xml` file must be present at the Configuration Service URL. If the Configuration Service is enabled, this file is obtained and used from the Configuration Server rather than from the application bundle. For more information, see Section 10.7.1, "What You May Need to Know About the URL Construction."

- Enter the user name and password, which are saved in the keystore.

Administrators can use a setting in the `adf-config.xml` file for the application to define whether or not to call the Configuration Service at the application startup time (see Example 10–11). If the application starts for the first time, the dialog is displayed and the credentials are cached in the keystore. On subsequent starts (not including resume), you need to instruct your application to check for modifications against the server and to trigger the display of the Configuration Service dialog. ADF Mobile provides an API whereby you can specify a place in the application where the check for updates should be performed and against which the client API should be called to

invoke the Configuration Service dialog. After the update, the application should run correctly (and be restarted if necessary).

Example 10–11 shows the use of the `use-configuration-service-at-startup` property, whose default value is `false`. In addition, this example shows the `adfmf-configuration-service-seed-url` property that is used to seed the endpoint URL in the user prompt for the Configuration Service connection information.

Example 10–11 Configuration Service Settings in adf-config.xml File

```
<adf:adf-properties-child xmlns="http://xmlns.oracle.com/adf/config/properties">
   <adf-property name="use-configuration-service-at-startup" value="true"/>
   <adf-property name="adfmf-configuration-service-seed-url"
                 value="http://myhost.us.example.com:7777/
                        ConfigServerWebDavFolder/"/>
</adf:adf-properties-child>
```

The Configuration Service performs the following steps when the application starts:

1. It starts at the application startup time.
2. It checks the `Documents` directory for a copy of the managed files. If they are found, the Configuration Service finishes its process.

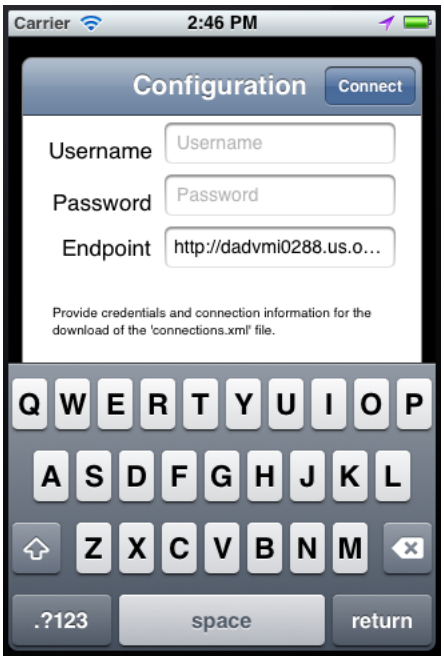
   If the Configuration Service does not locate configuration files in the `Documents` directory, the following takes place:

   - If the `adf-config.xml` file indicates that the Configuration Service is not in use, then the Configuration Service copies the managed files from the application bundle to the `Documents` directory managed folder, and then the Configuration Service finishes its process.
   - If the `adf-config.xml` file indicates that the Configuration Service is in use, then the Configuration Service tries to download the managed files.
3. It checks for stored credentials from the secure store. If the credentials are found and they provide access to the Configuration Server, then the `connections.xml` file is downloaded and placed in the `Documents` directory managed folder, and at this point the Configuration Service finishes its process.

   If the credentials are not found or the stored credentials fail to provide access to the Configuration Server, the user is prompted for the connection information (user name, password, and endpoint URL).
4. The Configuration Service reattempts the prompt or download five times before failing and dropping out of the application.
5. After a successful connection has been made to the Configuration Server, the `connections.xml` file is downloaded and the connection information is stored for later use.

Figure 10–4 shows the Configuration Service prompt dialog on an iPhone.

Figure 10–4 Configuration Service Dialog on iPhone



10.7.1 What You May Need to Know About the URL Construction

The URL to each of the Configuration Service-managed resources is constructed. It contains the application ID and the file name as follows:

If the user provides the URL of `http://my.server.com:port/SomeFolderLocation` and the application ID of `com.mycompany.appname`, the following URL will be used to download the configuration file:

```
http://my.server.com:port/SomeFolderLocation/com.mycompany.appname/connection.xml
```

11

Using the Local Database

This chapter describes how to use the local SQLite database with an ADF Mobile application.

This chapter includes the following sections:

- Section 11.1, "Introduction to the Local SQLite Database Usage"
- Section 11.2, "Using the Local SQLite Database"

11.1 Introduction to the Local SQLite Database Usage

SQLite is a relational database management system (RDBMS) specifically designed for embedded applications.

SQLite has the following characteristics:

- It is ACID-compliant: like other traditional database systems, it has the properties of atomicity, consistency, isolation, and durability.
- It is lightweight: the entire database consists of a small C library designed to be embedded directly within an application.
- It is portable: the database is self-contained in a single file that is binary-compatible across a diverse range of computer architectures and operating systems

For more information, see `http://www.sqlite.org`.

For a sample usage of the local SQLite database, see an ADF Mobile sample application called HR whose data is based on the default HR schema that is provided with all Oracle databases. The data is stored in a local SQLite database and persisted between each startup. The HR application is located in the `PublicSamples.zip` file within the *`jdev_install`*`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples` directory on your development computer

11.1.1 Differences Between SQLite and Other Relational Databases

SQLite is designed for use as an embedded database system, one typically used by a single user and often linked directly into the application. Enterprise databases, on the other hand, are designed for high concurrency in a distributed client-server environment. Because of these differences, there is a number of limitations compared to Oracle databases. Some of the most important differences are:

- Concurrency
- SQL Support and Interpretation

- Data Types
- Database Transactions
- Authentication

For more information, see the Documentation section of the SQLite site at http://www.sqlite.org/docs.html.

11.1.1.1 Concurrency

At any given time, a single instance of the SQLite database may have either a single read-write connection or multiple read-only connections.

Due to its coarse-grained locking mechanism, SQLite does not support multiple read-write connections to the same database instance. For more information, see *File Locking And Concurrency In SQLite Version 3* available from the Documentation section of the SQLite site at http://www.sqlite.org/lockingv3.html.

11.1.1.2 SQL Support and Interpretation

Although SQLite complies with the SQL92 standard, there are a few unsupported constructs, including the following:

- `RIGHT OUTER JOIN`
- `FULL OUTER JOIN`
- `GRANT`
- `REVOKE`

For more information, see *SQL Features That SQLite Does Not Implement* available from the Documentation section of the SQLite site at http://www.sqlite.org/omitted.html.

For information on how SQLite interprets SQL, see *SQL As Understood by SQLite* available from the Documentation section of the SQLite site at http://www.sqlite.org/lang_createtable.html.

11.1.1.3 Data Types

While most database systems are strongly typed, SQLite is dynamically typed and therefore any value can be stored in any column, regardless of its declared type. SQLite does not return an error if, for instance, a string value is mistakenly stored in a numeric column. For more information, see *Datatypes In SQLite Version 3* available from the Documentation section of the SQLite site at http://www.sqlite.org/datatype3.html.

11.1.1.4 Database Transactions

Although SQLite is ACID-compliant and hence supports transactions, there are some fundamental differences between its transaction support and Oracle's:

- Nested transactions: SQLite does not support nested transactions. Only a single transaction may be active at any given time.
- Commit: SQLite permits either multiple read-only connections **or** a single read-write connection to any given database. Therefore, if you have multiple connections to the same database, only the first connection that attempts to modify the database can succeed.
- Rollback: SQLite does not permit a transaction to be rolled back until all open `ResultSets` have been closed first.

For more information, see *Distinctive Features of SQLite* available from the Documentation section of the SQLite site at http://www.sqlite.org/different.html.

11.1.1.5 Authentication

SQLite does not support any form of role-based or user-based authentication. By default, anyone can access all of the data in the file. However, ADF Mobile provides encryption routines that you can use to secure the data and prevent access by users without the valid set of credentials.

11.2 Using the Local SQLite Database

A typical SQLite usage requires you to know the following:

- How to Connect to the Database
- How to Use SQL Script to Initialize the Database or How to Initialize the Database on a Desktop
- How to Encrypt and Decrypt the Database

11.2.1 How to Connect to the Database

Connecting to the SQLite database is somewhat different from opening a connection to an Oracle database. That said, once you have acquired the initial connection, you can use most of the same JDBC APIs and SQL syntax to query and modify the database.

You use the `java.sql.Connection` object associated with your application to connect to the SQLite database. When creating the connection, ensure that every SQLite JDBC URL begins with the text `jdbc:sqlite:`.

Example 11–1 shows how to open a connection to an unencrypted database.

Example 11–1 Connecting to Unencrypted Database

```
java.sql.Connection connection = new SQLite.JDBCDataSource
            ("jdbc:sqlite:/path/to/database").getConnection();
```

Example 11–2 shows how to open a connection to an encrypted database.

Example 11–2 Connecting to Encrypted Database

```
java.sql.Connection connection = new SQLite.JDBCDataSource
            ("jdbc:sqlite:/path/to/database").getConnection(null,"password");
```

In the preceding example, the first parameter of the `getConnection` method is the user name, but since SQLite does not support user-based security, this value is ignored.

Note: SQLite does not display any error messages if you open an encrypted database with an incorrect password. Likewise, you are not alerted if you mistakenly open an unencrypted database with a password. Instead, when you attempt to read or modify the data, an `SQLException` is thrown with the message "Error: file is encrypted or is not a database".

11.2.2 How to Use SQL Script to Initialize the Database

Typically, you can use an SQL script to initialize the database when the application starts. Example 11–3 shows the SQL initialization script that demonstrates some of the supported SQL syntax (described in Section 11.1.1.2, "SQL Support and Interpretation") through its use of the `DROP TABLE`, `CREATE TABLE`, and `INSERT` commands and the `NUMBER` and `VARCHAR2` data types.

Example 11–3 SQL Initialization Script

```
DROP TABLE IF EXISTS PERSONS;

CREATE TABLE PERSONS
(
PERSON_ID NUMBER(15) NOT NULL,
FIRST_NAME VARCHAR2(30),
LAST_NAME VARCHAR2(30),
EMAIL VARCHAR2(25) NOT NULL
);

INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 100,
'David', 'King', 'steven@king.net');
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 101,
'Neena', 'Kochhar', 'neena@kochhar.net');
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 102, 'Lex',
'De Haan', 'lex@dehaan.net');
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 103,
'Alexander', 'Hunold', 'alexander@hunold.net');
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 104,
'Bruce', 'Ernst', 'bruce@ernst.net');
```

To use the SQL script, add it to the ApplicationController project of your ADF Mobile application as a resource. Suppose a sample script has been saved as `initialize.sql` in the `META-INF` directory. Example 11–4 shows the code that you should add to parse the SQL script and execute the statements.

Example 11–4 Initializing Database from SQL Script

```
private static void initializeDatabaseFromScript() throws Exception {
   InputStream scriptStream = null;
   Connection conn = null;
   try {
      // ApplicationDirectory returns the private read-write sandbox area
      // of the mobile device's file system that this application can access.
      // This is where the database is created
      String docRoot = AdfmfJavaUtilities.getDirectoryPathRoot
                               (AdfmfJavaUtilities.ApplicationDirectory);
      String dbName = docRoot + "/sample.db";

      // Verify whether or not the database exists.
      // If it does, then it has already been initialized
      // and no furher actions are required
      File dbFile = new File(dbName);
      if (dbFile.exists())
         return;

      // If the database does not exist, a new database is automatically
      // created when the SQLite JDBC connection is created
      conn = new SQLite.JDBCDataSource("jdbc:sqlite:" + docRoot +
                                       "/sample.db").getConnection();
```

```
      // To improve performance, the statements are executed
      // one at a time in the context of a single transaction
      conn.setAutoCommit(false);

      // Since the SQL script has been packaged as a resource within
      // the application, the getResourceAsStream method is used
      scriptStream = Thread.currentThread().getContextClassLoader().
                           getResourceAsStream("META-INF/initialize.sql");
      BufferedReader scriptReader = new BufferedReader
                                        (new InputStreamReader(scriptStream));
      String nextLine;
      StringBuffer nextStatement = new StringBuffer();

      // The while loop iterates over all the lines in the SQL script,
      // assembling them into valid SQL statements and executing them as
      // a terminating semicolon is encountered
      Statement stmt = conn.createStatement();
      while ((nextLine = scriptReader.readLine()) != null) {
         // Skipping blank lines, comments, and COMMIT statements
         if (nextLine.startsWith("REM") ||
             nextLine.startsWith("COMMIT") ||
             nextLine.length() < 1)
            continue;
         nextStatement.append(nextLine);
         if (nextLine.endsWith(";")) {
            stmt.execute(nextStatement.toString());
            nextStatement = new StringBuffer();
         }
      }
      conn.commit();
   }
   finally {
      if (conn != null)
         conn.close();
   }
}
```

Note: In Example 11–4, the error handling was omitted for simplicity.

You invoke the database initialization code (see Example 11–4) from the `start` method of the `LifeCycleListenerImpl`, as Example 11–5 shows.

Example 11–5 Invoking Database Initialization Code

```
public void start() {
   try {
      initializeDatabaseFromScript();
   }
   catch (Exception e) {
      Trace.log(Utility.FrameworkLogger,
                Level.SEVERE,
                LifeCycleListenerImpl.class,
                "start",
                e);
   }
}
```

11.2.3 How to Initialize the Database on a Desktop

Because SQLite databases are self-contained and binary-compatible across platforms, you can use the same database file on iOS, Android, Windows, Linux, and Mac OS platforms. In complex cases, you can initialize the database on a desktop using third-party tools (such as MesaSQLite, SQLiteManager, and SQLite Database Browser), and then package the resulting file as a resource in your application.

To use the database, add it to the ApplicationController project of your ADF Mobile application as a resource. Suppose a database has been saved as `sample.db` in the `META-INF` directory. Example 11–6 shows the code that you should add to copy the database from your application to the mobile device's file system to enable access to the database.

Example 11–6 Initializing Database on Desktop

```
private static void initializeDatabase() throws Exception {
   InputStream sourceStream = null;
   FileOutputStream destinationStream = null;
   try {
      // ApplicationDirectory returns the private read-write sandbox area
      // of the mobile device's file system that this application can access.
      // This is where the database is created
      String docRoot = AdfmfJavaUtilities.getDirectoryPathRoot
                               (AdfmfJavaUtilities.ApplicationDirectory);
      String dbName = docRoot + "/sample.db";

      // Verify whether or not the database exists.
      // If it does, then it has already been initialized
      // and no furher actions are required
      File dbFile = new File(dbName);
      if (dbFile.exists())
         return;

      // Since the database has been packaged as a resource within
      // the application, the getResourceAsStream method is used
      scriptStream = Thread.currentThread().getContextClassLoader().
                           getResourceAsStream("META-INF/sample.db");
      destinationStream = new FileOutputStream(dbName);
      byte[] buffer = new byte[1000];
      int bytesRead;
      while ((bytesRead = sourceStream.read(buffer)) != -1) {
         destinationStream.write(buffer, 0, bytesRead);
      }
   }
   finally {
      if (sourceStream != null)
         sourceStream.close();
      if (destinationStream != null)
         destinationStream.close();
   }
}
```

Note: In Example 11–6, the error handling was omitted for simplicity.

You invoke the database initialization code (see Example 11–6) from the `start` method of the `LifeCycleListenerImpl`, as Example 11–7 shows.

Example 11–7 Invoking Database Initialization Code

```
public void start() {
   try {
      initializeDatabase();
   }
   catch (Exception e) {
      Trace.log(Utility.FrameworkLogger,
                Level.SEVERE,
                LifeCycleListenerImpl.class,
                "start",
                e);
   }
}
```

11.2.4 What You May Need to Know About Commit Handling

`Commit` statements are ignored when encountered. Each statement is committed as it is read from the SQL script.

11.2.5 Limitations of the ADF Mobile's SQLite JDBC Driver

The following methods from the `java.sql` package have limited or no support in ADF Mobile:

- The `getByte` method of the `ResultSet` is not supported. If used, this method will throw an `SQLException` when executed.
- The `execute` method of the `Statement` always returns `true` (as opposed to returning `true` only for statements that return a `ResultSet`).

11.2.6 How to Encrypt and Decrypt the Database

ADF Mobile allows you to provide the SQLite database with an initial or subsequent encryption. To do so, you have to establish the database connection (see Section 11.2.1, "How to Connect to the Database"), and then use the following utility method to encrypt the database with a new key:

```
AdfmfJavaUtilities.encryptDatabase(connection, "newPassword");
```

> **Caution:** If you open a database incorrectly (for example, use an invalid password to open an encrypted database), and then encrypt it again, neither the old correct password, the invalid password, nor the new password can unlock the database resulting in the irretrievable loss of data.

In addition to encrypting, you can permanently decrypt the database. To do so, open the encrypted database with the correct password, and then use the following method:

```
AdfmfJavaUtilities.decryptDatabase(connection);
```

Part V

Advanced Topics

Describes how to enable the content derived from remote URLs can access native device controls, how to augment ADF Mobile applications with user preference pages, and how setting constraints can determine how application features display or are accessed.

Part V contains the following chapters:

12

Implementing Application Features as Remote URLs

This chapter describes how application features with content from remote URLs can access (or be restricted from) device services.

This chapter includes the following sections:

- Section 12.1, "Overview of Remote URL Applications"
- Section 12.2, "Overview of Enabling Remote URL Implementations to Access PhoneGap"
- Section 12.3, "Enabling Remote Application to Access Device Services through Whitelists."
- Section 12.4, "Creating Whitelists for Application Components."
- Section 12.5, "Enabling the Browser Navigation Bar on Remote URL Pages"

12.1 Overview of Remote URL Applications

By configuring the content type for an application feature in the overview editor for the `adfmf-feature.xml` file as **Remote URL** as described in Section 5.9.1, "How to Define the Application Content," you create a browser-based application that is served from the configured URL. Such server-hosted applications differ from applications written in ADF Mobile AMX or a platform-specific language such as Objective-C in two ways: they are intended for occasional use and cannot access the device memory or services directly, as these interactions are contingent upon the capabilities of the device browser.

Browser-based applications targeted for smartphones are authored with Apache Trinidad components (described at `http://myfaces.apache.org/trinidad/`) to enable proper rendering of a variety of feature phones and smartphones. An application that runs on a tablet can be created in the same manner as an ADF desktop web application.

Note: Oracle recommends using ADF Mobile browser for application features that derive their content from remote URLs. ADF Mobile browser applications are comprised of JSF pages populated with Apache Trinidad components. For more information, see *Oracle Fusion Middleware Mobile Browser Developer's Guide for Oracle Application Development Framework*.

12.2 Overview of Enabling Remote URL Implementations to Access PhoneGap

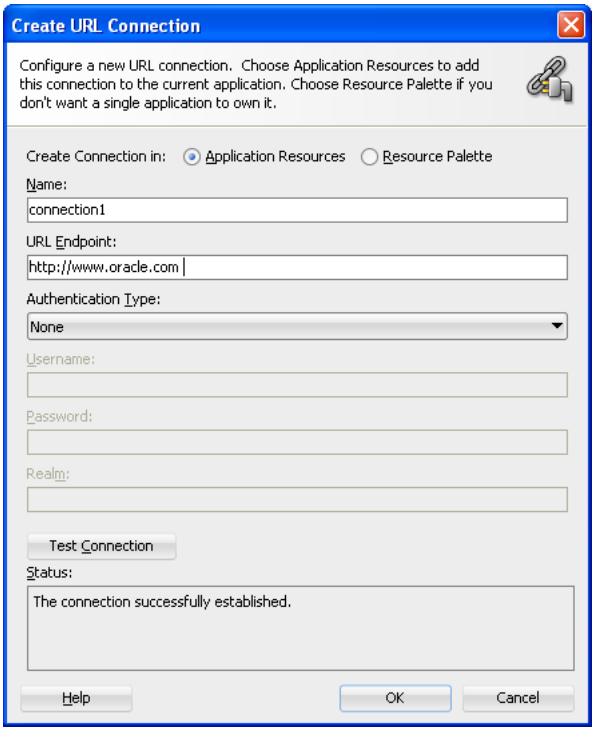
By default, access to such device capabilities as contacts or GPS (Global Positioning Services) are restricted to ADF Mobile applications implemented as a bundle of locally stored HTML pages or as ADF Mobile AMX. In the case of the latter, you can enable the application feature to access device capabilities by adding device data controls to an ADF Mobile AMX UI component. For more information, see in Section 9.5, "Using the DeviceFeatures Data Control." Remote URL implementations, however, can only access device-native controls through PhoneGap JavaScript APIs. As described in Chapter 6, "Controlling the Display of Application Features," the ADF Mobile JavaScript file is the mechanism through which ADF Mobile interfaces with PhoneGap.

12.3 Enabling Remote Application to Access Device Services through Whitelists

ADF Mobile supports the concept of whitelists, a registry of URLs that open within the application web view and can access various device services, such as GPS, a camera, or a file system. If a web page is not included on a whitelist (that is, it is not whitelisted), the ADF Mobile PhoneGap implementation opens a web page in the device browser (such as Safari) instead. Without whitelisting, a remote web page cannot open within a web view.

By default, the domains defined in the `connections.xml` file, the repository for all of the connections defined in the ADF Mobile application, are whitelisted automatically. These connections are defined using the Create URL Connection dialog, shown in Figure 12–4. ADF Mobile parses the domain from each of the connection strings and adds these domains to the whitelist.

Figure 12–1 Creating the Connection to Retrieve the Content of the Remote URL Application Feature



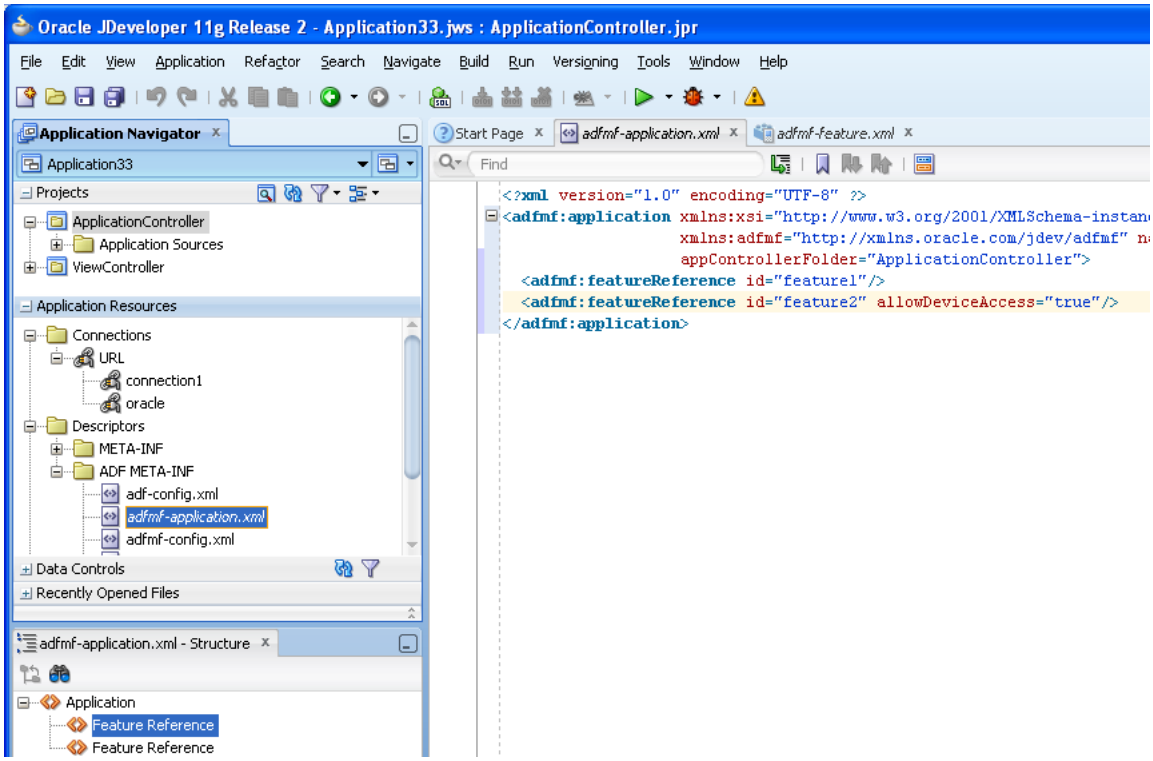
JDeveloper then populates the `connections.xml` file, located in the Application Resources panel, with the connections information and also creates the connection resources, such as *oracle* and *connection 1* in Figure 12–2.

Note: Only whitelisted domains open in the web view; all other (non-whitelisted) domains open in the device browser.

12.3.1 How to Control Access to Device Capabilities

By editing the `adfmf-application.xml` file, shown in Figure 12–2, you can restrict access to the PhoneGap-enabled device services on a per-application feature basis. By default, all application features have such access, meaning that any whitelisted domain configured for the ADF Mobile application can also access the device. In addition to the domains that ADF Mobile includes from the `connections.xml` file, you can also whitelist domains using the Security page of the `adfmf-application.xml` file, as described in Section 12.3.2, "How to Create a Whitelist."

Figure 12–2 Editing <adfmf-FeatureReference> to Limit Domain Access



The Source editor illustrated in Figure 12–2 shows the `<adfmf-feature-reference>` element populated with the `allowDeviceAccess` attribute. JDeveloper populates all application features with remote URL content with this attribute, which by default, is set to `true`. This default setting signifies that all URLs can access the PhoneGap APIs.

Before you begin:

Using the overview editor for the `adfmf-feature.xml` file (located in the Application Navigator in the **Project** panel under the **View Controller** and **META-INF** nodes), you must designate the content for an application as Remote URL and then create the connection as described in Section 5.5.1, "How to Designate the Content for a Mobile Application."

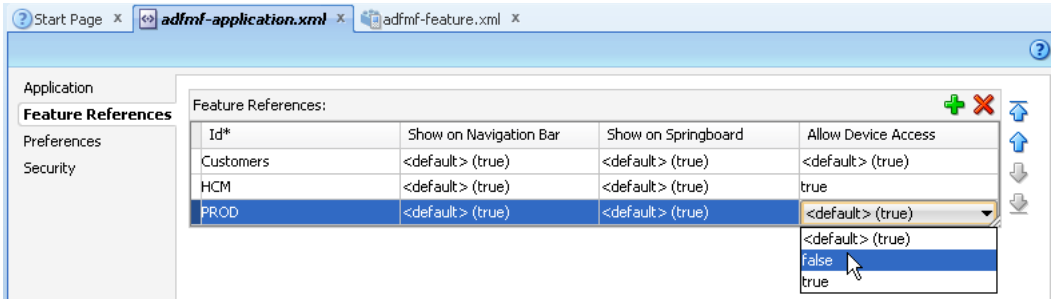
Open the overview editor for the `adfmf-application.xml` file by double-clicking the `adfmf-application.xml` file (located Application Resources panel under the **Resources** and **ADF META-INF** nodes, as shown in Figure 12–2).

You can use the Source editor, Structure Window, and the Properties editor to configure this attribute.

To restrict access to device services:

1. Open the `adfmf-application.xml` file and then open the Feature References page.
2. Using the dropdown menu from the Allow Device Access column of the Feature References table, change the default value (`<default> true`) to `false`, as shown in Figure 12–3.

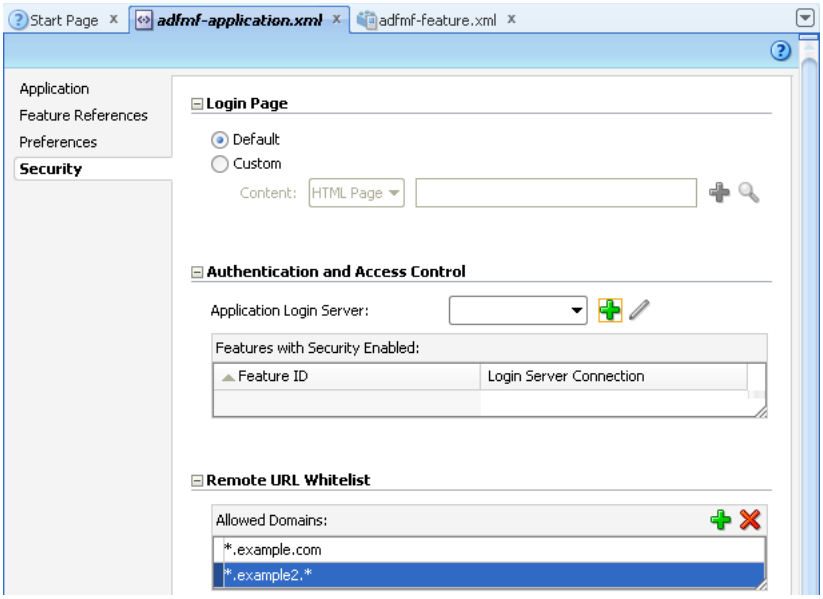
Figure 12–3 Preventing Domains from Accessing Device Services Using the Feature References Page



12.3.2 How to Create a Whitelist

You configure the whitelist in the Security page of `adfmf-application.xml`, shown in Figure 12–4.

Figure 12–4 Configuring a Whitelist



Before you begin:

Be aware that some URLs configured in the ADF Mobile application may open to other domains.

To create whitelists:

1. Open the `adfmf-application.xml` file and then select the Security page.
2. Click **Add** and then enter the domains that can be called from within the web view of the application feature. These domains can include a wildcard (*). For example, `*.example.com` is a valid domain entry as is `*.example.*`. You cannot enter a fully qualified path.

Caution: Entering only the wildcard allows the web view to request all domains and can pose a security risk; adding all domains to the whitelist not only enables all of them to open within the web view, but also enables all of them to access the device (whether or not it is intended for them to do so).

12.3.3 What Happens When You Add Domains to a Whitelist

When you add a domain, JDeveloper updates `<adfmf:remoteURLWhiteList>` element as illustrated in Example 12–1.

Example 12–1 Configuring the Whitelist

```
...
<adfmf:remoteURLWhiteList>
  <adfmf:domain>*.oracle.*</adfmf:domain>
  <adfmf:domain>www.oracle.*</adfmf:domain>
  </adfmf:remoteURLWhiteList>
...
```

12.3.4 What You May Need to Know About Remote URLs

Some URLs are redirected to a URL that may not be part of the whitelist domain. These URLs may open in the device browser rather than the application web view. For example, if you whitelist `www.oracle.com` (`<adfmf:domain>www.oracle.com<adfmf:domain>`) and open that site on the device, ADF Mobile redirects to the mobile version of this site (`www.oracle.mobi`), because it does not pass the whitelist. Figure 12–5 shows a web page that has not been whitelisted and has opened within the device browser.

Figure 12–5 A Web Page Opening in the Device Browser, Not the ADF Mobile Web View

To enable `www.oracle.com` to open within the application web view, you must specify `*.oracle.*` or `www.oracle.*` as shown in Example 12–1.

Because ADF Mobile whitelist is at the domain-level, you cannot restrict an individual page within a whitelisted domain from opening with an application feature web view; all pages are allowed.

12.4 Creating Whitelists for Application Components

Use a whitelist for pages that contain links to URLs that point to another domain. Such pages would otherwise open in the device browser instead of the ADF Mobile web view. In such a case, you can create an anchor tag or an `<amx:goLink>` component with a `url` attribute for the `<amx:goLink>` component that points outside of the application, such as the `url` attribute in `<goLink2>` in Example 12–2.

Example 12–2 <amx:goLink> with a url Parameter

```
<?xml version="1.0" encoding="UTF-8" ?>
<amx:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amx="http://xmlns.oracle.com/adf/mf/amx"
          xmlns:dvtm="http://xmlns.oracle.com/adf/mf/amx/dvt">
  <amx:panelPage id="pp1">
    <amx:panelGroupLayout id="panelGroupLayout1">
      <amx:goLink text="This opens in the device browser"
                  id="golink1"
                  url="http://www.example.com"
                  shortDesc="Opens in device browser"/>
      <amx:goLink text="This opens in the web view"
                  id="golink2"
                  url="http://www.example2.com"
                  shortDesc="Accesses device services"/>
    </amx:panelGroupLayout>
  </amx:panelPage>
</amx:view>
```

See also Section 8.3, "Creating and Using UI Components."

12.5 Enabling the Browser Navigation Bar on Remote URL Pages

ADF Mobile enables you to add a navigation bar with buttons for back, forward, and refresh actions for application features implemented as remotely served web content that open within the ADF Mobile web view, as shown in Figure 12–6. The forward and back buttons are disabled when either navigation forward or back is not possible.

Note: The back button is disabled on Android-powered devices.

Figure 12–6 A Remote Web Page Displaying the Navigation and Refresh Buttons

12.5.1 How to Add the Navigation Bar to a Remote URL Application Feature

You enable users to navigate through, or refresh remote content through the Content tab of the overview editor for the `adfmf-feature.xml` file.

Before you begin:

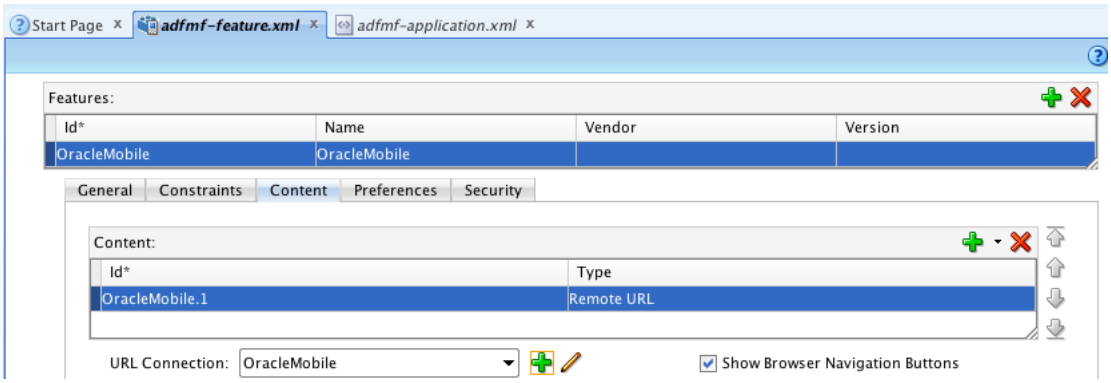
Designate an application feature's content be delivered from a remotely hosted application by first selecting **Remote URL** and then by creating the connection to the host server, as described in Section 5.9, "Defining the Content Types for an Application Feature."

Ensure that the domain is whitelisted.

To enable a navigation bar:

1. Select the Remote URL application feature listed in the Features table in the `adfmf-feature.xml` file.
2. Click **Content**.
3. Select **Show Browser Navigation Buttons**, as shown in Figure 12–7.

Figure 12–7 Selecting Navigation Options



12.5.2 What Happens When You Enable the Browser Navigation Buttons for a Remote URL Application Feature

JDeveloper updates the `adfmf:remoteURL` element with an attribute called `showNavButtons`, which is set to `true`, as shown in Example 12–3.

Example 12–3 The showNavButtons Attribute

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:adfmf="http://xmlns.oracle.com/adf/mf">
  <adfmf:feature id="oraclemobile" name="oraclemobile">
    <adfmf:content id="oraclemobile.1">
      <adfmf:remoteURL connection="connection1"
                       showNavButtons="true"/>
    </adfmf:content>
  </adfmf:feature>
</adfmf:features>
```

After you deploy the application, ADF Mobile applies the forward, back, and refresh buttons to the web pages that are traversed from the home page of the Remote URL application feature, as shown in Figure 12–8.

Figure 12–8 Traversing Through a Remote URL Application Feature

13

Enabling User Preferences

This chapter describes how to create both ADF Mobile application-level and application feature-level user preference pages.

This chapter includes the following sections:

- Section 13.1, "Creating User Preference Pages for an ADF Mobile Application"
- Section 13.2, "Creating User Preference Pages for Application Features"
- Section 13.3, "Using EL Expressions to Retrieve Stored Values for User Preference Pages"
- Section 13.4, "Platform-Dependent Display Differences"

13.1 Creating User Preference Pages for an ADF Mobile Application

Preferences enable you to add settings that can be configured by end users. Within both the `adfmf-application.xml` and `adfmf-feature.xml` files, the user preference pages are defined with the `<adfmf:preferernces>` element. As shown in Example 13–1, the child element of `<adfmf:preferences>` called `<adfmf:preferenceGroup>` and its child elements define the user preferences by creating pages that present options in various forms, such as read-only strings, dropdown menus, or in the case of Example 13–1, as a child page that can present the user with additional options for application settings.

You also use the `<adfmf:preferences>` element to create the preferences that users manage within each application feature.

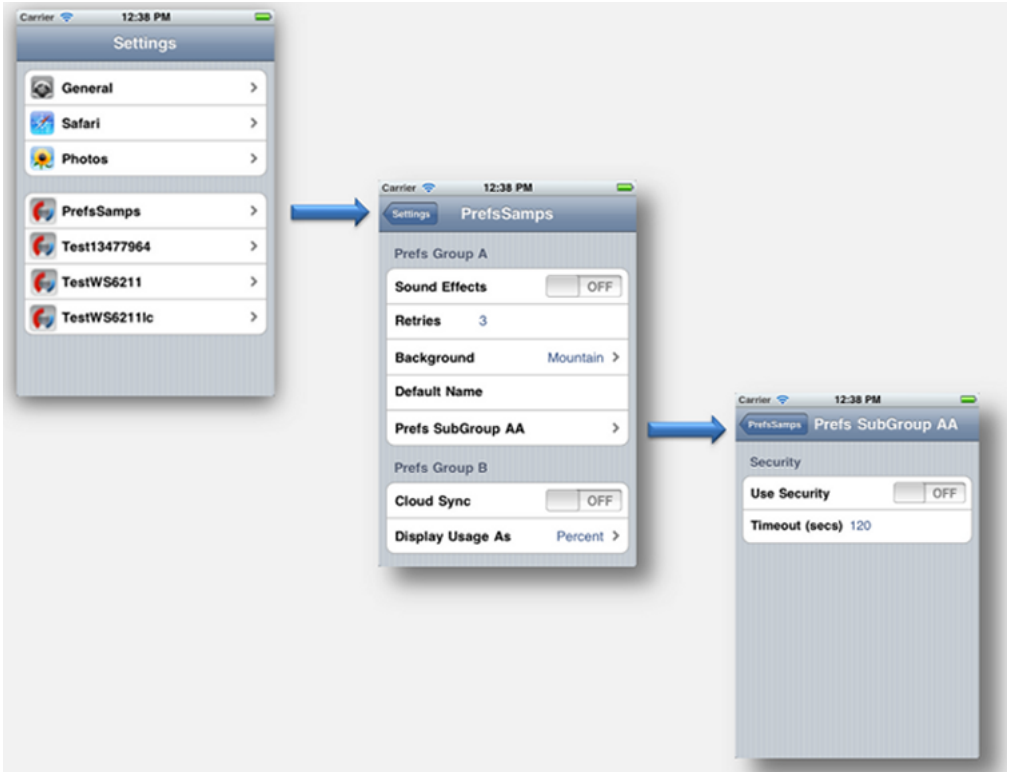
Example 13–1 Defining Application-Level Preferences with the <adfmf:preferences> Element

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:application xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                   xmlns:adfmf="http://xmlns.oracle.com/adf/mf"
                   name="MobileApplication"
                   id="com.company.MobileApplication"
                   appControllerFolder="ApplicationController"
                   version="1"
                   vendor="oracle"
                   listener-class="application.LifeCycleListenerImpl">
  <adfmf:description>This is an ADF Mobile application</adfmf:description>
  <adfmf:featureReference id="PROD"/>
  <adfmf:featureReference id="HCM"/>
  <adfmf:featureReference id="Customers"/>
      <adfmf:preferences>
      <adfmf:preferenceGroup id="a" label="Prefs Group A">
        <adfmf:preferenceBoolean id="a1-sound" label="Sound Effects"/>
```

```
        <adfmf:preferenceNumber id="a2-retries" label="Retries" default="3"/>
        <adfmf:preferenceList id="a3-background" label="Background" default="3">
          <adfmf:preferenceValue name="None" value="0"/>
          <adfmf:preferenceValue name="Field" value="1"/>
          <adfmf:preferenceValue name="Galaxy" value="2"/>
          <adfmf:preferenceValue name="Mountain" value="3"/>
        </adfmf:preferenceList>
        <adfmf:preferenceText id="a4-name" label="Default Name"/>
        <adfmf:preferencePage id="aa" label="Prefs SubGroup AA">
          <adfmf:preferenceGroup id="aa-sec" label="Security">
            <adfmf:preferenceBoolean id="aa-sec-useSec" label="Use Security"/>
            <adfmf:preferenceNumber id="aa-sec-timeout" label="Timeout (secs)" default="120"/>
          </adfmf:preferenceGroup>
        </adfmf:preferencePage>
      </adfmf:preferenceGroup>
      <adfmf:preferenceGroup id="b" label="Prefs Group B">
        <adfmf:preferenceBoolean id="b-cloudSync" label="Cloud Sync"/>
        <adfmf:preferenceList id="b-dispUsage" label="Display Usage As" default="1">
          <adfmf:preferenceValue name="Percent" value="1"/>
          <adfmf:preferenceValue name="Minutes" value="2"/>
        </adfmf:preferenceList>
      </adfmf:preferenceGroup>
    </adfmf:preferences>
</adfmf:application>
```

Figure 13–1 shows an example of how opening child user preferences page can offer subsequent options.

Figure 13–1 User Preferences Pages



Preference pages are defined within the `<adfmf:preferenceGroup>` element and have the following child elements:

- `<adfmf:preferencePage>`—Specifies a new page in the user interface.
- `<adfmf:preferenceList>`—Provides users with a specific set of options.
  - `<adfmf:preferenceValue>`—A child element that defines a list element.
- `<adfmf:preferenceBoolean>`—A boolean setting.
- `<adfmf:preferenceText>`—A text preference setting.

See *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile Tag Documentation* for more information on these elements and their attributes.

For an example of creating preference pages at both the ADF Mobile application and application-feature levels, refer to the PrefDemo sample application. This sample application is located in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

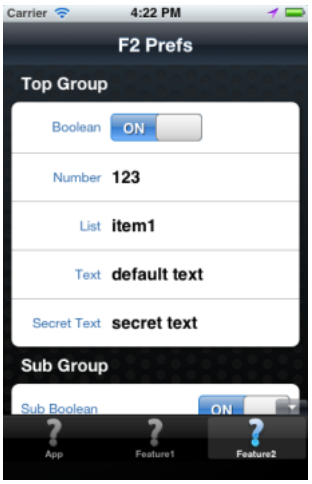
jdev_install`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples`

The PrefDemo application is comprised of an application-level settings page as well as three application feature preference pages, which are implemented as ADF Mobile AMX. Figure 13–2 shows the PrefDemo application settings page, which you invoke from the general settings page. In this illustration, the preference settings page is invoked from the iOS Settings application.

Figure 13–2 The PrefDemo Application Settings Page

The application feature preference pages, illustrated by *App*, *Feature1* (which is selected), and *Feature 2* in Figure 13–3, provide examples of preferences pages constructed from the ADF Mobile Boolean Switch, Input Text, and Output Text components that use EL (Expression Language) to access the application feature and the various `<adfmf:preferences>` components configured within it. For more information, see Section 13.3, "Using EL Expressions to Retrieve Stored Values for User Preference Pages."

Figure 13–3 An Application Feature Preference Page from PrefDemo

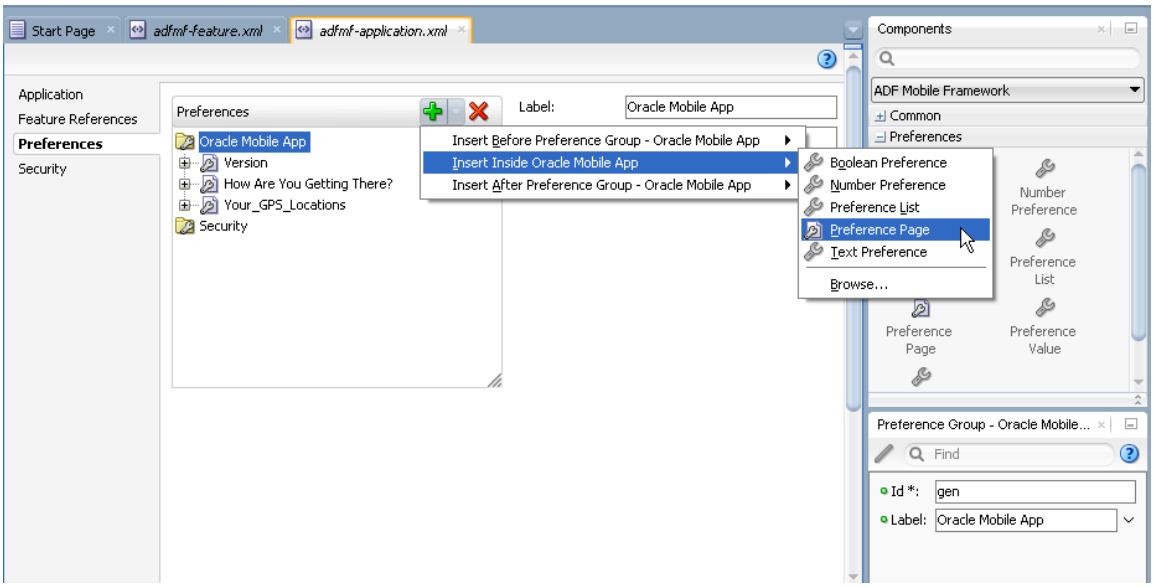


In the PrefDemo application, each ADF Mobile AMX preference page is referenced by a single bounded task flow comprised of a view activity and a control flow case that enables the page refresh.

13.1.1 How to Create Mobile Application-Level Preferences Pages

The Preferences page of the `adfmf-application.xml` overview editor, shown in Figure 13–4, enables you to build sets of application-level preference pages by nesting the child preference page elements within `<adfmf:preferenceGroup>`. The page presents the `<adfmf:preferencesGroup>` and its child elements as similarly named options (that is, Preference Group, Preference Page, Preference List, and so on), which you assemble into a hierarchy (or tree), similar to the Structure window in JDeveloper.

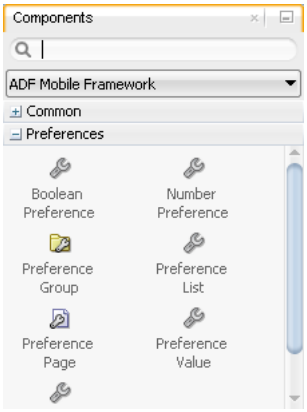
Figure 13–4 Adding Mobile Application-Level Preferences Using the Preference Page



To ensure that the `adfmf-application.xml` file is well-formed, use the Preferences page's **Add** drop-down list, shown in Figure 13–4 to construct the user preferences pages. While you can also drag components from the Preferences palette, shown in

Figure 13–5, into either the editor, the Source window, or the Structure window, the page's drop-down list presents only the elements that can have the appropriate parent, child, or sibling relationship to a selected preferences element. For example, Figure 13–4 shows only the components that can be inserted within the Preference Group element, *Oracle Mobile App*. The editor also enables you to enter the values for the attributes specific to each preference element.

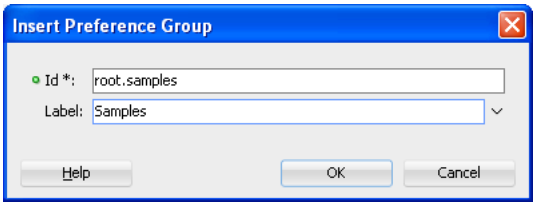
Figure 13–5 Preferences in the Component Palette



To create preferences pages:

1. Click **Preferences**.
2. Click **Add** to create the parent `<adfmf:preferenceGroup>` element.
3. Enter the following information in to the Insert Preference Group dialog, shown in Figure 13–6.

Figure 13–6 Defining the Parent Preference Group Element



   - Enter a unique identifier for the Preference Group element.
   - Enter the descriptive text that displays in the user interface. For an example of how this text displays in the user interface, see *Sample* in Figure 13–1.
4. Click **Add** to further define the preference pages using the **Insert Before**, **Insert Inside**, **Insert After** options to ensure that the XML document is well formed.

13.1.1.1 How to Create a New User Preference Page

The Preference Page component enables you to create a new user interface page. You create a Preference Page using the **Insert Before**, **Insert Inside**, **Insert After** options.

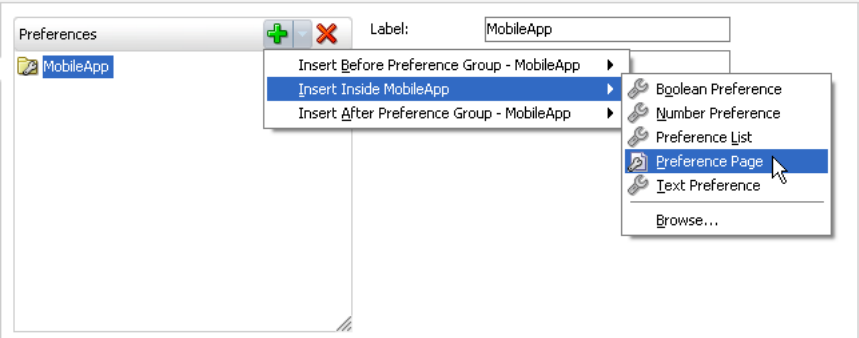
Before you begin:

You must create a Preferences Group element.

To create a new user preference page:

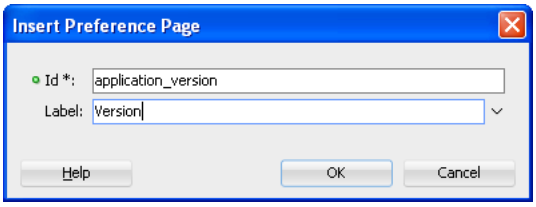
1. Select the Preference Group element.
2. Click **Add**, choose **Insert Inside** (Preference Group), then select **Preference Page**. As shown in Figure 13–7, the Preference Group is called *MobileApp*.

Figure 13–7 Selecting the Preference Page Component



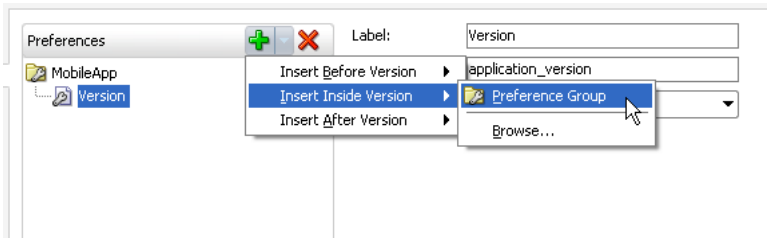
3. Define the following Preference Page attributes in the Insert Preference Page dialog, shown in Figure 13–8:
   - Enter a unique identifier for the Preference Page element.
   - Enter the descriptive text that displays in the user interface.

Figure 13–8 The Insert Preference Page Dialog



4. Create the body of the preference page by inserting a child Preference Group element by selecting the Preference Page, and then first choosing **Insert Inside** (Preference Page) and then **Preference Group**, as shown in Figure 13–9. After you define a unique identifier and display name for the child Preference Group, you can populate it with other elements, such as a Preference List element, as shown in Example 13–2.

Figure 13–9 Adding a Preference Group to a Preference Page



13.1.1.2 What Happens When You Add a Preference Page

After you define the Preference Page and its child Preference Group components in the overview editor, JDeveloper generates an `<adfmf:preferencePage>` with

attributes similar to Example 13–2. The `<adfmf:preferencePage>` is nested within a parent `<adfmf:preferenceGroup>` element.

Example 13–2 Adding an <adfmf:PreferencePage element>

```
<adfmf:preferences>
          <adfmf:preferenceGroup id="gen" label="Oracle Mobile App">
               <adfmf:preferencePage id="application_version" label="Version">
                    <adfmf:preferenceGroup id="version_select" label="Select Your Version">
                         <adfmf:preferenceList id="edition" label="Edition" default="PERSONAL">
                              <adfmf:preferenceValue name="Enterprise" value="ENTERPRISE"/>
                              <adfmf:preferenceValue name="Personal" value="PERSONAL"/>
                         </adfmf:preferenceList>
                    </adfmf:preferenceGroup>
               </adfmf:preferencePage>
     </adfmf:preferences>
```

13.1.1.3 How to Create User Preference Lists

Add a Preference List component to create a list of options.

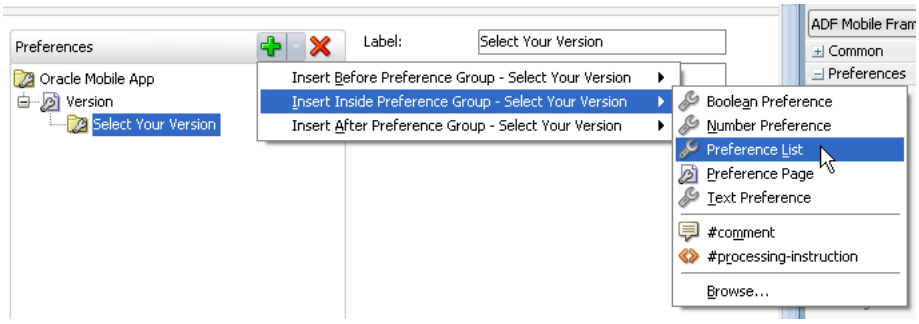
Before you begin:

You must create Preference Group as the parent to the Preference List or any other list-related component.

To create a user preference list:

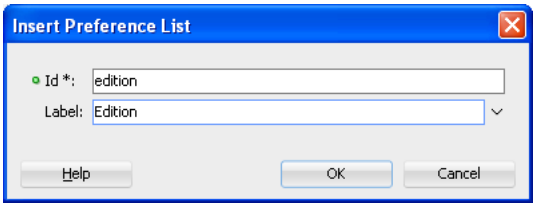
1. Select a Preference Group or Preference Page and then click **Add**, then Insert Inside, and then **Preference List**. Example 13–10 shows adding a Preference List as a child of a Preference Group component called *Select Your Version*.

Figure 13–10 Adding a Preference List Component to a Preference Group



2. Define the following attributes using the Insert Preference List dialog, shown in Example 13–11, and then click **OK**.
   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.

Figure 13–11 The Insert Preference List Dialog

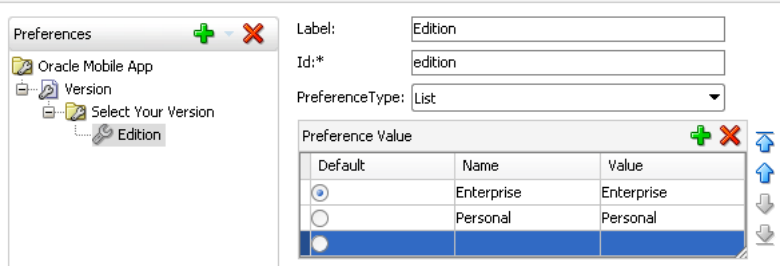


3. Define a list of read-only items by clicking **Add** in the Preference Value table, shown in Figure 13–12. You can also remove a preference value by selecting it and then clicking **Delete**. You can change the order in which the preference values display by selecting the preference value and then using the up- and down-arrows.

   You can present the user with a default setting by choosing **Default**. As illustrated in Example 13–2, the default status is defined within the `<adfmf:preferenceList>` element as `default="ENTERPIRSE"`.

   Tip: In addition to clicking **Add**, you can add Preference Value components by dragging them either into the Structure window or the Source window.

Figure 13–12 Adding Preference Values



13.1.1.4 What Happens When You Create a Preference List

After you add Preference List component to a Preference Group and then define a series of Preference Values, JDeveloper updates the `<adfmf:preferences>` section with an `<adfmf:preferenceList>` element, as shown in Example 13–2.

13.1.1.5 How to Create a Boolean Preference List

See, for example, Example 13–1.

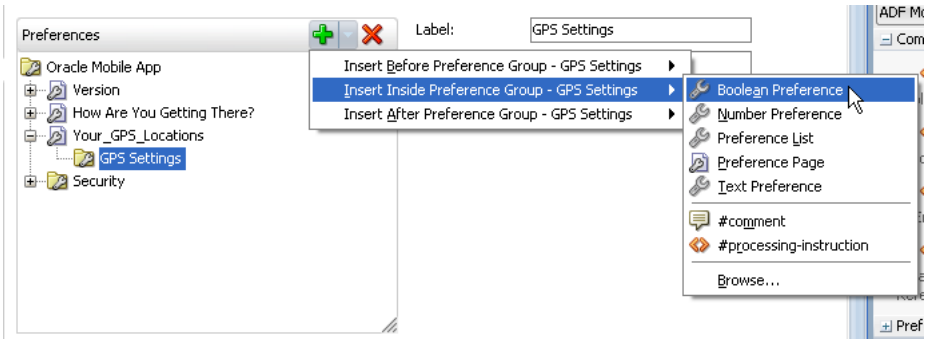
Before you begin:

Because an `<adfmf:preferenceBoolean>` element must be nested within an `<adfmf:preferenceGroup>` element, you must first insert a Preference Group component to the hierarchy.

To create a boolean preference list:

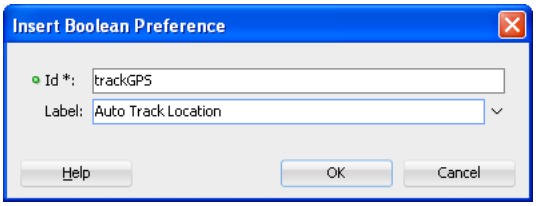
1. Select a Preference Group element, such as GPS Settings in Figure 13–13.

Figure 13–13 Adding a Boolean Preference to a Preference Group



2. Define the following attributes using the Insert Boolean Preference dialog, shown in Figure 13–14, and then click **OK**.
   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.

Figure 13–14 The Insert Boolean Preference Dialog



3. Accept the default value of `false`, or select `true`.

13.1.1.6 What Happens When You Add a Boolean Preference

When you add a Boolean Preference and designate its default value, JDeveloper updates the `<adfmf:preferences>` section of the `adfmf-application.xml` file with a `<adfmf:preferenceBoolean>` element, as illustrated in Example 13–3.

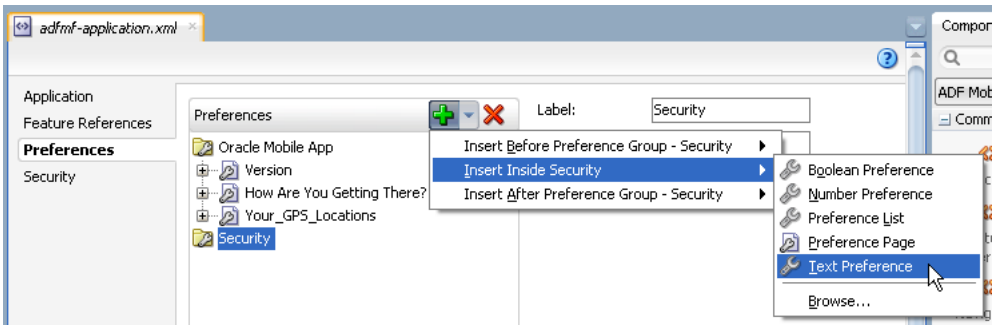
Example 13–3 Adding an <adfmf:preferenceBoolean> Element

```
<adfmf:preferencePage id="gps_tracking"
                      label="Your_GPS_Locations">
   <adfmf:preferenceGroup id="gps"
                          label="GPS Settings">
      <adfmf:preferenceBoolean id="trackGPS"
                               label="Auto Track Location"
                               default="true"/>
```

13.1.1.7 How to Add a Text Preference

Use the insert options, shown in Figure 13–15, to create a Text Preference, a dialog that enables users to store information or view default text. Figure 13–15 shows creating a text preference within a Preference Group called *Security*.

Figure 13–15 Inserting a Text Preference



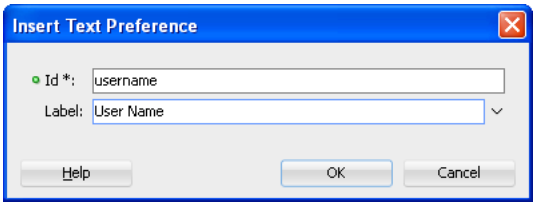
Before you begin:

Create a Preference Group element.

To create a text preference:

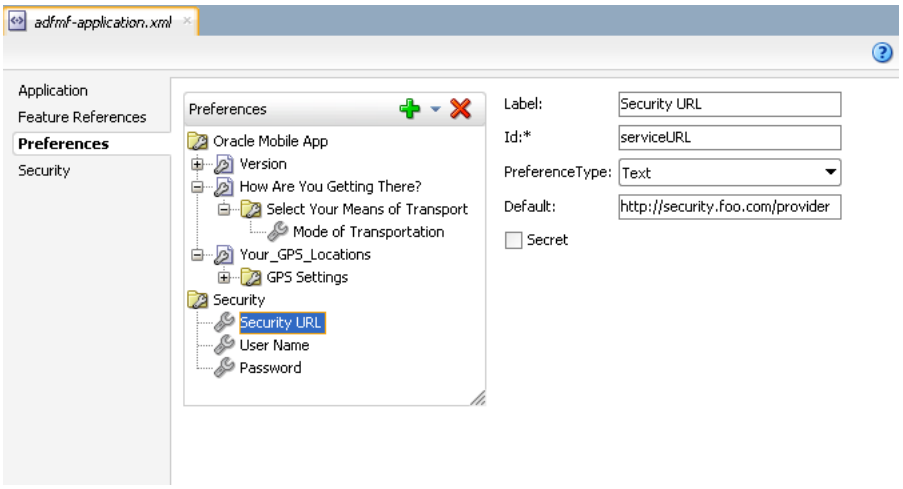
1. Select a Preference Group element.
2. Select **Insert Inside** and then **Text Preference**.
3. Enter the following information into the Insert Text Preference dialog, shown in Figure 13–16, and then click **OK**.
   - Enter a unique identifier.
   - Enter the descriptive text that displays in the user interface.

Figure 13–16 The Insert Text Preference Dialog



4. Define the following for the preference text dialog:
   - Enter the default text value.
   - Select **Secret** to hide the text preference to prevent users from editing it.

Figure 13–17 Defining the Text Preference



13.1.1.8 What Happens When You Define a Text Preference

When you add a Text Preference and designate its default value, JDeveloper updates the `<adfmf:preferences>` section of the `adfmf-application.xml` file with a `<adfmf:preferenceText>` element, as illustrated in Example 13–4.

Example 13–4 Adding the <adfmf:preferenceText> Element

```
<adfmf:preferenceGroup id="security" label="Security">
   <adfmf:preferenceText id="serviceURL"
                         label="Security URL"
                         default="http://security.foo.com/provider"/>
   <adfmf:preferenceText  id="username"
                          label="User Name"/>
   <adfmf:preferenceText  id="password"
                          label="Password"
                          secret="true"/>
</adfmf:preferenceGroup>
```

The Preference Group elements that define a security URL, user name, and password preference setting display similarly to Figure 13–18.

Figure 13–18 Text Preferences

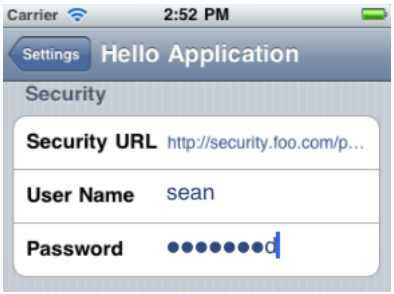


Figure 13–18 illustrates `<adfmf:preferenceText>` elements with a read-only value for the Security URL and an input value for the User Name. Because the ADF Mobile preferences are integrated with the iOS Settings application, the `secret="true"` attribute for the Password input text results in the application following the iOS convention of obscuring the user input with bullet points. For more information, see

the description for the `isSecure` text field element in *Settings Application Schema Reference*, available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`) and Section 13.4, "Platform-Dependent Display Differences."

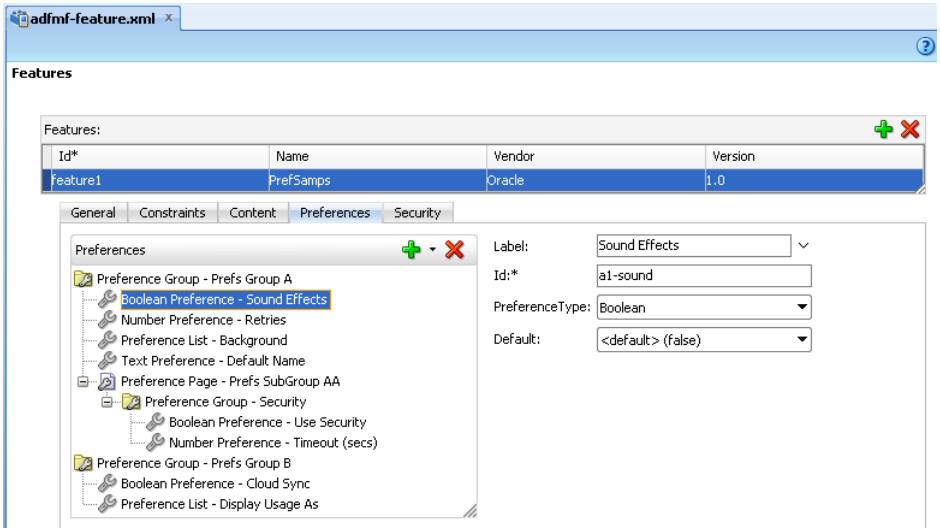
13.1.2 What Happens When You Create an Application-Level Preference Page

After you deploy the ADF Mobile application, the application-wide preference settings page is propagated to the device's global settings application, such as the Settings application on iOS-powered devices. For more information, see Section 16.2.3.6, "What Happens in JDeveloper When You Deploy an Application to an Android Emulator or Android-Powered Device," Section 16.4.5.1, "What Happens in JDeveloper When You Deploy an Application to an iOS Simulator or iOS-Powered Device," and Appendix B, "Converting Preferences for Deployment."

13.2 Creating User Preference Pages for Application Features

The Preferences tab, shown in Figure 13–19, enables you to define the user preferences that are applicable to a particular application feature. You build these preferences in the manner that you build the mobile application-level preferences as described in Section 13.1, "Creating User Preference Pages for an ADF Mobile Application." You then build the actual preference by creating an application feature that references an ADF Mobile AMX page that is embedded with the Boolean Switch, Input, and Output components described in Section 8.3, "Creating and Using UI Components."

Figure 13–19 Setting Application Feature-Level Preferences



13.3 Using EL Expressions to Retrieve Stored Values for User Preference Pages

When creating an application feature-level preference page, you add EL expressions to the ADF Mobile AMX components, such as the Input Text component in Example 13–5.

Example 13–5 Referencing Preference Values Using EL in ADF Mobile AMX Components

```
<amx:inputText label="Number" id="it1" inputType="number"
```

```
value="#{preferenceScope.feature.Feature1.f1top.f1Number}"/>
```

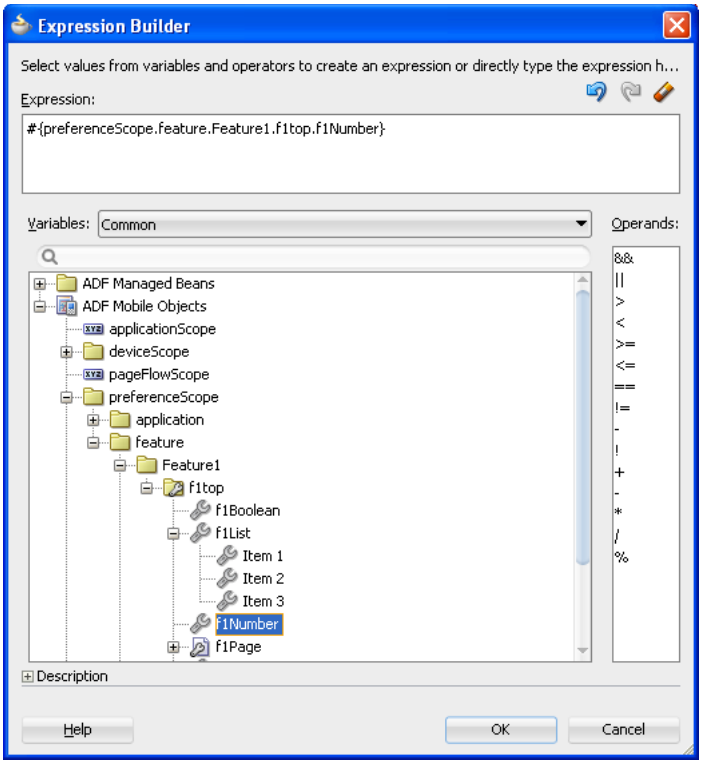
As illustrated in Example 13–5, EL expressions use the `preferenceScope` object to enable applications to access an application feature-level preference. These EL expressions are in the following format:

`preferenceScope.feature.`*`feature-id.group-id.property-id`*

Figure 13–20 illustrates using the Expression Builder to create the EL expression.The preference itself is designated by the IDs configured for various components in `adfmf-feature.xml`, such as the ID of the application feature `<adfmf:feature id="Feature1">`), the ID of a Preference Group (`<adfmf:preferenceGroup id="f1top">`), and the ID of a preference property (`<adfmf:preferenceNumber id="f1Number">`).

The EL expression may include zero or more `group-id` and `property-id` elements.

Figure 13–20 Building an EL Expression for a Preference



13.3.1 What You May Need to Know About preferenceScope

An EL expression has the following resolution pattern:

- From the JavaScript layer, EL value expressions are resolved using the following JavaScript function:

  ```
  adf.mf.el.getValue(expression, success, failed)
  ```

  The resolution of `adf.mf.el.getValue` begins with an attempt to resolve the expression locally using the JS-EL parser and JavaScript Context Cache. If the expression cannot be resolved locally, the expression is passed to the embedded Java layer for evaluation where it is resolved by the Java EL parser. This is done through the `GenericInvokeRequest` to the Model's `getValue` method.

- At the Java layer, an EL value expression is resolved using the following approach:

  ```
  String val =
  AdfmfJavaUtilities.evaluateELExpression("#{preferenceScope.feature.f0.vendor}")
  ;
  ```

  For a `setValue` method, the expression is resolved as follows:

  ```
  ValueExpression ve =
  AdfmfJavaUtilities.getValueExpression("#{preferenceScope.feature.f0.vendor}");
  ve.setValue(AdfmfJavaUtilities.getADFELContext(), value);
  ```

  Evaluation of the EL expression involves looking up the `preferenceScope` object. The evaluation is from left to right, where each token is resolved independently. After a token is resolved, it is used to resolve the next token (which is on its right).

Preferences cannot be exposed without the `preferenceScope` object.

13.3.2 Reading Preference Values in iOS Native Views

ADF Mobile integrates APIs provided for a native UI (such as `UIView` or `UIViewController`) to allow certain configurations on iOS platform.

When the native UI is initialized, an instance of the `ADFSession` object becomes available. You can use its `getPreferences` method to instruct ADF Mobile to provide a listing of the available preferences for the application as defined in the `adfmf-application.xml` file. As shown in Example 13–6, this method returns a `NSArray*` of preference property objects that can include the `id`, `value`, and `label` for the preference. This API call ensures that either the end user provided the value for a particular preference, or that the default value of the preference is returned.

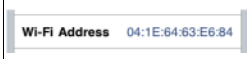
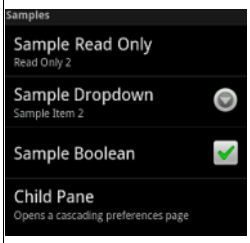
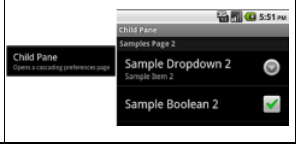
Example 13–6 Getting Preferences

```
//...
-(id) initWithADFSession:(id<ADFSession>) providedSession
{
   id me = [self init];
   session = providedSession;
   //...
   // Dump the preferences to the data display
   NSArray* prefsArray = [session getPreferences];
   NSString* prefs = [prefsArray JSONRepresentation];
   self.theData.text = [[NSString alloc ] initWithFormat:
        :@"%@\nUser Preferences = --> %@ <--", self.theData.text, prefs];
   //...
   return me;
}
```

13.4 Platform-Dependent Display Differences

The ADF Mobile preference pages maintain the native look-and-feel for both the iOS and Android platforms. Consequently, the ADF Mobile preference pages display differently on the two platforms. As shown in Table 13–1, preferences display inline on the iOS platform, meaning that the system does not invoke dialog pages. With a few exceptions, the Android platform presents these components as dialogs.

Table 13–1 Preference Component Comparison by Platform

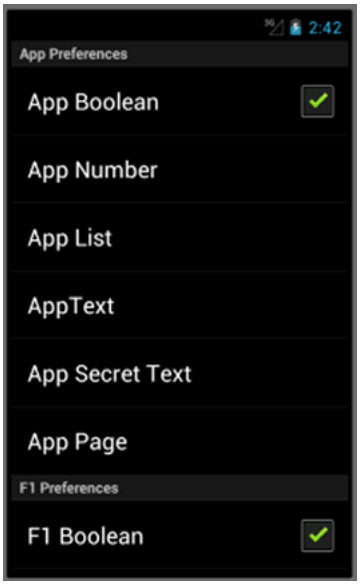
Component	iOS	iOS Display Examples	Android	Android Display Examples
Boolean Preference List	The Boolean preference is represented as value pair, such as *on* and *off*.		Android presents the Boolean preference as a check box.	
Text Preference (as default, read-only text)	iOS displays the read-only text inline.		Android displays the read-only text inline.	
Text Preference (as input text)	On iOS platforms, users enter text inline.		Android launches an input text dialog.	
Text Preference (as secret input text)	On iOS platforms, users enter text inline, with each character obscured by a bullet point after it has been entered. For more information, see Section 13.1.1.8, "What Happens When You Define a Text Preference."		Android launches an input text dialog and obscures each character with a bullet point after it has been entered.	
Single Item Selection List (from a Preference List)	iOS platforms display the single item selection list in a separate preferences page.		Android displays the single item selection list in a dialog.	
Preference Groups (Category Selection)	The iOS platform displays the preference elements within their parent preference group.		The Android platform displays the preference elements within their parent preference group.	
Preference Page	iOS launches a child preference page from a preference group.		Android launches a child preference page from a preference group.	

Although iOS and Android platforms have a Settings application, only the iOS platform supports integrating application-level preferences into the Settings application, as shown by the Oracle preferences in Figure 13–21.

Figure 13–21 Oracle Mobile Preferences in the iOS Settings Application

On Android-powered devices, users access preferences pages through the Preferences menu, as shown in Figure 13–22.

Figure 13–22 The Preferences Menu on an Android-Powered Device



14 Setting Constraints

This chapter describes how to set constraints that can restrict an application feature based on user access privileges or on device requirements.

This chapter includes the following sections:

14.1 Introduction to Constraints

A constraint describes when application features or content should be used. Constraints describe users and user roles, the characteristics of the device on which the mobile application is targeted to run, as well the hardware available on the device. You can define a constraint for the entire application feature, such as restricting an application feature for approvals to only users with a manager role. In terms of the application feature content, you can, for example, set a constraint that enables one task flow to be used only by users with a manager role and another task flow to only be used by users with a sales representative role.

14.2 Defining Constraints for Application Features

Constraints are evaluated at runtime and must evaluate to `true` to enable the application feature to be deployed. The `property`, `operator`, and `value` attributes of the `<adfmf:constraint>` element (a child element of `<adfmf:constraints>`) enable you to reference user, device, and hardware objects. An example of defining these attributes, shown in Example 14–1, illustrates defining these attributes to restrict access to an application feature to a Field Rep and restrict the application to run only on an iOS-powered device.

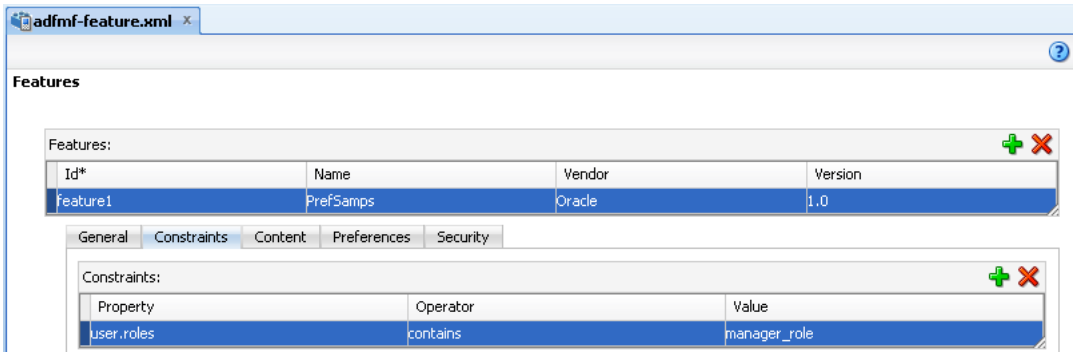
Example 14–1 The <adfmf:constraint> Element

```
<adfmf:constraints>
      <adfmf:constraint property="user.roles"
                        operator="contains"
                        value="Field Rep"/>
      <adfmf:constraint property="device.model"
                        operator="contains"
                        value="ios"/>
</adfmf:constraints>
```

14.2.1 How to Define the Constraints for an Application Feature

You declaratively configure the constraints for a selected application feature using the Constraints tab in the Features page, shown in Figure 14–1.

Figure 14–1 Defining Constraints



Defining the constraints for an application feature:

1. Click the **Constraints** tab.
2. Click **Add**.
3. Select a property and appropriate operator and then enter a value. For more information on using properties, see Section 14.2.3, "About the property Attribute."

14.2.2 What Happens When You Define a Constraint

Entering the values in the Constraints tab updates the descriptor file's `<adfmf:constraints>` element with defined `<adfmf:constraint>` elements, similar to Example 14–1.

14.2.3 About the property Attribute

ADF Mobile provides a set of `property` attributes that reflect users, devices, and hardware properties. Using these properties in conjunction with the following operators and an appropriate value determines how an application feature can be used.

- `contains`
- `equal`
- `less`
- `more`
- `not`

14.2.3.1 User Constraints

The `user.privileges` and `user.roles` constraints and the `contains` or `not` operators restrict access to an application feature by a logged-in user based on the user's roles and privileges depending on how the ADF Mobile runtime evaluates them.

Access control for the application login server connection is used to evaluate the constraints based on `user.roles` and `user.privileges`. In general, you control access to application features using constraints based on `user.roles` and

`user.privileges`. For example, to allow only a user with *manager_role* role to access an application feature, you must add a constraint of `user.roles contains manager_role` to the definition of the application feature.

At the start of application, the Access Control Service (ACS) web service, whose end point is defined in the **Access Control URL** option of the Create ADF Mobile Login Server Connection dialog, is invoked and the roles and privileges assigned to the user are fetched so that they can be evaluated by ADF Mobile. The constraints set for each application feature are evaluated, and only the application features that satisfy all the associated constraints are made available to the user. See also Section 17.4.2, "How to Designate the Login Page" and Section 17.4.6, "What You May Need to Know About the Access Control Service."

Logging into Oracle Identity Connect (OIC) populates the roles and privilege collections to the `user` object. See also Section 17.3, "Introduction to Authentication."

The `user.roles` and `user.privileges` use the `contains` and `not` operators as follows:

- `contains`—If the collection of roles or privileges contains the named role or privilege, then the runtime evaluates the constraint to `true`. Example 14–2 shows an example of using the `user.roles` property with the `contains` operator. The application feature will appear in the mobile application if the user's roles include the role of employee.

Example 14–2 Using the contains Operator for a User Role Collection

```
<feature ...>
   ...
   <constraints>
      <constraint property="user.roles"
                  operator="contains"
                  value="employee" />
   </constraints>
   ...
</feature>
```

- `not`—If the collection of roles or privileges does not contain the named role or privilege, then the runtime evaluates the constraint to `true`. In Example 14–3, the application feature is not included if the user's privileges contain the manager privilege.

Example 14–3 Using the not Operator with the user.privileges Property to Restrict Access to an Application Feature

```
<feature ...>
   ...
   <constraints>
      <constraint property="user.privileges"
                  operator="not"
                  value="manager" />
   </constraints>
   ...
</feature>
```

14.2.3.1.1 Hardware-Related Constraints The hardware object references the hardware available on the device, such as the presence of a camera, the ability to provide compass heading information, or to store files. These properties use the `equal` operator.

- `hardware.networkStatus`—Indicates the state of the network at the startup of the application. This property can be modified with three attribute values: `NotReachable`, `CarrierDataConnection`, and `WiFiConnection`. Example 14–4 illustrates the latter value. As illustrated in this example, setting this value means that this mobile application feature only displays in the mobile application if the device hardware indicates that there is a Wi-Fi connection. In other words, if the device does not have a Wi-Fi connection when the ADF Mobile application loads, then this application feature will not display.

Example 14–4 Defining the hardware.networkStatus Property

```
<feature ...>
...
   <constraints>
      <constraint property="hardware.networkStatus"
                  operator="equal"
                  value="WiFiConnection" />
   </constraints>
...
</feature>
```

> **Note:** This constraint is evaluated at startup on iOS-powered devices. If a device does not have a Wi-Fi connection at startup but subsequently attains one, for example, when a user enters a Wi-Fi hotspot, then the application feature remains unaffected and does not become available until the user stops and then restarts the mobile application.

- `hardware.hasAccelerometer`—Indicates whether or not the device has an accelerometer. This property is defined by a `true` or `false` value. Example 14–5 shows a `true` value, indicating that this application feature is only available if the hardware has an accelerometer.

Example 14–5 Using the hardware.hasAccelerometer Property

```
<feature ...>
...
   <constraints>
      <constraint property="hardware.hasAccelerometer"
                  operator="equal"
                  value="true" />
</constraints>
...
</feature>
```

> **Note:** Because all iOS-based hardware have accelerometers, this property must always have a value of `true` for the application feature to be available on iOS-powered devices.

- `hardware.hasCamera`—Indicates whether or not the device has a camera. This constraint is defined using a value attribute of `true` or `false`. In Example 14–6, the value is set to `true`, indicating that the application feature is only available if the device includes a camera.

Example 14–6 Using the hardware.hasCamera Property

```
<feature ...>
...
   <constraints>
      <constraint property="hardware.hasCamera"
                  operator="equal"
                  value="true" />
</constraints>
...
</feature>
```

> **Note:** Not all iOS-based hardware have cameras. This value is dynamically evaluated at the startup of ADF Mobile applications on an iOS-powered device. At this time, the mobile application removes the application features that do not evaluate to `true`.

- `hardware.hasCompass`—Indicates whether the device has a compass. You define this constraint with the attribute value of `true` or `false`, as shown in Example 14–7.

Example 14–7 Using the hardware.hasCompass Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasCompass"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** Not every iOS-powered device has a compass. This value is dynamically evaluated at the startup of ADF Mobile applications on an iOS-powered device. At this time, the mobile application removes the application features that do not evaluate to `true`.

- `hardware.hasContacts`—Indicates whether the device has an address book or contacts. You define this constraint with the attribute value of `true` or `false`, as shown in Example 14–8.

Example 14–8 Using the hardware.hasContacts Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasContacts"
                  operator="equal"
                  value"=true" />
      </constraints>
   ...
</feature>
```

> **Note:** Because contacts on iOS-based hardware are accessed through PhoneGap, the `value` attribute is always set to `true` for iOS-powered devices.

- `hardware.hasFileAccess`—Indicates whether the device provides file access. You define this constraint with the attribute value of `true` or `false`, as shown in Example 14–9. The application feature is only available if the runtime evaluates this constraint to `true`.

Example 14–9 Using the hardware.hasFileAccess Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasFileAccess"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** Because file access on iOS-based hardware is accessed through PhoneGap, the `value` attribute is always `true` for iOS-powered devices.

- `hardware.hasGeoLocation`—Indicates whether or not the device provides geolocation services. You define this constraint with the attribute value of `true` or `false`, as shown in Example 14–10. The application feature is only available if the device supports geolocation.

Example 14–10 Using the hardware.hasGeoLocation Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasGeoLocation"
                  operator="equal"
                  value="true"/>
   </constraints>
   ...
</feature>
```

> **Note:** PhoneGap does not provide access to the gelocation service for all iOS-powered devices. Depending on the device, the application feature may not be available when the constraint is evaluated by the runtime.

- `hardware.hasLocalStorage`—Indicates whether the device provides local storage of files. You define this constraint with the `value` attribute of `true` or `false`, as shown in Example 14–11. The application feature only displays if the device supports storing files locally.

Example 14–11 Using the hasLocalStorage Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasLocalStorage"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** Because PhoneGap provides access to local file storage on all iOS hardware, the `value` attribute is always `true` for iOS-powered devices.

- `hardware.hasMediaPlayer`—Indicates whether or not the device has a media player. You define this constraint with the `value` attribute of `true` or `false`, as shown in Example 14–12. The application feature only displays if the device has a media player.

Example 14–12 Using the hardware.hasMediaPlayer Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasMediaPlayer"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** For iOS-powered devices, the `value` attribute is always `true`, because PhoneGap provides access to media players on iOS-based hardware.

- `hardware.hasMediaRecorder`—Indicates whether or not the device has a media recorder. You define this constraint with the `value` of `true` or `false`, as shown in Example 14–13. The application feature is only included if the device hardware supports a media recorder.

Example 14–13 Using the hardware.hasMediaRecorder Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasMediaRecorder"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** Set this value to `true` for all iOS-powered devices because all iOS-based hardware have media recorders which can be accessed through PhoneGap. Some devices, such as the Apple iTouch, do not have a microphone but can allow end users to make recordings by attaching an external microphone.

- `hardware.hasTouchScreen`—Indicates whether or not the hardware provides a touch screen. You define this constraint with the `value` attribute of `true` or `false`, as shown in Example 14–14. The application feature is only included if the device hardware supports a touch screen.

Example 14–14 Using the hardware.hasTouchScreen Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.hasTouchScreen"
                  operator="equal"
                  value="true" />
   </constraints>
   ...
</feature>
```

> **Note:** Set the `value` attribute to `true` for iOS-powered devices, because all iOS-based hardware provides touch screens.

- `hardware.screen.width`—Indicates the width of the screen for the device in its current orientation. Enter a numerical value that reflects the screen's width (in pixels), such as 320 in Example 14–15. The value depends on the orientation of the device.

Example 14–15 Using the hardware.screen.width Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.screen.width"
                  operator="equal"
                  value="320" />
   </constraints>
   ...
</feature>
```

> **Note:** This value is evaluated at the startup of the mobile application when the runtime evaluates constraints and dismisses application features with constraints that do not evaluate to `true`. If a user rotates the device after the ADF Mobile application starts, ADF Mobile's runtime does not re-evaluate this constraint because the set of application features is fixed after the mobile application starts.

- `hardware.screen.height`—Indicates the height of screen for the device in its current position. Enter a numerical value that reflects the screen's width (in

pixels), such as 320 or 480, as shown in Example 14–16. The value depends on the orientation of the device.

Example 14–16 Using the hardware screen.height Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.screen.height"
                  operator="equal"
                  value="480" />
   </constraints>
   ...
</feature>
```

> **Note:** When the mobile application starts, the ADF Mobile runtime evaluates the screen height value for this constraint as part of the process of dismissing application features with constraints that do not evaluate to `true`. If a user changes the orientation of the device after the ADF Mobile application starts, the runtime does not re-evaluate this constraint, because the set of application features is fixed after the mobile application starts.

- `hardware.screen.availableWidth`—Indicates the available width of the device's screen in its current orientation. Enter a numerical value that reflects the screen's width (in pixels), such as 320 or 480, as shown in Example 14–17. The value depends on the orientation of the device.

Example 14–17 Using the hardware.screen.availableWidth Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.screen.availableWidth"
                  operator="equal"
                  value"320" />
   </constraints>
   ...
</feature>
```

- `hardware.screen.availableHeight`—Indicates the available height of the screen for the device in its current position. Enter a numerical value that reflects the screen's width (in pixels), such as 320 or 480, as shown in Example 14–18. The value depends on the orientation of the device.

Example 14–18 Using the hardware.screen.availableHeight Property

```
<feature ...>
   ...
   <constraints>
      <constraint property="hardware.screen.availableHeight"
                  operator="equal"
                  value"480" />
   </constraints>
   ...
</feature>
```

15

Accessing Data on Oracle Cloud

This chapter describes how an ADF Mobile application can access data hosted on Oracle Java Cloud Service

15.1 Enabling ADF Mobile Applications to Access Data Hosted on Oracle Cloud

ADF Mobile applications can access both SOAP and REST web services hosted on Oracle Cloud. To enable access to the hosted SOAP web services, create a web service data control as described in Section 10.2, "Creating a Web Service Data Control." You can enable access to RESTful web services by creating a URL Service data control as described in the "Exposing URL Services with ADF Data Controls" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. Depending on the content type, ADF Mobile applications can access cloud data by dragging and dropping a data control into an ADF Mobile AMX UI component, as described in Section 7.3.2, "How to Add UI Components and Data Controls to an ADF Mobile AMX Page," or programmatically, for applications whose content is delivered from a either a remote web server, or from locally stored HTML files.

15.1.1 How to Authenticate Against Oracle Cloud

You use the ADF Mobile Login Server Connection dialog to create a login server connection to authenticate against Oracle Cloud.

Before you begin:

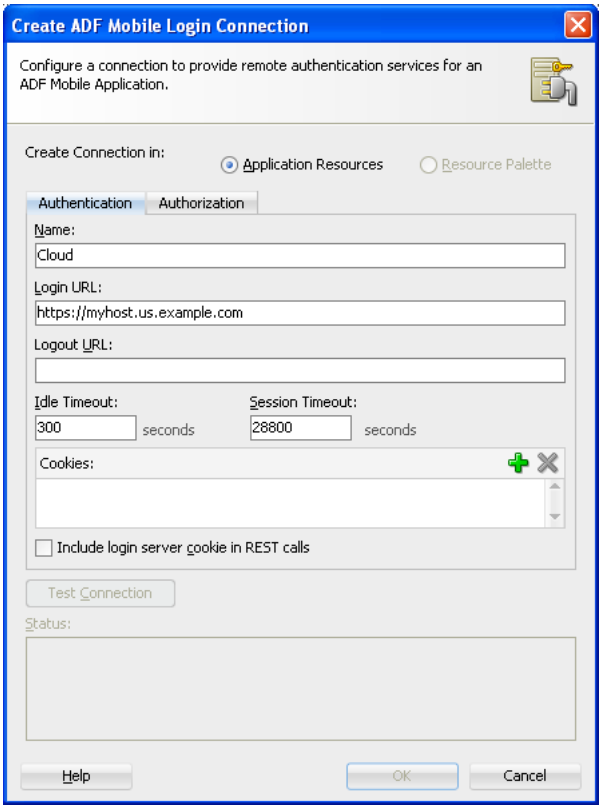
Obtain the Oracle Cloud URL that is used for the login server connection.

To create a login URL with an Oracle Cloud endpoint:

1. Select **Application**, then **New**, and then **Connections**.
2. Select **ADF Mobile Server Login Connection**.
3. Complete the Connect ADF Mobile Login Connection dialog, shown in Figure 15–1, by entering the following:
   - A name for the connection in the **Name** field.
   - The URL for Oracle Cloud in the **Login URL** field.

   For more information, refer to the Oracle JDeveloper online help and Section 17.4.2, "How to Designate the Login Page."

Figure 15–1 Creating the Login to Oracle Cloud



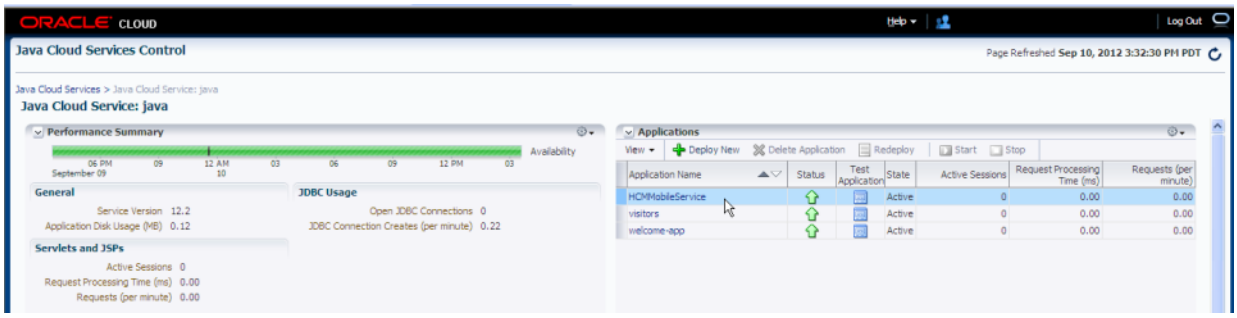
15.1.2 How to Create a Web Service Data Control to Access Oracle Java Cloud

The Create Data Service Control Wizard enables you to create the data control that accesses the hosted data. You use the WSDL URL of the SOAP web service deployed to Oracle Java Cloud to create this data control. If you do not know this URL, then you must create the URL to the WSDL document by appending the web service port name and `?wsdl` to the application context root.

Before you begin:

You must have access to a SOAP web service application that has been deployed to Oracle Java Cloud Service. This application must be available through the Applications pane of the Oracle Java Cloud Service Control home page. In addition, its Status and State must be noted as both Up and Active, respectively, as illustrated by the HCMMobileService application shown in Figure 15–2.

Figure 15–2 The Java Cloud Services Control Home Page



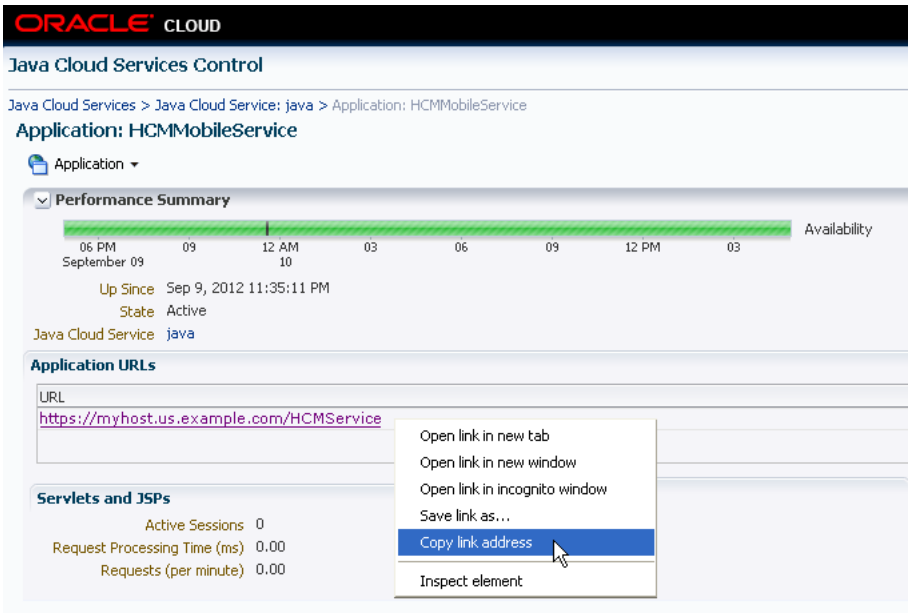
To create a web service data control:

1. Obtain the application context root of the web service hosted on Oracle Cloud as follows:

   a. Traverse to the application home page, shown in Figure 15–3, by clicking the application in the Applications pane (shown in Figure 15–2).

   b. Copy the URL, as shown in Figure 15–3. This URL is the application context root of the WSDL document.

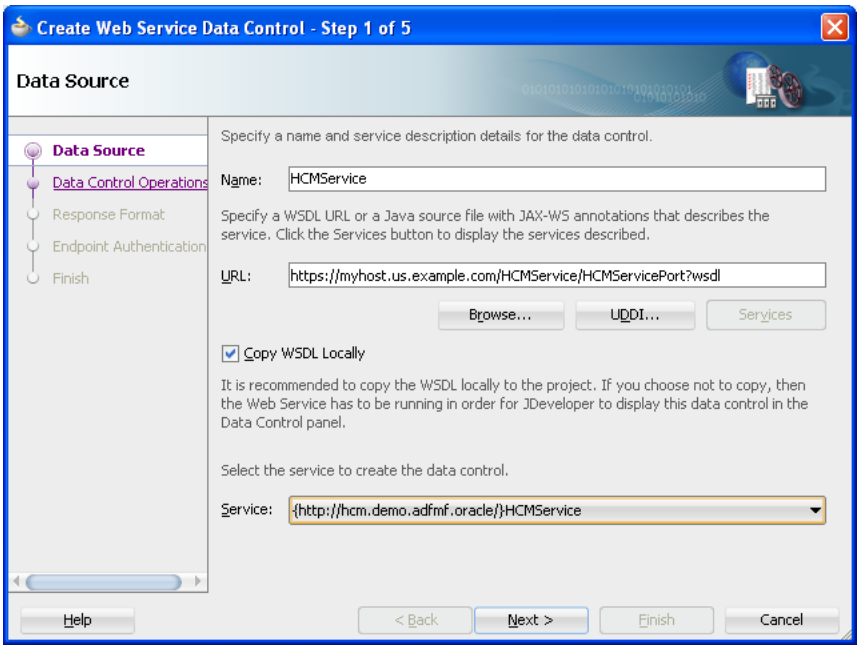
Figure 15–3 Copying the Web Service Application Context Root



2. In JDeveloper, right-click the view controller project in the Application Navigator and then open the Create Web Service Data Control Wizard, as described in Section 10.2, "Creating a Web Service Data Control."

3. In the Data Source page, shown in Figure 15–4, enter the name of the data control.

Figure 15–4 Entering the URL for the WSDL Document



4. In the **URL** field, paste the URL of the SOAP-based web service that is deployed to (and currently running on) Oracle Cloud Java Service.
5. Enable the data control to access the WSDL by appending a web service port name and `?wsdl` to the application context root, such as `HCMServicePort?wsdl` in Figure 15–4.
6. In the Data Controls Operations page, shown in Figure 15–5, select from among the web service operations that can be used by the application feature to retrieve data. Click **Finish**.

Figure 15–5 Selecting the Web Service Operations

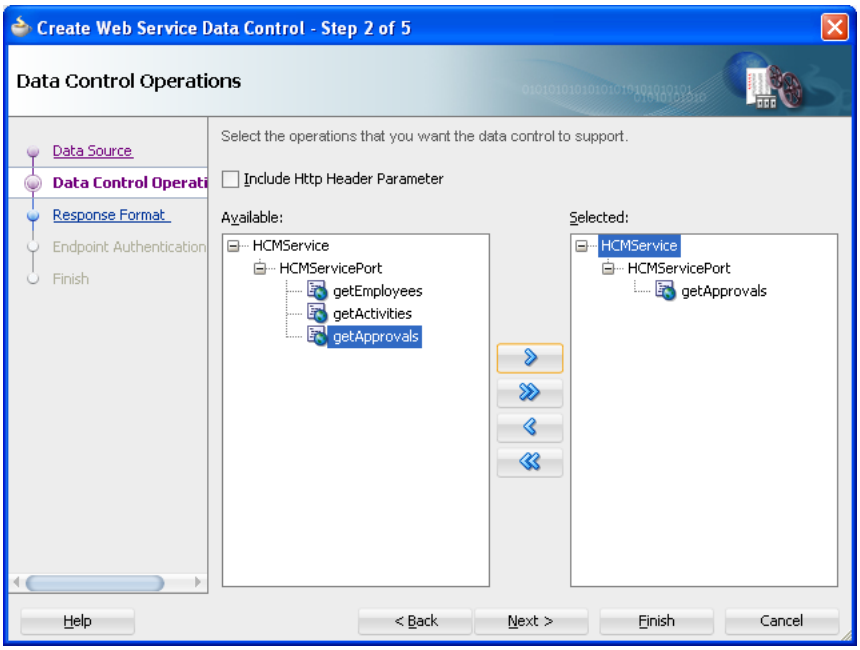


Figure 15–5 shows the web service operations returned by the ADF Mobile design time that can be made available to the ADF Mobile application. In this example, the design time has queried a web service that hosts human resources data and has returned operations to retrieve employee data, including expense approvals.

15.1.2.1 Configuring the Policy for SOAP-Based Web Services

You must configure a policy for a SOAP-based web service that is secured on Oracle Cloud. Using the Edit Data Service Control Policies dialog, described at Section 10.5, "Accessing Secure Web Services," you can select the `oracle/wss_http_token_over_ssl_client_policy`. See also the "How to Define Web Service Data Control Security" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Note: Only the `oracle/wss_http_token_over_ssl_client_policy` is supported for SOAP-based web services. For RESTful web services, basic authentication or SSL policies are supported.

15.1.3 What Happens When You Deploy an ADF Mobile Application that Accesses Oracle Java Cloud Service

After you deploy the application, the operations of the web service data control retrieve the data from a web service running on the Oracle Java Cloud Service instance.

Part VI

Completing Your Application

Describes how to enable security for an ADF Mobile application as well as how to debug and deploy an application

Part VI contains the following chapters:

16

Deploying ADF Mobile Applications

This chapter describes how to deploy ADF Mobile applications for testing and for publishing.

This chapter includes the following sections:

16.1 Introduction to Deployment of Mobile Applications

Before you can publish an application for distribution to end users, you must test it on a simulator or on an actual device to assess its behavior and ease of use. By deploying an iOS application bundle (an `.ipa` file) or Android application package file (`.apk`) to the platform-appropriate device or simulator, ADF Mobile enables you to test applications before publishing them to the App Store (Apple iTunes), or to an application marketplace, such as Google Play.

16.1.1 How ADF Mobile Deploys Applications

Deployment of an ADF Mobile project is accomplished by copying a platform-specific template application to a temporary location, updating that application with the code, resources, and configuration defined in the ADF Mobile project, and then building and deploying the application using the tools of the target platform. The tasks that comprise deployment, are as follows:

1. Validate that the platform-specific build tools are available from the specified platform SDK.
2. Validate the application's content, including the XML files.
3. If necessary, unzip the platform-dependent template file to a temporary location.

   Note: The template file may not be present if you selected **Build > Clean All**.

4. Copy default images to the proper location in the template.

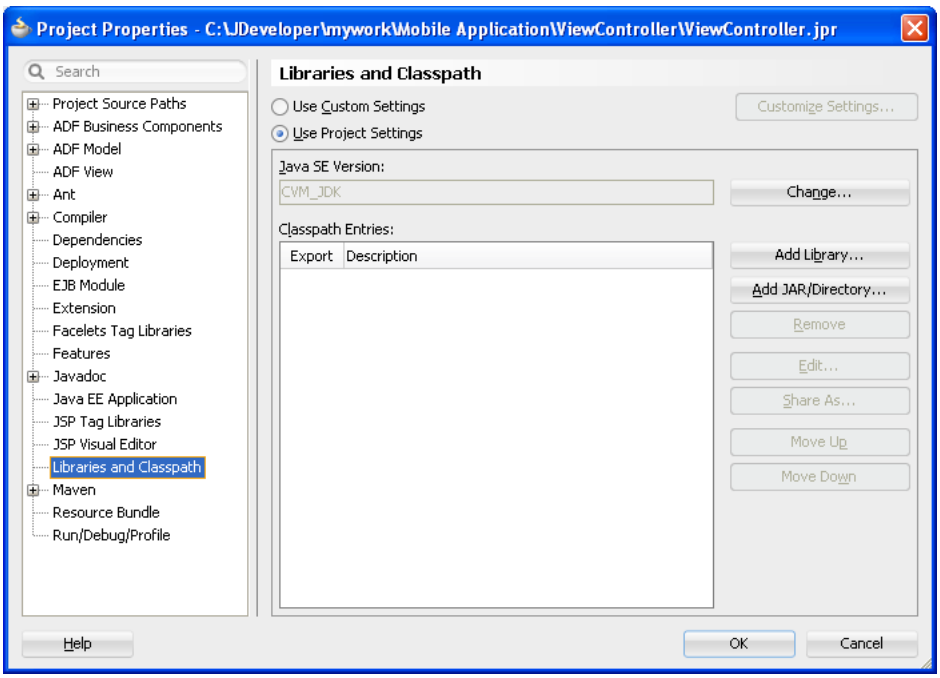
5. Copy custom images to the proper location in the template.
6. Copy the web content (including any images related to feature application) to the template location.
7. Copy the libraries of the Java Virtual Machine 1.4 (JVM 1.4) and any support libraries to the proper location in the template. For more information, see Section 16.1.1.2, "Deployment of the JVM 1.4 Libraries."
8. Copy custom content into the proper location in the template (this includes libraries and JAR files that have been included in the project properties).
9. Copy, or update, the metadata files in the template with preferences defined in the deployment profiles and the `adfmf-application.xml` and `adfmf-feature.xml` files.
10. Run the platform-specific build tools on the template project.
11. Run the platform-specific tools on the template project to sign the project (if necessary) and package the project for the chosen deployment.

For a detailed description of the deployment flow for Android applications, see Section 16.2.3.6, "What Happens in JDeveloper When You Deploy an Application to an Android Emulator or Android-Powered Device." For iOS applications, see Section 16.4.5.1, "What Happens in JDeveloper When You Deploy an Application to an iOS Simulator or iOS-Powered Device."

16.1.1.1 Deployment of Project Libraries

The libraries that you declare for the project using the Customize Libraries and Classpaths Dialog, shown in Figure 16–1, are included in the deployment artifacts for the project. This dialog enables the application features to access these libraries at runtime.

Figure 16–1 Adding Libraries to the Project



16.1.1.2 Deployment of the JVM 1.4 Libraries

For both Android and iOS applications, each ADF Mobile deployment includes a set of a different libraries that are specific to the type of deployment (release or debug) in combination with the deployment target (simulators or actual devices). In addition, each set of these libraries includes a JAR file of JVM 1.4. The application binding layer resides within this virtual machine, which is a collection of Objective-C libraries. For example, ADF Mobile deploys a JVM 1.4 JAR file and a set of libraries for a debug deployment targeted at an iOS simulator, but deploys a different JVM 1.4 JAR file and set of libraries to a debug deployment targeted to an actual iOS-powered device.

16.2 Working with Deployment Profiles

Preparing ADF Mobile applications for deployment begins with the creation of platform-specific deployment profiles. A deployment profile defines how an application is packaged into the archive that will be deployed to iOS- or Android-powered devices, iOS simulators, or Android emulators. The deployment profile does the following:

- Specifies the format and contents of the archive. For iOS, the archive format is an `.ipa` file, known as an application bundle. For Android, the format is an Android application package file (`.apk`).

  Note: The `.apk` file is archive-compatible, meaning that you can view its contents using an archiving tool such as WinZip or 7-Zip.

- Lists the source files, deployment descriptors, and other auxiliary files that will be packaged into the archive file.
- Describes the type and name of the archive file to be created.
- Highlights dependency information, platform-specific instructions, and other information.

16.2.1 How to Create a Deployment Profile

The ADF Mobile extension adds ADF Mobile-specific pages (described in Table 16–1) for both iOS and Android deployments to the standard ADF deployment pages that include:

- File Groups
- Library Dependencies
- Profile Dependencies

The Create Deployment Profile wizard, shown in Figure 16–2, enables you to create a default deployment profile. You can create as many deployment profiles as needed. For more information on these standard ADF deployment profile pages, click **Help** to see the JDeveloper online help.

Note: ADF Mobile application deployment only requires the creation of an application-level deployment profile; you do not have to create a view controller-level deployment profile.

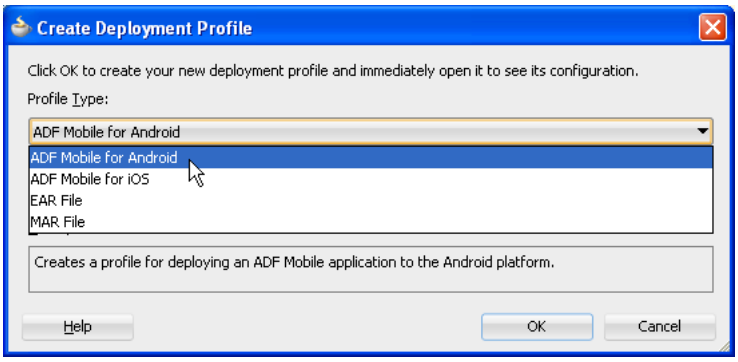
Before you begin:

To enable JDeveloper to deploy ADF Mobile applications, you must designate the SDKs for the target platforms using the ADF Mobile Preferences page as described in Section 3.3.1, "How to Configure the Development Environment for Platforms and Form Factors."

To create a deployment profile:

1. Choose **Application** and then **Deploy**.
2. Choose **New Deployment Profile**.
3. Depending on the target platform, select either **ADF Mobile for Android** or **ADF Mobile for iOS**, as shown in Figure 16–2.
4. Accept the default name for the profile or enter a new one. Click **OK**.
5. If needed, use the Options and Application Images pages as required for the applications and then click **OK**.

Figure 16–2 The Create Deployment Profile Wizard



16.2.2 What Happens When You Create a Deployment Profile

After you complete the wizard, JDeveloper creates a deployment profile and opens the Deployment Profile Properties editor.

Table 16–1 lists the ADF Mobile-specific pages in the Deployment Profile Properties editor, shown in Figure 16–6.

Table 16–1 ADF Mobile-Specific Deployment Profile Pages

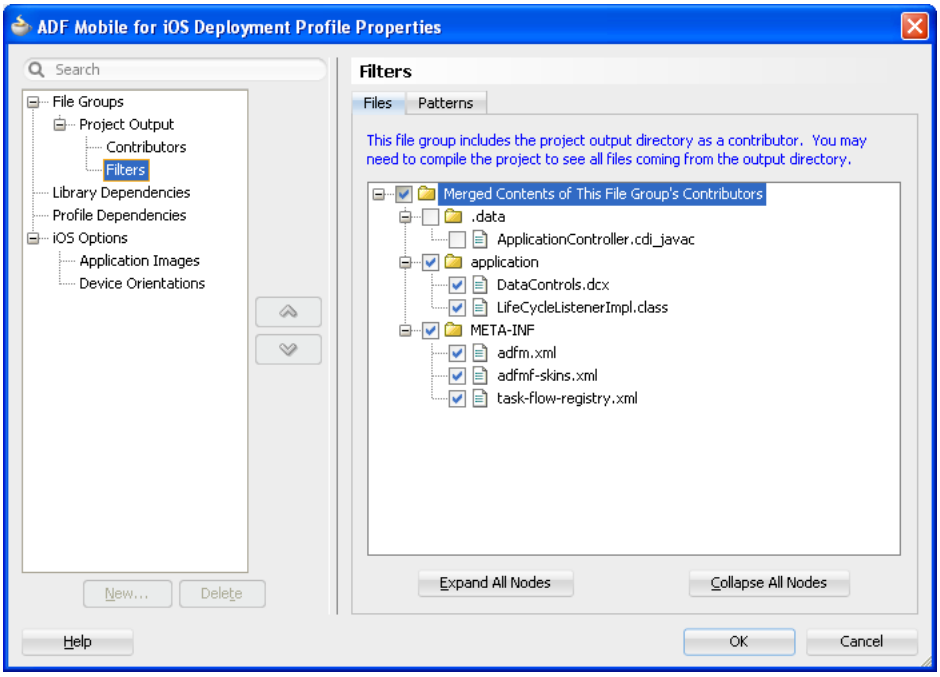
Page	Function
iOS Options	Enables you to modify the settings for an application to be deployed on an iOS-powered device or iOS simulator.
Android Options	Enables you to modify the settings for an application deployed to an Android-powered device or Android emulator.
Application Images	Enables you to assign custom icons to an application by adding the appropriate graphics file.
Device Orientations	Enables you to restrict the display of an application to certain device orientations. This page is used only for iOS deployment profiles.

Note: Deployment depends on the needs of your application. You can deploy an application using the default values seeded in the pages listed in Table 16–1.

When you deploy an application, JDeveloper creates a deployment directory and related subdirectory. It also creates Feature Archive files (FARs) for both the view controller and application controller projects. In addition to these two FARs, JDeveloper creates copies of any FARs that were imported into the project. Changes to the compilation profiles require the removal of the deployment directory. You can remove this directory, as well as the deployment directory within the view controller project that contains the FAR, by selecting **Build** and then **Clean All**.

Using the Filters panel, shown in Figure 16–3, you can exclude any selected artifact from being deployed. In particular, you can use the Filters panel to exclude any artifact that does not match the target deployment platform. For more information, refer to the online help for Oracle JDeveloper.

Figure 16–3 Removing Files from Deployment Using the Filters Panels



16.2.3 How to Create an Android Deployment Profile

The deployment profile creates the template for the application deployment to an Android device or emulator, or for creating an application as an Android application package file (`.apk`).

To create the deployment profile for Android, you must define the signing options for the application, the behavior of the `javac` compiler, and if needed, override the default Oracle images used for application icons with custom ones.

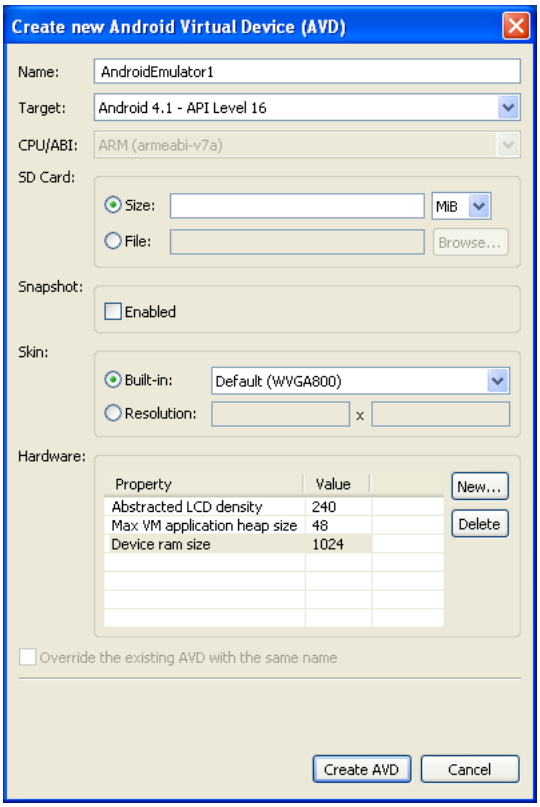
Before you begin:

Install and download the Android SDK as described in Section 3.5, "Setting Up Development Tools for Android Platform." If you deploy to an Android emulator, you

must create a virtual device for each emulator instance using the Android Virtual Device Manager, as described on the Android Developers website (`http://developer.android.com/tools/devices/index.html`). In creating an Android Virtual Device (AVD), ensure that the **CPU/ABI** value reflects an ARM system image, such as *ARM (armeabi-v7a)* in Figure 16–4. In addition, set the size of the AVD's SD card image file to a large RAM size, such as the value of 1024 MB (1 GB) set for the **Device ram size** property in Figure 16–4. You can, however, opt for a smaller size.

Note: ADF Mobile only supports the ARM EABI v7a System Image. Although ADF Mobile supports Android-powered devices that run on the Android 2.3.*n* platform (API Level 9), it does not support deployment to emulators running on this platform because they are based on the ARM926EJ-S rev 5 (v51) processor. Further, JVM 1.4 cannot run properly when deployed to Android emulators for the 2.*n* platform.

Figure 16–4 Creating an Android Virtual Device

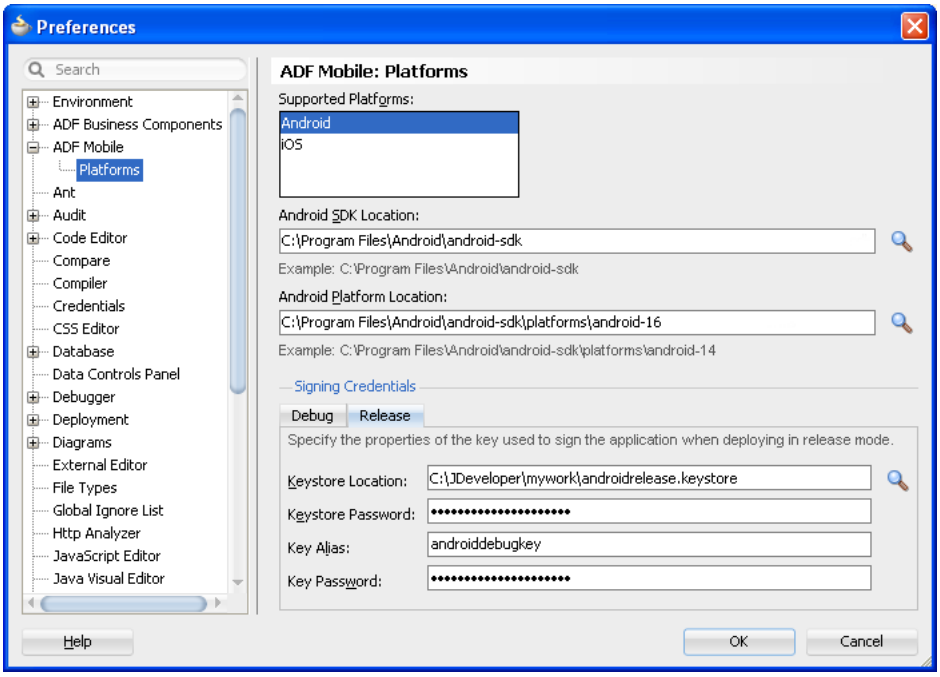


You must also set the ADF Mobile preferences for the Android platform SDKs (accessed by choosing **Tools**, then **Preferences**, then **ADF Mobile**, then **Platforms**, and then **Android**) to the locations for the SDK and platform, which are part of the Android SDK package download. Figure 16–5 shows these locations. Using the Platforms page, you also define the debug and release properties for a key that is used to sign the Android applications. Within the deployment profile, you subsequently designate an ADF Mobile application's release as either debug or release. You only need to define the signing key properties once. For more information, see Section 16.2.3.3, "Defining the Android Signing Options." See also the application

publishing information available on the Android Developers website (http://developer.android.com/tools/publishing/app-signing.html).

> **Note:** To deploy an application to an Android emulator, you must install API 14 or later (that is, Platform 4.0.*n*)

Figure 16–5 Setting the Android SDK, Platform and Signing Properties

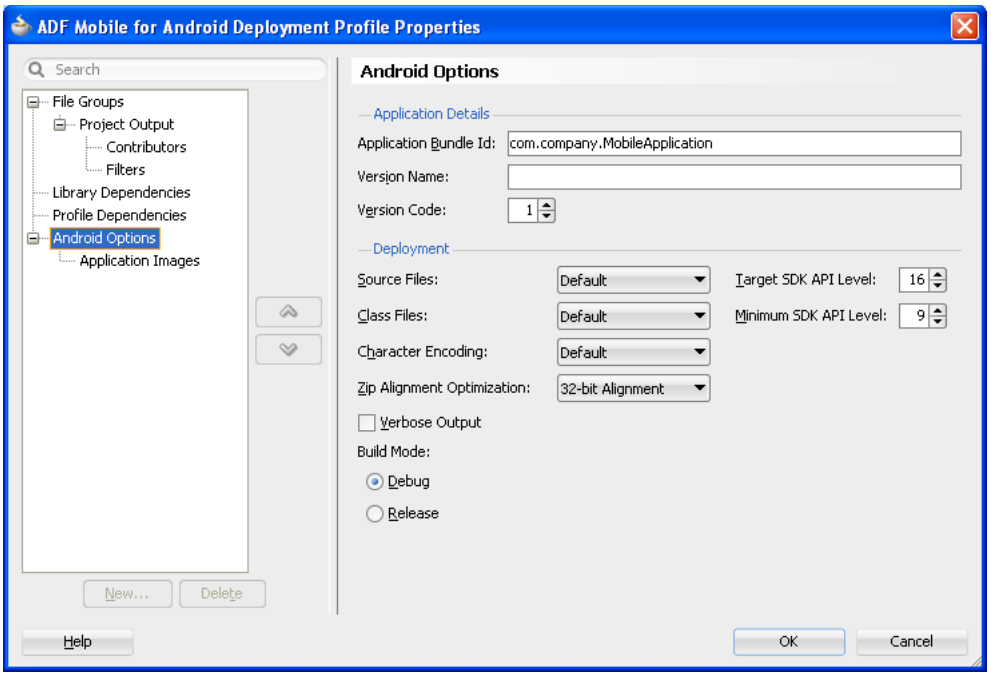


16.2.3.1 Setting the Options for the Application Details

The Android Options page, shown in Figure 16–6, enables you to do the following:

- Denote the version of the application. For information on versioning an Android application, refer to the Android Developers website (http://developer.android.com/tools/publishing/versioning.html).
- Configure the Android zipalign tool, an archive alignment tool that optimizes the packaging of `.apk` files. Data files stored in each application package, such as data manifests, are continually accessed by multiple processes within the Android operating environment. For more information on the zipalign tool, refer to the Android Developers website (http://developer.android.com/tools/help/zipalign.html).
- Set the logging level output as either verbose or debug.
- Set the signing options appropriate to the deployment target (emulator or device). For more information, see Section 16.2.3.3, "Defining the Android Signing Options."

Figure 16–6 The Deployment Profile Properties Editor (Android Options Page)



To set the application options:

1. Choose **Android Options**, as shown in Figure 16–6.
2. Accept the default values, or define the following options:

   - **Application Bundle ID**—A unique ID for the application, as set in the `id` attribute of the `adfmf-application.xml` file. Each application deployed to an Android device has a unique ID, one that cannot start with a numeric value. For more information, see Section 5.3, "Setting the Basic Information for an ADF Mobile Application."

     If needed, you can override this value in the deployment file. However, for the application to deploy, this name must follow the `<manifest>` element's `package` attribute of the Android manifest file, which is described on the Android Developers website (`http://developer.android.com/guide/topics/manifest/manifest-intro.html`). Specifically, the ID uses a reverse package format of an internet domain (*com.company.application*). To avoid naming collisions, the package name reflects domain ownership, such as *com.oracle.application*.

     Note: The application bundle ID cannot contain spaces.

   - **Version Name**— The release version of the application code that displays for the user. See also Section 5.3, "Setting the Basic Information for an ADF Mobile Application."
   - **Version Code**—An integer value that represents the version of the application code, which is checked programmatically by other applications for upgrades or downgrades. The minimum and default value is 1. You can select any value and increment it by 1 for each successive release.

16.2.3.2 Setting Deployment Options

The Options page enables you to set values that are passed in by the `javac` compiler tool options, set the zipalign options, and also the Android API revisions.

To set the JDK-Compatibility level for the R.java and .class Files:

1. Select the JDK-compatibility level from the **Source Files** dropdown list. The default value is 1.5. The value is specified when the deployment runs the `javac` tool to compile `R.java`, the Android-generated file for referencing application resources, using the `javac -source` option. Available values include:

   - 1.5
   - 1.6

   For information on `R.java`, see the Android Developers website (`http://developer.android.com/guide/topics/resources/accessing-resources.html`).

2. Select the JDK version compatibility for the compiled `.class` files from the **Class Files** dropdown list. The value is specified when the deployment runs the `javac` tool to compile the `R.java` file using the `javac -target` option. The default value is 1.5. Available values include:

   - 1.5
   - 1.6

3. Select the intended API Level on which the application is designed to run from the **Target SDK API Level** dropdown list. The minimum (and default) value is API Level 9, which corresponds to the Android 2.3.*n* platform. For more information, refer to the description of the `<uses-sdk>` attribute of the Android manifest file, available through the Android Developers website (`http://developer.android.com/guide/topics/manifest/manifest-intro.html`).

4. Select the minimum API Level on which the application is able to run from the **Minimum SDK API Level** dropdown list. The minimum and default value is 9, which corresponds to Android 2.3.*n* platform.

5. Select the native-encoding name that controls how the compiler interprets characters beyond the ASCII character set from the **Character Encoding** dropdown list. The default is **UTF-8**.

To set the ZIP alignment options:

Select the byte alignment (32-bit or 64-bit). Selecting 32-bit (the default) provides 4-byte boundary alignment.

To set the logging level:

Select **Verbose Output** for the Android deployment to log the full output provided by each of the command-line tools invoked by the deployment while building the `.apk`. If you do not select this option, then the deployment does not log the full output.

16.2.3.3 Defining the Android Signing Options

An application must be signed before it can be deployed to an Android device or emulator. Android does not require a certificate authority; an application can instead be self-signed.

Defining how the deployment signs an ADF Mobile application is a two-step process: within the ADF Mobile Platforms preference page, you first define debug and release properties for a key that is used to sign Android applications. You only need to configure the debug and release signing properties once. After you define these options, you configure the deployment profile to designate if the application should be deployed in the debug or release mode.

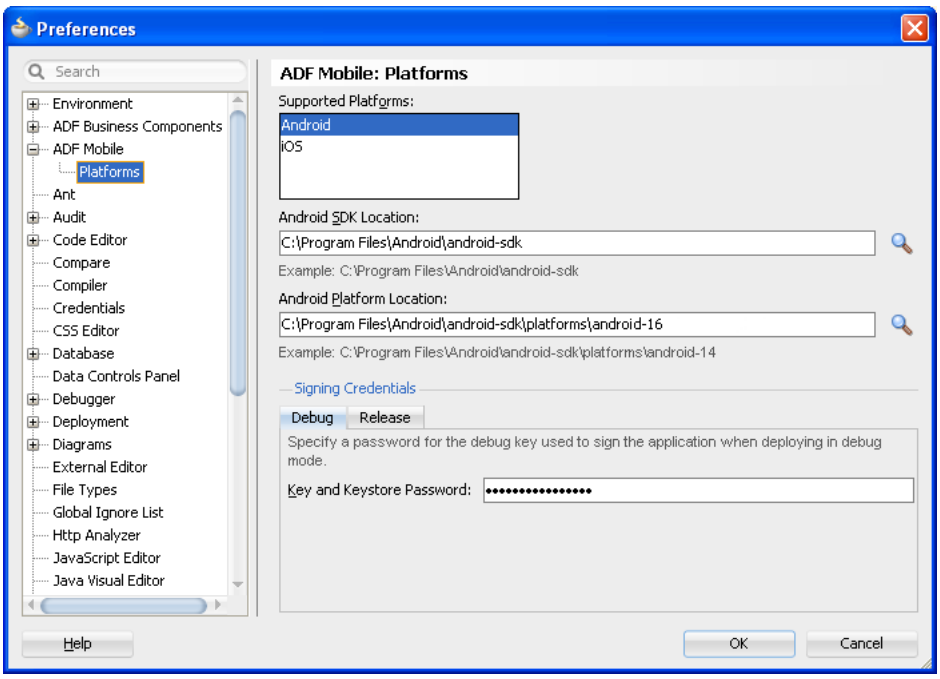
Before you begin:

Refer to the signing in release and debug mode described on the Android Developers website (http://developer.android.com/tools/publishing/app-signing.html).

To configure the key options for debug or release modes:

1. Choose **Tools**, then **Preferences**, and then **ADF Mobile**.
2. Choose **Platforms**.
3. Select the Debug tab, shown in Figure 16–7.

Figure 16–7 Configuring a Debug Deployment



4. Enter a password used by the deployment to create a keystore file and key needed for a debug deployment in the **Key and Keystore Password** field. This password, which generates a keystore and keyfile for deployment to an Android-powered device or emulator, can be any value, but must be at least six characters long. The default password is *Android*.
5. Select the Release tab, shown in Figure 16–5, and then define the following:
   - **Keystore Location**—Enter the directory of the keystore containing the private key.
   - **Keystore Password**—Enter the password for the keystore.
   - **Key Alias**—Enter an alias for the key. Only the first eight characters of the alias are used.

– **Key Password**—Enter the password for the key.

In addition to designating how the application will be signed, these parameters designate how the `R.Java` classes are compiled. For more information, see Section 16.2.3.6, "What Happens in JDeveloper When You Deploy an Application to an Android Emulator or Android-Powered Device."

6. Click **OK**.

To Set the Android build mode:

1. Select either **Debug** or **Release** as the build mode:

   - Select **Debug** for developing and testing an application. This option enables you to deploy an application on the Android platform without having to provide a private key. Use this option when deploying an application to an Android emulator or to an Android-powered device for testing.

     Note: You cannot publish an application signed with the debug keystore and key; this keystore and key are used for testing purposes only and cannot be used to publish an application to end users.

   - When the application is ready to be published, select **Release**. Use this option when the application is ready to be published to an application marketplace, such as Google Play.

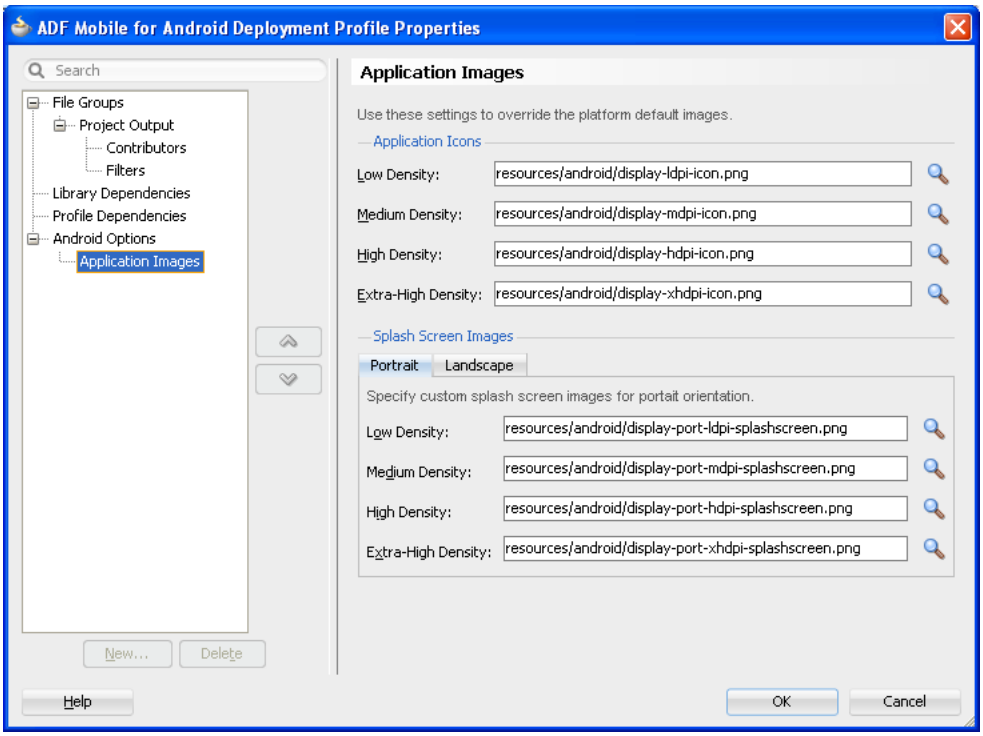
2. Click **OK**.

   After the `.apk` file is signed in either the debug or release modes, you can deploy it to a device or emulator.

16.2.3.4 How to Add a Custom Image to an Android Application

Enabling ADF Mobile application icons to display properly on Android-powered devices of different sizes and resolutions requires low-, medium-, and high-density versions of the same images. ADF Mobile provides default Oracle images that fulfill these display requirements. However, if the application requires custom icons, you can use the Application Images page, shown in Figure 16–8, to override default images by selecting PNG-formatted images for the application icon and for the splash screen. For the latter, you can add portrait and landscape images. If you do not add a custom image file, then the default Oracle icon is used instead. When creating custom images, follow the icon design guidelines described on the Android Developers website (`http://developer.android.com/guide/practices/ui_guidelines/icon_design.html`).

Figure 16–8 Setting Custom Portrait Images for an Android Application



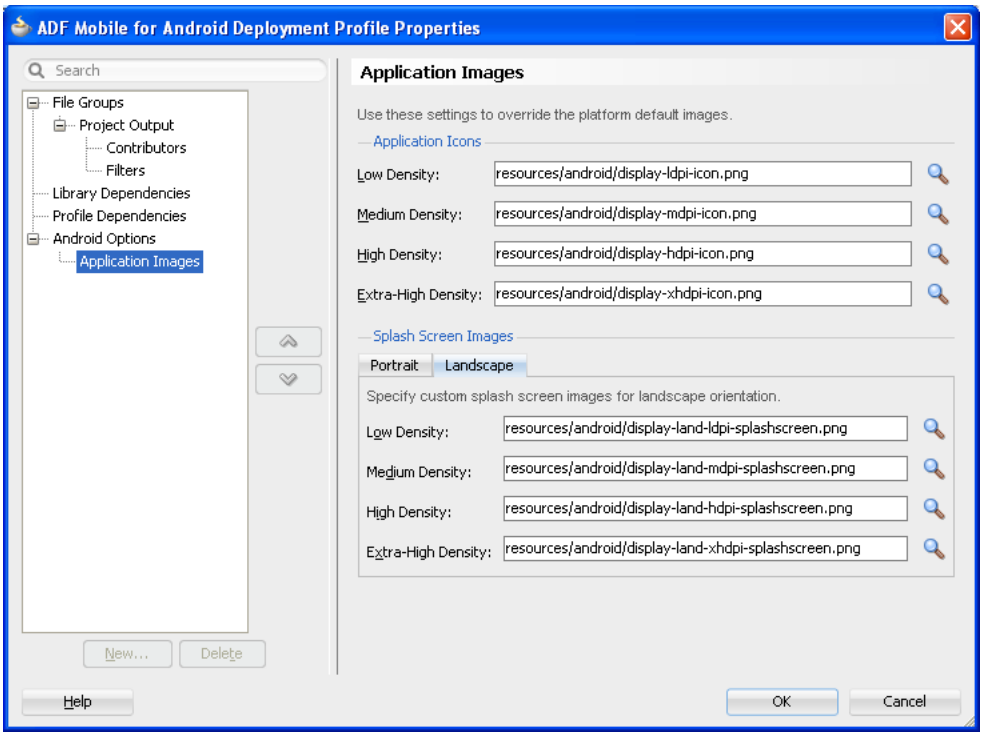
Before you begin:

Obtain the images in the PNG, JPEG, or GIF file format that use the dimensions, density, and components that are appropriate to Android theme and that can also support multiple screen types, as described on the Android Developers website (`http://developer.android.com/guide/practices/screens_support.html`).

To add custom images:

1. Click **Application Images**.
2. Use the **Browse** function to select the splash screen and icon image files from the project file. Figure 16–8 shows selecting images for application icons and portrait orientation splash screen images that applications use for displaying on devices with low-, medium-, high- and extra-high density displays. Figure 16–9 shows the Landscape tab selected for splash screen images, which enables you to add the landscape-orientation splash screen images that applications use for devices with low-, medium-, high- and extra-high density displays. These files can be located anywhere within the ADF Mobile application.
3. Click **OK**.

Figure 16–9 Setting Landscape Splash Screen Images



16.2.3.5 What Happens When JDeveloper Deploys Images for Android Applications

During deployment, ADF Mobile enables JDeveloper to copy the images from their source location to a temporary deployment folder. For the default images that ship with the ADF Mobile extension (located at *workspace_directory*`\Application Resources\Resources\images`), JDeveloper copies them from their seeded location to a deployment subdirectory of the view controller project (*workspace_directory*`\ViewController\deploy`). As shown in Table 16–2, each image file is copied to a subdirectory called `drawable`, named for the `drawable` object, described on the Android Developers website (`http://developer.android.com/reference/android/graphics/drawable/Drawable.html`). Each `drawable` directory matches the image density (`ldpi`, `mdpi`, `hdpi`, and `xhdpi`) and orientation (`port`, `land`). Within these directories, JDeveloper renames each icon image file as `adfmf_icon.png` and each splash screen image as `adfmf_loading.png`.

Table 16–2 Deployment File Locations for Seeded Application Images

Source File (...\resource\Android)	Temporary Deployment File (...ViewController\deploy)
`display-ldpi-icon.png`	`drawable-ldpi\adfmf_icon.png`
`display-mdpi-icon.png`	`drawable-mdpi\adfmf_icon.png`
`display-hdpi-icon.png`	`drawable-hdpi\adfmf_icon.png`
`display-xhdpi-icon.png`	`drawable-xhdpi\adfmf_icon.png`
`display-port-ldpi-splashscreen.png`	`drawable-port-ldpi\adfmf_loading.png`
`display-port-mdpi-splashscreen.png`	`drawable-port-mdpi\adfmf_loading.png`
`display-port-hdpi-splashscreen.png`	`drawable-port-hdpi\adfmf_loading.png`

Table 16–2 (Cont.) Deployment File Locations for Seeded Application Images

Source File (...\resource\Android)	Temporary Deployment File (...ViewController\deploy)
`display-port-xhdpi-splashscreen.png`	`drawable-port-xhdpi\adfmf_loading.png`
`display-land-ldpi-splashscreen.png`	`drawable-land-ldpi\adfmf_loading.png`
`display-land-mdpi-splashscreen.png`	`drawable-land-mdpi\adfmf_loading.png`
`display-land-hdpi-splashscreen.png`	`drawable-land-hdpi\adfmf_loading.png`
`display-land-xhdpi-splashscreen.png`	`drawable-land-xhdpi\adfmf_loading.png`

For custom images, JDeveloper copies the set of application icons from their specified location to the corresponding density and orientation subdirectory of the temporary deployment location.

16.2.3.6 What Happens in JDeveloper When You Deploy an Application to an Android Emulator or Android-Powered Device

For the runtime deployment to an Android-powered device, the deployer performs the following:

1. Validates all selected deployment options.
2. Deletes any pre-existing template for this deployment profile.
3. Unzips Android template from `Oracle_ADFmf_Framework.zip`.
4. Creates an `AndroidManifest.xml` file from an `AndroidManifest.template.xml` file that is provided in the template.
5. Generates the Android `preferences.xml`, `strings.xml` and `arrays.xml` files from the content in the `adfmf-application.xml` and `adfmf-feature.xml` files. For more information, see Section B.2, "Converting Preferences for Android."
6. Invokes the Android Asset Packaging Tool (`aapt.exe` on Windows, `appt` on Mac) to generate an `R.java` file and initial Android application package file (`.apk`).
7. Compiles the `R.java` file by invoking `javac` externally from the JDeveloper project compilation process. This file must be compiled with a JDK version compatibility of 1.5 or greater because the file runs within the Android JVM.
8. Analyzes and evaluates dependencies. Compiles user-added `.java` code using the JVM 1.4 JDK. These files must be compiled with the JDK and are executed in the JVM 1.4.
9. Invokes the Android tool that generates Dalvik byte code (`dex.bat`) from `.class` files or archive compatible files containing `.class` files. The content in the Android template JAR files is included as well (`phonegap.jar`, `container.jar`, and `ksoap.jar`).
10. Invokes the Android Asset Packaging Tool (`aapt.exe`) tool to update the application file created in Step 6 with the Dalvik byte code file.
11. Creates a keystore and key for debug-mode deployment.
12. Signs the Android application file.
13. Aligns application (for a release-mode deployment).

14. Invokes the Android Debug Bridge (`adb.exe` on Windows, `adb` on Mac) tool to deploy the application to an Android-powered device.

16.2.4 How to Create an iOS Deployment Profile

For iOS, you can use the Deployment Profiles Properties Editor to define the iOS application build configuration as well as the locations for the splash screen images and application icons.

Before you begin:

Download Xcode 4.2 or later, which includes the Xcode IDE, performance analysis tools, the iOS simulator, the Mac OS X and iOS SDKs, to the Apple computer that also runs JDeveloper. Because Xcode is used during deployment, you must install it on the Apple computer before you deploy the ADF Mobile application from JDeveloper. For instructions on downloading Xcode, refer to

`http://developer.apple.com/xcode/`

After you download Xcode, you must enter the location of its xcodebuild tool and, for deployment to iOS simulators, the location of the iOS simulator's SDK, in the ADF Mobile Platforms preference page. For more information, see Chapter 3.3.1, "How to Configure the Development Environment for Platforms and Form Factors."

To deploy an ADF Mobile application to an iOS-powered device (as opposed to deployment to an iOS simulator), you must obtain both a provisioning profile and a certification from the iOS Provisioning Profile as described in Section 16.2.4.2, "Setting the Device Signing Options."

To create a deployment profile:

1. Choose **iOS Options**, as shown in Figure 16–10.
2. Accept the default values, or define the following:
   - **Application Bundle Id**—If needed, enter a bundle ID to use for this application that identifies the domain name of the company. The application bundle Id must be unique for each application installed on an iOS device and must adhere to the reverse-package style naming conventions described in the "Application ID Errors" section in *iOS Development Workflow Guide* (that is, *com.<organization name>.<company name>*). For information on obtaining the Bundle Seed Id using the iOS Provisioning Portal, see Section 16.4.4.3, "Registering an Application ID." See also Section 5.3, "Setting the Basic Information for an ADF Mobile Application."

     Note: The application bundle ID cannot contain spaces.

     Because each application bundle ID is unique, you can deploy multiple ADF Mobile applications to the same device. Two applications can even have the same name as long as their application bundle IDs are different. Mobile applications deployed to the same device are in their own respective sandboxes. They are unaware of each other and do not share data (they have only the Device scope in common).
   - **Application Archive Name**—If needed, enter the name for the `.ipa` file created by ADF Mobile when you select either the **Deploy to distribution package** or **Deploy to iTunes for synchronization to device** options in the Deployment Action dialog, shown in Figure 16–21. Otherwise, accept the

default name. For more information, see Section 16.4.2, "How to Deploy an Application to an iOS-Powered Device" and Section 16.4.5, "How to Distribute an iOS Application to iTunes."

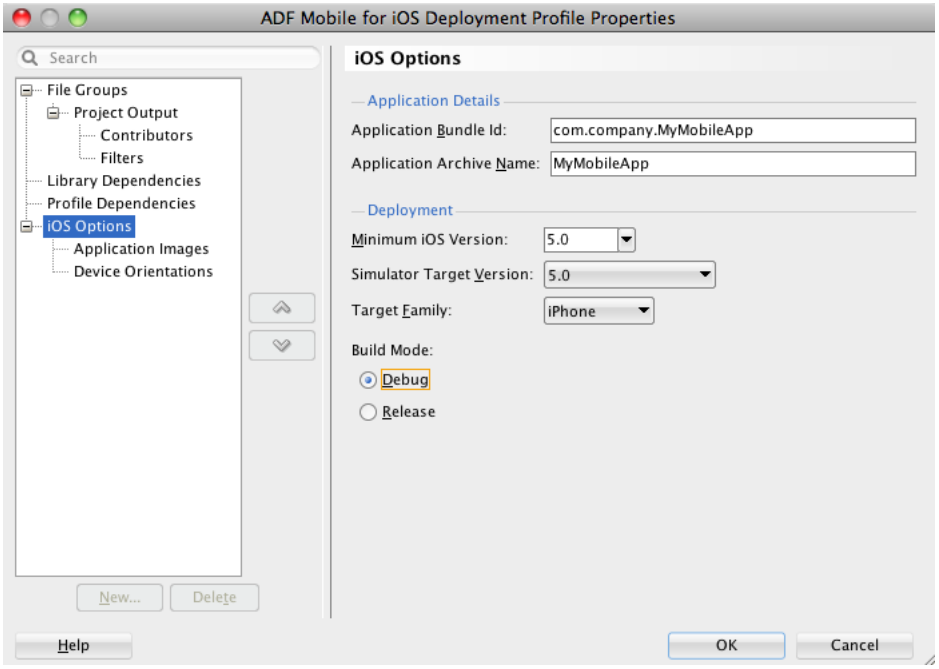
By default, ADF Mobile bases the name of the `.ipa` file on the application `id` attribute configured in the `adfmf-application.xml` file. For more information, see Section 5.3, "Setting the Basic Information for an ADF Mobile Application."

- **Minimum iOS Version**—Indicates the earliest version of iOS to which you can deploy the application. The default value is the current version. The version depends on the version of the installed SDK. The minimum version supported by ADF Mobile is 5.0.
- **Simulator Target Version**—Select the version of the emulator to which you are deploying the application. To find the available target versions, select **Hardware** and then **Version** on an iPhone simulator. The minimum version is 5.0. The default setting is **<Highest Available>**. See also the "Using iOS Simulator" chapter in *Tools Workflow Guide for iOS*, which is available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

Note: Older versions of the iOS target version are usually available in the simulator for testing.

- **Target Family**—Select the family of iOS products on which the application is intended to run. The default option is for both iPad and iPhone.

Figure 16–10 Setting the iOS Options



16.2.4.1 Defining the iOS Build Options

The iOS build options enable you to deploy an application with debug or release bits and libraries. The Options page presents the configuration options for the iOS signing modes, debug and release modes.

Before you begin:

Deployment of an iOS application (that is, an `.ipa` file) to an iOS-powered device requires a provisioning profile, which is a required component for installation, and also a signed certificate that identifies the developer and an application on a device. You must obtain these from the iOS Provisioning portal as described in Section 16.4.4, "What You May Need to Know About Deploying an Application to an iOS-Powered Device." In addition, you must enter the location for a provisioning profile and the name of the certificate in the ADF Mobile Platforms preference page, as described in Section 16.2.4.2, "Setting the Device Signing Options."

How to set the build options:

1. Chose **iOS Options**, as shown in Figure 16–10.
2. Select one of the following build options.
   - **Debug**—Select this option for development builds. Designating a debug build results in the inclusion of debugging symbols. See also Section 18.3.1, "How to Debug on iOS Platform."
   - **Release**—Select to compile the build with release bits and libraries.

16.2.4.2 Setting the Device Signing Options

The ADF Mobile Platforms preference page for iOS includes fields for the location of the provisioning profile on the development computer and the name of the certificate. You must define these parameters if you deploy an application to an iOS device.

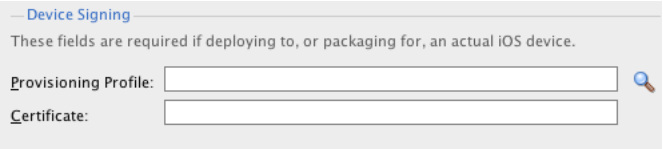
> **Note:** Neither a certificate nor a provisioning profile are required if you deploy an ADF Mobile application to an iOS simulator.

To set the signing options:

1. Choose **Tools**, then **Preferences**, and then **ADF Mobile**.
2. Choose **Platforms** and then choose **iOS**.
3. In the Device Signing section of the page, shown in Figure 16–11, enter the location of the provisioning profile in the **Provisioning Profile** field.
4. In the **Certificate** field, enter the name of the developer or distribution certificate that identifies the originator of the code (such as a developer or a company). You can view the name of the certificate using the Keychain Access utility (accessed from the Applications folder). Copy the entire name from the Keychain Access utility. The name entered into Certificate field must be in the following format:

   ```
   iPhone Developer: John Smith (PN3ENLQ3DU)
   ```

Figure 16–11 The Device Signing Section of the ADF Mobile Platforms Preference Page



Note: There are provisioning profiles used for both development and release versions of an application. While a provisioning profile used for the release version of an application can be installed on any device, a provisioning profile for a development version can only be installed on the devices whose IDs are embedded into the profile. For more information, see the "Configuring Development and Distribution Assets" chapter in *Tools Workflow Guide for iOS*, which is available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

16.2.4.3 Adding a Custom Image to an iOS Application

The Application images page enables you to override the default Oracle image used for application icons with custom images. The options in this page, shown in Figure 16–13, enable you to enter the locations of custom images used for different situations, device orientation, and device resolutions. For more information on iOS application icon images, see the "Application Icons" and "Providing Launch Images for Different Orientations" sections in *iOS App Programming Guide* and the "Custom Icon and Image Creation Guidelines" section in *iOS Human Interface Guidelines*. These documents are available from the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

Note: All images must be in the PNG format.

To add custom images:

1. Click **Application Images**.
2. Select **Do Not Add Gloss Effect to Icons** if the default iOS-style icon, which has a shine effect over the top of the icon is not used. Figure 16–12 illustrates the gloss effect.

Figure 16–12 An Icon with the Default Gloss Effect

3. Chose **Browse** to select images for the following:
   - iTunes Artwork—Browse to, and select, an icon image to override the default Apple image that iTunes assigns to `.ipa` files. This image is required for all applications and must be 512 x 512 pixels for both iPhone and iPad applications. For more information, see Section 16.2.4.4, "What You May Need to Know About iTunes Artwork."
   - Application Icons—Enter the location of the icon images for iPhones and iPads, or for the versions of these images used for devices equipped with

Retina displays. The file names must be the same for both the Retina and non-Retina version of the icon. For iPhone icons, the ADF Mobile deployment framework differentiates between the two by adding the *@2x* modifier to the file name of the Retina image. For iPad Retina displays, the deployment framework adds *-144* to the file name to reflect the dimensions of its associated image (144 x 144, in pixels). For non-Retina iPad displays, the deployment framework likewise denotes the image dimensions (72 x 72, in pixels) by adding the *-72* modifier to the iPad icon file name. Table 16–3 lists the required dimensions, in pixels, for the device-specific application images.

Table 16–3 Dimensions for Application Icons

Device	Dimension (in Pixels)
iPhone	57 x 57
iPhone Retina	114 x 114
iPad	72 x 72
iPad Retina	144 x 144

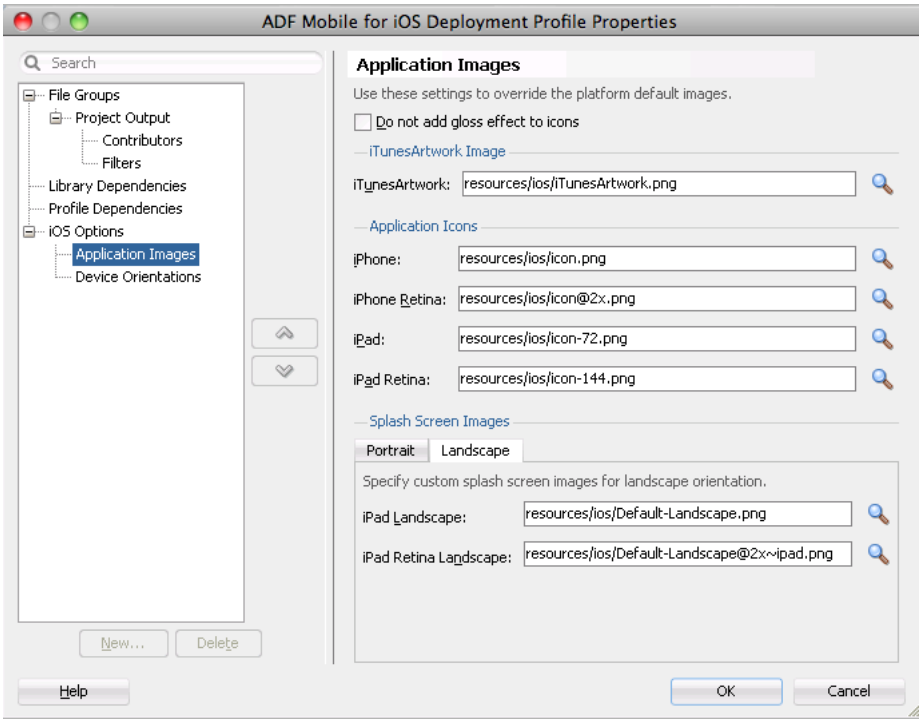
- Splash Screen Images—Enter the location of the image used when the application launches. For the iPhone, this image must be named `Default.png`. Depending on the device type and orientation, this image can be overridden by the images selected for its portrait or landscape versions. The ADF Mobile deployment framework attaches the *Portrait* and *Landscape* modifiers to the file names, as well as *@2x* for images used on Retina displays. Table 16–4 lists the required dimensions, in pixels, for the splash screen images.

Table 16–4 Dimensions for Splash Screen Images

Device	Portrait	Landscape
iPhone	320 x 480	N/A
iPad	768 x 1004	1024 x 748
iPad Retina	640 x 960	2048 x 1496
iPhone Retina	1536 x 2008	N/A

4. Click **OK**.

Figure 16–13 Adding Custom Images



16.2.4.4 What You May Need to Know About iTunes Artwork

By default, ADF Mobile applications deployed to an iOS device through iTunes, or deployed as an archive (`.ipa` file) for download, use the default Oracle image, shown in Figure 16–12 unless otherwise specified.

By selecting an iTunes artwork image as the icon for the deployed application, you override the default image. You can use an image to differentiate between versions of the application. Figure 16–14 illustrates the difference between the default image and a user-selected image, where Application4 is displayed with the default image and Application6 is displayed with a user-selected image (the Oracle icon, scaled to 512 x 512 pixels).

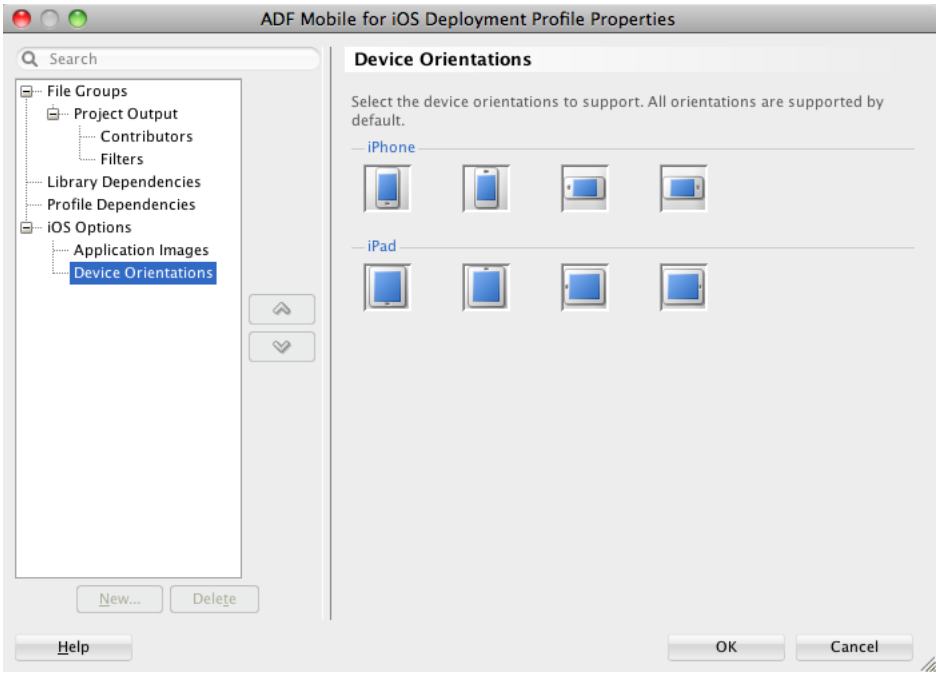
Figure 16–14 Custom and Default Application Icons

During deployment, ADF Mobile ensures that the icon displays in iTunes by adding the iTunes artwork image to the top-level of the `.ipa` file in a file called *iTunesArtwork*.

16.2.4.5 How to Restrict the Display to a Specific Device Orientation

By default, ADF Mobile supports all orientations for both iPhone and iPad. If, for example, an application must display only in portrait and in upside-down orientations on iPads, you can limit the application to rotate only to these orientations using the Device Orientation page, shown in Figure 16–15

Figure 16–15 Select a Device Orientation



To limit the display of an application to a specific device orientation:

1. Choose **Device Orientations**, as shown in Figure 16–15.
2. Clear all unneeded orientations from among those listed in Table 16–5. By default, ADF Mobile deploys to all of these device orientations. By default, all of these orientations are selected.

Table 16–5 iPhone Device Orientations

Icon	Description
	iPad, portrait—The home button is at the bottom of the screen.
	iPad, upside-down—The home button is at the top of the screen.
	iPad, landscape left—The home button is at the left side of the screen.
	iPad, landscape right—The home button is at the right side of the screen.
	iPhone, portrait—The home button is at the bottom of the screen.
	iPhone, upside-down—The home button is at the top of the screen.
	iPhone, landscape left—The home button is at the left side of the screen.
	iPhone, landscape right—The home button is at the right side of the screen.

3. Click **OK**.

16.2.4.6 What Happens When You Deselect Device Orientations

Deselecting a device orientation updates the source `.plist` file.

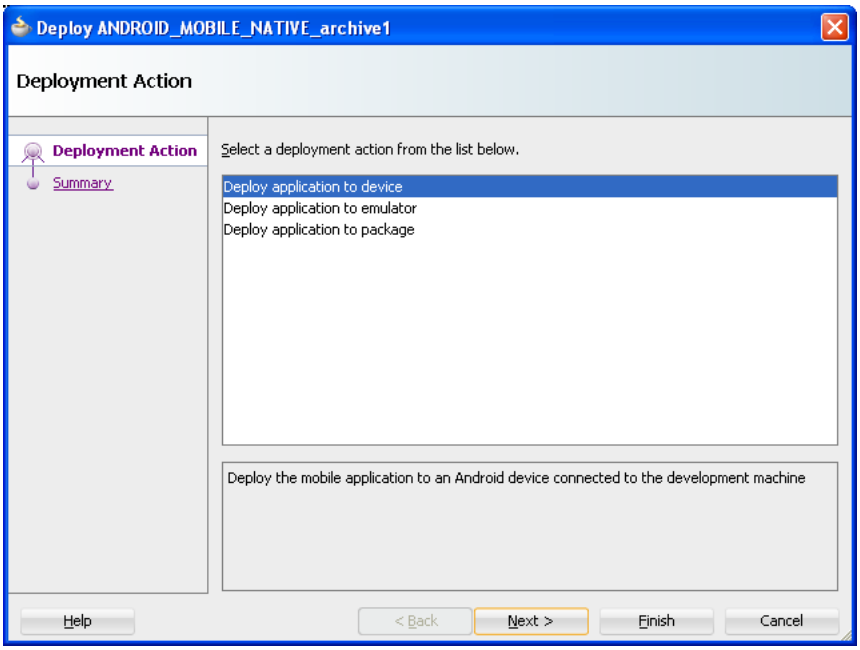
16.3 Deploying an Android Application

After you define the deployment profile, you can deploy an ADF Mobile application to the Android platform using the Deployment Action dialog, shown in Figure 16–16. Using this dialog, you can deploy the completed application to an Android emulator

or to an Android-powered device for testing. After you have tested and debugged the application, this dialog enables you to bundle the ADF Mobile application as an Android application package file (`.apk`) so that it can be published to end users through an application marketplace, such as Google Play.

> **Tip:** As an alternative to the Deployment Action dialog, you can deploy an ADF Mobile application to the Android platform manually using the OJDeploy command line tool as described in Section 16.6, "Deploying ADF Mobile Applications from the Command Line."

Figure 16–16 Deployment Action Dialog for Android Applications



16.3.1 How to Deploy an Android Application to an Android Emulator

You can deploy the ADF Mobile application directly to an Android emulator.

Before you begin:

Deployment to an Android emulator requires the following:

- Configure the debug password in the ADF Mobile Platforms preference page (accessed by choosing **Tools** > **Preferences** > **ADF Mobile**).

> **Note:** You must install the Android 4.0.*n* platform (API 14 or later).

- Ensure that the Android Virtual Device instance configuration reflects the ARM system image.
- In the Android Options page of the deployment profile:
  1. Ensure that **Debug** is selected.
  2. Click **OK**.

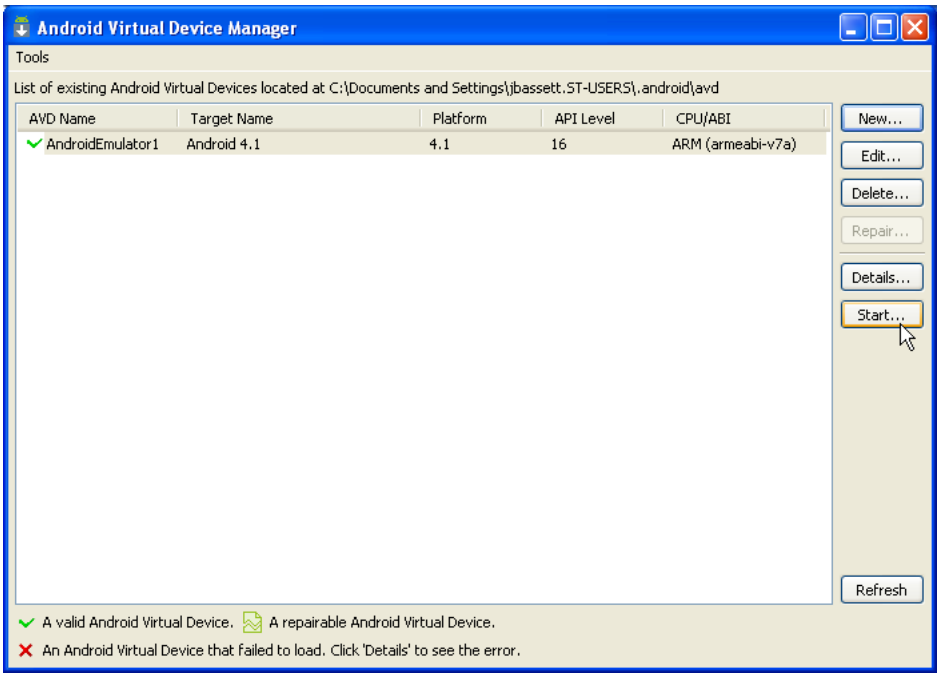
> **Note:** The ADF Mobile Platforms preferences page must be configured with the password that is used to generate the keystore and key for debug-mode deployment. See Section 16.2.3.3, "Defining the Android Signing Options."

- Start the Android emulator before you deploy an application.

  You can start the emulator using the Android Virtual Device Manager, as illustrated in Figure 16–17, or from the command line by first navigating to the `tools` directory (located in `Android\android-sdk`) and then starting the emulator by first entering `emulator -avd` followed by the emulator name (such as `-avd AndroidEmulator1`).

> **Note:** You can run only one Android emulator during a deployment.

Figure 16–17 Starting an Emulator Using Android Virtual Device Manager



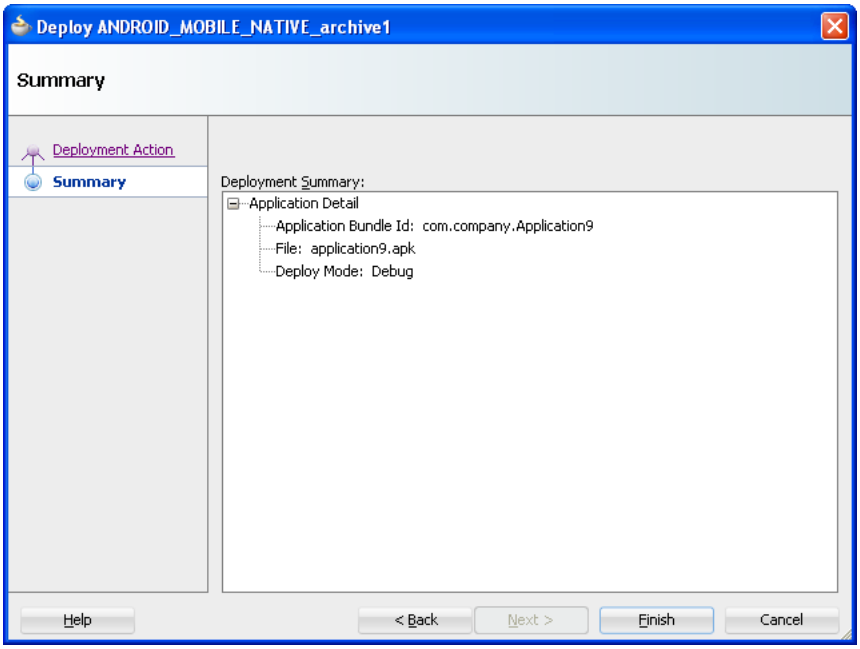
To deploy an application to an Android emulator:

1. Choose **Applications**, then **Deploy**, and then select an Android deployment profile.
2. Choose **Deploy application to emulator** and then choose **Next**.
3. Review the Summary page, shown in Figure 16–2, choose **Back** to select another deployment activity or choose **Finish**. The Summary page displays the following parameters from the deployment profile:
   - **Application Bundle Id**—The unique, Java language-like package name identifying the application.

Note: The Summary page shown in Figure 16–18 shows that the application bundle ID is in the reverse package format required for a successful deployment to an emulator. Deploying an application that does not follow the reverse-package format causes the emulator to shut down, which prevents the deployment from completing.

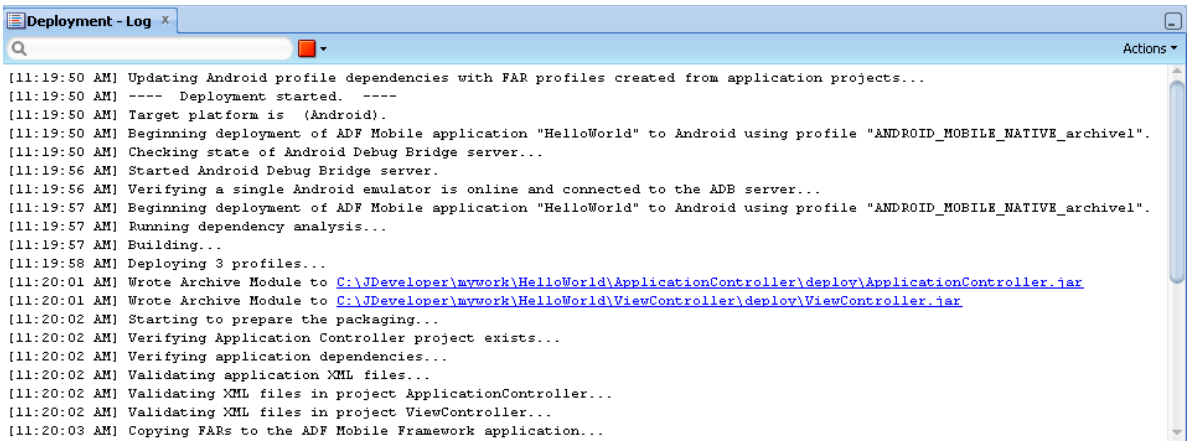
- **File**—The name of the `.apk` that is deployed to an Android target.
- **Deploy Mode**—The build mode. This value is either *Release* or *Debug*, depending on the value set in the deployment profile.

Figure 16–18 Summary for Android Emulator Deployment



4. Review the deployment log, as shown in Figure 16–19. The deployment log notes that the deployer starts the Android Debug Bridge server when it detects a running instance of an Android emulator. See also Section 16.3.6, "What You May Need to Know About Using the Android Debug Bridge."

Figure 16–19 The Deployment Log



16.3.2 How to Deploy an Application to an Android-Powered Device

You can deploy an ADF Mobile application directly to an Android-powered device that runs on a platform of 2.*n* (API Level 9) or later.

Before you begin:

Connect the device to the development computer that hosts JDeveloper, as described in Section 3.5.3, "How to Set Up an Android-Powered Device."

In the Deployment Options page, shown in Figure 16–6, select **Debug** as the build mode. Ensure that the debug signing credentials are configured in the ADF Mobile Platforms Preferences page, shown in Figure 16–7.

To deploy an application to an Android device:

1. Choose **Applications**, then **Deploy**, then select an Android deployment profile.
2. Choose **Deploy application to device** and then choose **Next**.
3. Review the Summary page. Click **Back** or **Next**.
4. Click **Finish**.

16.3.3 How to Publish an Android Application

After you have tested and debugged the application, as described in Chapter 18, "Testing and Debugging ADF Mobile Applications," you can publish it to an application marketplace (such as Google Play) by following the instructions provided on the Android Developers website (`http://developer.android.com/tools/publishing/publishing_overview.html`).

Before you begin:

In the Android Options page of the deployment profile, select **Release** as the build mode.

Note: You must configure the signing options in the ADF Mobile Platforms preferences page (accessed by choosing **Tools** > **Preferences** > **ADF Mobile**) as described in Section 16.2.3.3, "Defining the Android Signing Options."

To deploy an application as an .apk file:

1. Choose **Applications**, then **Deploy**, then select an Android deployment profile.
2. Choose **Deploy application to package** and then choose **Next**.
3. Review the Summary page, shown in Figure 16–18. Click **Back** or **Next**.
4. Click **Finish**.
5. Publish the application to an application marketplace.

16.3.4 What Happens in JDeveloper When You Create an .apk File

Deploying an application results in the following being deployed in an `.apk` file.

- The content in the `adfmsrc`
- The content in the `.adf` folder
- `adfmf-application.xml` and `adfmf-feature.xml` files
- `logging.properties` file
- The JVM 1.4 files

Table 16–6 Contents of the .apk File

Content	Location Within the .apk File
The content in the `.adf` folder	The root folder of the Android application file (`[apkRoot]\.adf`)
The content in the `adfmsrc` folder	The deployment packages the content in the `adfmsrc` folder into the default JAR file, which is located in a folder called `user` (`[apkRoot]\user`). This JAR file is added to the `.apk` using the Android Asset Packaging Tool (AAPT) and has a default name of the form `ANDROID_MOBILE_NATIVE_archiveN` where `N` is the nth Android created profile (you can override this name when creating the profile). This JAR file contains the following: ■ Any `.class` files generated from `.java` files that are added to the view controller project, as well as the `adfmsrc` content. The `.java` files are compiled using the JVM 1.4 JDK `javac` tool. ■ Contains data binding and pagedef metadata files. This JAR file is not processed by the Dalvik virtual machine. Because the `.class` files run in the JDK, they do not need to be converted into the Dalvik bytecode format (`.dex`).
`adfmf_application.xml` and `adfmf_feature.xml` files	Located in a file called Configuration (`[apkRoot]\Configuration`).
`logging.properties` file	Located in the root of the application file.
JVM 1.4 files	The JVM files are packaged into two separate folders: ■ The library file (`\framework\build\java_res\libs\`) in the template will be packaged into a `lib` folder in the `APK - [apkRoot]\lib`. ■ The `\framework\build\java_res\assets\storage` file is packaged in the `assets\storage` directory in the `APK - [apkRoot]\assets\storage`

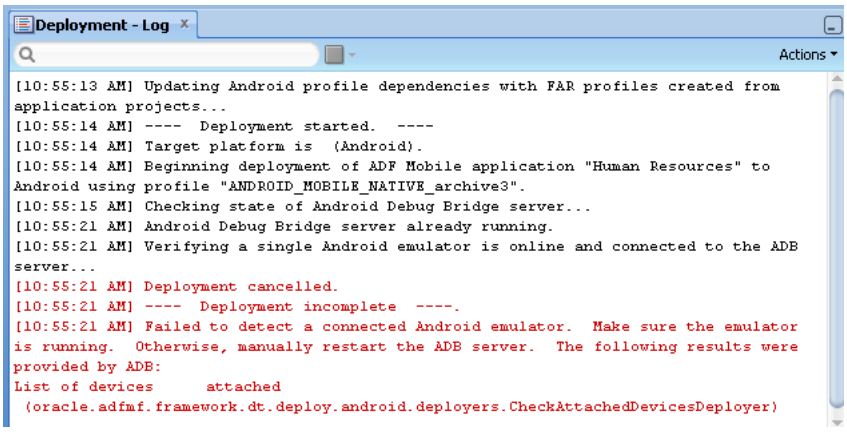
16.3.5 Selecting the Most Recently Used Deployment Profiles

After you select a deployment action, JDeveloper creates a shortcut on the Deploy menu that enables you to easily redeploy the application using that same deployment action.

16.3.6 What You May Need to Know About Using the Android Debug Bridge

The deployment restarts the Android Debug Bridge server five times until it detects a device (if deploying to a device) or emulator (if deploying to an Android emulator). If it detects neither, then it ends the deployment process, as shown in Figure 16–20.

Figure 16–20 Deployment Terminated

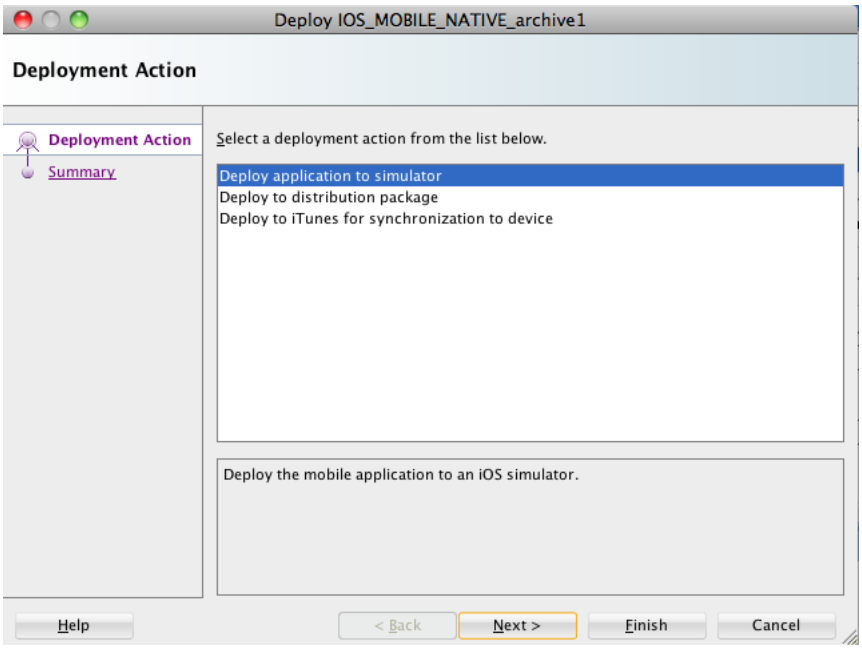


If you are using the Android Debug Bridge command line tool prior to deployment, then you must enter the same command again after the deployment has completed. For example, if you entered `adb logcat` to view logging information for an emulator or device prior to deployment, you would have to enter `adb logcat` again after the application has been deployed to resume the retrieval of the logging output. For more information about the Android Debug Bridge command line tool, which is located within (and executed from) the `platform-tools` directory of the Android SDK installation, refer to the Android Developers website (`http://developer.android.com/tools/help/adb.html`).

16.4 Deploying an iOS Application

The Deployment Action dialog, shown in Figure 16–21, enables you to deploy an iOS application directly to an iOS simulator or to a device through iTunes. You can only deploy an iOS application from an Apple computer.

Figure 16–21 The Deployment Action Dialog (for iOS Applications)



Tip: As an alternative to the Deployment Action dialog, you can deploy an ADF Mobile application to the Android platform manually using the OJDeploy command line tool as described in Section 16.6, "Deploying ADF Mobile Applications from the Command Line."

16.4.1 How to Deploy an iOS Application to an iOS Simulator

The Deployment Actions dialog enables you to deploy an iOS application directly to an iOS simulator.

Before you begin:

To enable deployment to an iOS simulator, you must perform the following tasks:

- Set the location of the SDK of the iOS simulator in the ADF Mobile Platforms preference page, shown in Figure 16–23.
- In the iOS Options page of the deployment profile, select **Debug**, and then click **OK**.

Note: You must enter the location of the provisioning profile and the name of the certificate in the ADF Mobile Platforms page (accessed by choosing **Tools** > **Preferences** > **Platforms**). For more information, refer to Section 16.2.4.2, "Setting the Device Signing Options."

- Run Xcode after installing it, agree to the licensing agreements, and perform other post-installation tasks as prompted.

Note: You must run Xcode at least once before you deploy the application to the iOS simulator. Otherwise, the deployment will not succeed.

Refer to the "Using iOS Simulator" section in *iOS Development Workflow Guide*. The iOS simulator is installed with Xcode. For deployment to an actual device, refer to the "Managing Devices and Digital Identities" section in *iOS Development Workflow Guide*.

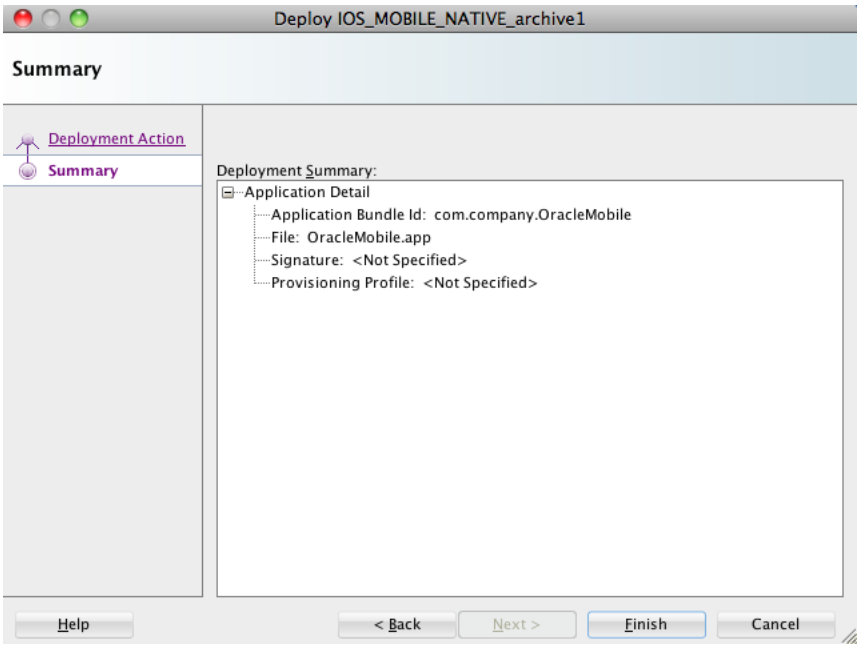
- Shut down the iOS simulator (if it is running).

To deploy an application to an iOS simulator:

1. Choose **Applications**, then **Deploy**, then select an iOS deployment profile.
2. Choose **Deploy application to simulator** and then choose **Next**.
3. Review the Summary page, shown in Figure 16–22, which displays the following values. Click **Finish**.
   - **Application Bundle Id**—The unique name that includes a Java language-like package name (*com.<organization name>.<application name>*) prepended with the Bundle Seed that is generated from the iOS Provisioning Portal.
   - **File**—The file name of the final image deployed to an iOS target.
   - **Certificate**—The developer or company that authored the application. If this value has not been configured in the Options page of the deployment profile, then the Summary page displays *<Not Specified>*.
   - **Provisioning Profile**—The name of the provisioning profile that associates one or more development certificates and devices with an application ID. If this value is not configured in the Options page of the deployment profile, then the Summary page displays *<Not Specified>*.

Note: Deployment to an iOS simulator does not require that the values for Certificate and Provisioning profile be defined. In this deployment scenario, the Summary page displays *<Not Specified>* for these values.

Figure 16–22 The Deployment Actions Summary Dialog



16.4.2 How to Deploy an Application to an iOS-Powered Device

The **Deploy to iTunes for Synchronization to device** option enables you to deploy an ADF Mobile application to an Apple device for debugging and testing.

Before you begin:

You cannot deploy an application directly from JDeveloper to a iOS device; an application must instead be deployed from the Applications folder in Apple iTunes. To accomplish this, you must perform the following tasks:

- Download Apple iTunes to your development computer and set the location of the `Automatically Add to iTunes` folder (the location used for application deployment) in the ADF Mobile Platforms preference page, shown in Figure 16–23.

  Tip: Although your user home directory (`/User/<username>/Music/iTunes/iTunes Media/Automatically Add to iTunes`) is the default directory for the iTunes Media folder, you can change the location of this folder as follows:

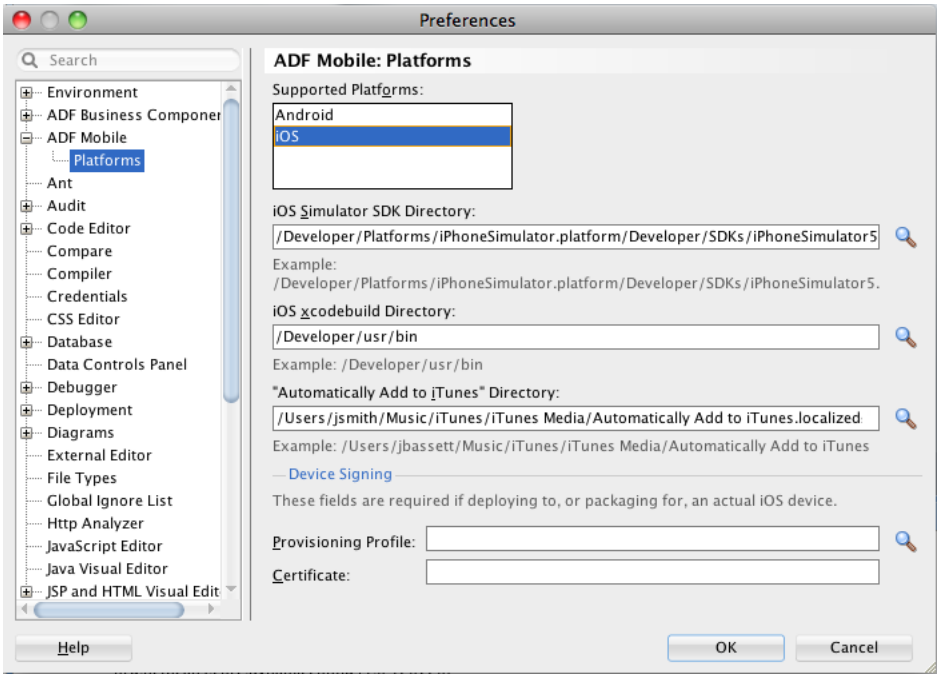
  1. In iTunes, select **Edit**, **Preferences**, then **Advanced**.
  2. Click **Change** and then browse to the new location.
  3. Consolidate the library.
  4. Delete the original iTunes Media folder.

  For instructions, refer to Apple Support (`http://support.apple.com`).

  You must also update the location in the ADF Mobile Platforms preference page.

- Set the location of the Xcode folder where the xcodebuild tool is invoked, such as `/Developer/usr/bin` in Figure 16–23.

Figure 16–23 Setting the Locations for the iTunes Media Folder and the xcodebuild System



- Enter the name of the certificate and the location of the provisioning profile in the ADF Mobile Platforms preference page. The OS Provisioning Portal generates the certificate and provisioning profile needed for deployment to iOS devices, or for publishing `.ipa` files to the App Store or to an internal download site.

> **Note:** The deployment will fail unless you set the iOS provisioning profile and certificate to deploy to a device or to an archive. ADF Mobile logs applications that fail to deploy under such circumstances. For more information, see Section 16.4.4, "What You May Need to Know About Deploying an Application to an iOS-Powered Device."

- In the iOS Options page of the deployment profile, select **Debug** as the build mode and then **OK**.

To deploy an application to an iOS-powered device:

1. Choose **Applications**, then **Deploy**, then select an iOS deployment profile.
2. Choose **Deploy to iTunes for Synchronization to device** and then choose **Next**.
3. Review the Summary page, which displays the following values. Click **Finish**.
   - **Application Bundle Id**—The unique name that includes a Java language-like package name (*com.<organization name>.<application name>*) prepended with the Bundle Seed that is generated from the iOS Provisioning Portal.
   - **File**—The file name of the final image deployed to an iOS target.
   - **Certificate**—The developer (or company) who authored the application. If this value has not been configured in the Options page of the deployment profile, then the Summary page displays *<Not Specified>*.

- **Provisioning Profile**—The name of the provisioning profile that associates one or more development certificates and devices with an application ID. If this value is not configured in the Options page of the deployment profile, then the Summary page displays <*Not Specified*>.

> **Note:** The Certificate and Provisioning Profile values cannot be noted as <*Not Specified*>; you must specify these values in the Options page to enable deployment to iTunes.

4. Connect your iOS device to the development computer.
5. Open iTunes and then synchronize your device.

16.4.3 What Happens When You Deploy an Application to an iOS Device

The application appears in the iTunes Apps Folder, similar to the one illustrated in Figure 16–14 after a successful deployment.

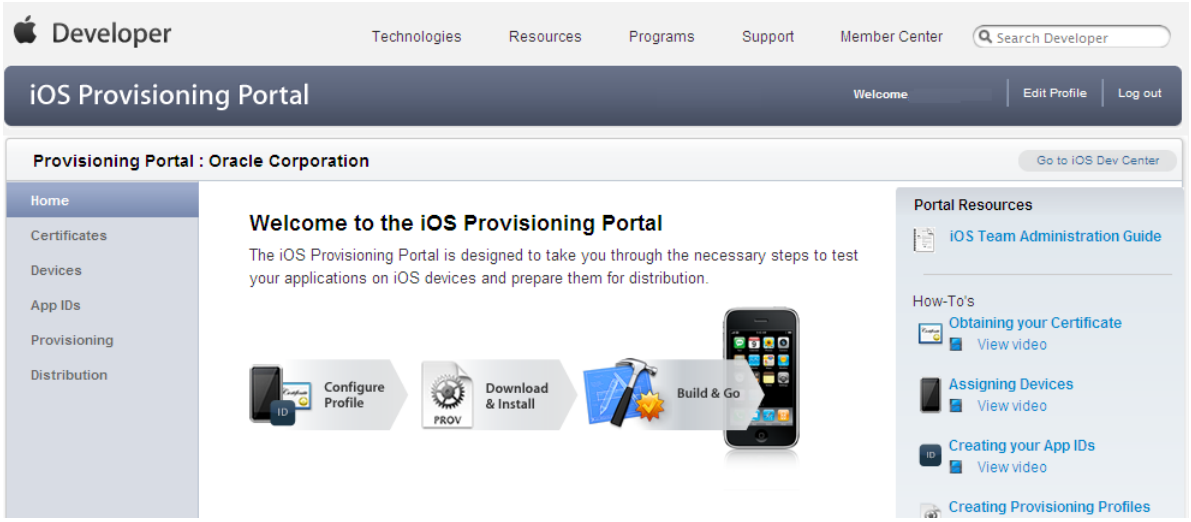
16.4.4 What You May Need to Know About Deploying an Application to an iOS-Powered Device

You cannot deploy an iOS application (that is, an `.ipa` file) to an iOS-powered device or to publish it to either the App Store or to an internal hosted download site without first creating a provisioning profile using the iOS Provisioning Portal, which is accessible only to members of the iOS Development Program. You enter the location of the provisioning profile and the name of the certificate in the Options page as described in Section 16.2.4.2, "Setting the Device Signing Options."

As noted in the "Distributing Apps" chapter in *Tools Workflow Guide for iOS*, a provisioning profile associates development certificates, devices, and an application ID. The iOS Provisioning Portal enables you to create these entities as well as the provisioning profile.

> **Tip:** After you download the provisioning profile, double-click this file to add it to your `Library/MobileDevice/Provisioning Profile` directory.

Figure 16–24 The iOS Provisioning Portal



16.4.4.1 Creating iOS Development Certificates

A certificate is an electronic document that combines information about a developer's identity with a public key and private key. After you download a certificate, you essentially install your identity into the development computer, as the iOS Development Certificate identifies you as an iOS developer and enables the signing of the application for deployment. In the iOS operating environment, all certificates are managed by the Keychain.

Using the Certificates page in the iOS Provisioning Portal, you log a CSR (Certificate Signing Request). The iOS Provisioning Portal issues the iOS Development Certificate after you complete the CSR.

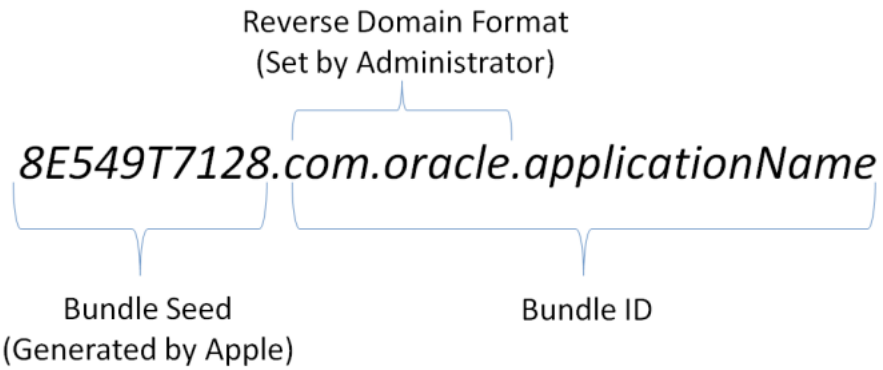
16.4.4.2 Registering an Apple Device for Testing and Debugging

After you install a certificate on your development computer, review the Current Available Devices tab (located in the iOS Provisioning Portal's Devices page) to identify the Apple devices used by you (or your company) for testing or debugging. The application cannot deploy unless the device is included in this list, which identifies each device by its serial number-like Unique Device Identifier (UDID).

16.4.4.3 Registering an Application ID

An application ID is a unique identifier for an application on a device. An application ID is comprised of the administrator-created reverse domain name called a Bundle Identifier in the format described in Section 5.3.1, "How to Set the ID and Display Behavior for a Mobile Application" prepended by a ten-character alpha-numeric string called a bundle seed, which is generated by Apple. Figure 16–25 illustrates an application ID that is unique, one that does not share files or the Keychain with any other applications.

Figure 16–25 An Explicit Application ID



Applications that share files or Keychains (or do not use a Keychain), however, substitute a wildcard character (*) for the application name, such as *8E549T7128.com.oracle.**. Using this format, a suite of applications can share an application ID. For example, if the administrator names *com.oracle.ADF.** on the iOS Provisioning Portal, it enables you to specify different applications (*com.oracle.ADF.application1* and *com.oracle.ADF.application2*).

16.4.5 How to Distribute an iOS Application to iTunes

After you test and debug an application on an iOS device, you can distribute the application to a wider audience through the App Store or an internal download site. To publish an application to iTunes, you must submit the `.ipa` file to iTunes Connect,

which enables you to add `.ipa` files to iTunes, as well as update applications and create test users.

Before you begin:

Before you distribute the application, you must perform the following tasks:

- Test the application on an actual iOS device. See Section 16.4.2, "How to Deploy an Application to an iOS-Powered Device."
- Obtain a distribution certificate through the iOS Provisioning Portal.

> **Note:** Only the Team Agent can create a distribution certificate

- Obtain an iTunes Connect account for distributing the `.ipa` file to iTunes. For information, see "Prepare App Submission" in the iOS Development Center's App Store Resource Center. Specifically, review the *App Store Review Guidelines* to ensure acceptance by the App Review Team.
- You may want to review both the "Distributing Apps" section in *Tools Workflow Guide for iOS* and *iTunes Connect Developer Guide*. These guides are both available through iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).
- In the iOS Options page of the deployment profile, select **Release** as the build mode and then click **OK**.

To distribute an iOS application to iTunes:

1. Choose **Applications**, then **Deploy**, and then select an iOS deployment profile.
2. Choose **Deploy to Distribution Package**.
3. Review the Summary page, which displays the following values. Click **Finish**.
   - **Application Bundle Id**—The unique name that includes a Java language-like package name (*com.<organization name>.<application name>*) prepended with the Bundle Seed that is generated from the iOS Provisioning Portal.
   - **File**—The file name of the final image deployed to an iOS target.
   - **Certificate**—The application's author. If this value has not been configured in the Options page of the deployment profile, then the Summary page displays *<Not Specified>*.
   - **Provisioning Profile**—The name of the provisioning profile that associates one or more development certificates and devices with an application ID. If this value is not configured in the Options page of the deployment profile, then the Summary page displays *<Not Specified>*.

   > **Note:** The Certificate and Provisioning Profile values cannot be noted as *<Not Specified>*; you must specify these values in the Options page to enable the `.ipa` file to be accepted by iTunes.

4. Log in to iTunes Connect.
5. Submit the `.ipa` file to iTunes Connect for consideration using the Manage Your Applications module and the Application Loader described in the "Adding New Apps" and "Using Application Loader" sections in iTunes Connect Developer Guide.

6. After the application has been approved, refer to the "Creating Test Users" section in *iTunes Connect Developer Guide* for information on using the *Manage Users* module. For testing multi-language applications, create a test user account for the regions for which the application is localized.
7. Refer to the "Editing and Updating App Information" section in *iTunes Connect Developer Guide* for information on updating the binary using the *Managing Your Application* module.

16.4.5.1 What Happens in JDeveloper When You Deploy an Application to an iOS Simulator or iOS-Powered Device

JDeveloper performs the following tasks after you complete the deployment wizard:

1. Validates that the necessary software prerequisites are installed. For example, such tasks include:
   a. Validating that the path to the iOS SDK that was set in Preferences is valid and accessible.
   b. Validating that the path to the xcodebuild tool (which is set in the ADF Mobile Platforms page) that was set in Preferences is valid and accessible. The xcodebuild tools is an Apple utility that compiles iOS projects.
   c. Performing other validation as needed.
2. Deletes any pre-existing template for this deployment profile to ensure that any new deployment does not use configuration set for previous deployments.
3. Validates the applications and all of the XML content of both the applications and the projects.
4. For a first-time deployment (or after you select **Build** > **Clean**), JDeveloepr unzips the `Oracle_ADFmc_Container_Template` file for iOS to a temporary location. That is, `modules\adf-mobile\dist\jdev\extensions\oracle.adf.mobile\Oracle_ADFmc_Container_Template.zip` is unpackaged to a profile sub-directory in the `deploy` directory.

   Tip: Selecting **Build** > **Clean** enables you to modify the Xcode project settings, which provide additional options to those available in JDeveloper.

   JDeveloper also unzips the necessary ADF Mobile libraries from `modules\adf-mobile\dist\jdev\extensions\oracle.adf.mobile\Oracle_ADFmc_Framework_Libraries.zip` to the `Framework` directory in the previously unzipped template.

   Note: Any libraries dependent on the type of deployment (that is, debug, release, or deployment to a simulator, device, or as an archive) are added to the template at a later stage.

5. Copies custom images selected by the user to the template location. If custom images have been selected in the deployment profile, JDeveloper copies those images and renames them to the appropriate name. The iOS template internally uses the names `icon.png`, `icon-2x.png`, `icon-72.png`, `Default.png`, `Default-Landscape.png` and `Default-Portrait.png` for the application

icon names and splash screen names. For any image that has not been customized, JDeveloper copies the default image to the template.

6. Copies the web content (including any images related to features) to the template location. As a result, the `public_html` folder and its subdirectories within the ADF Mobile project are copied to the `www` folder in the template.
7. Copies the metadata files to the template location:
   a. Copies the `adfmf-application.xml` file.
   b. Combines all of the `adfmf-feature.xml` files into one file and places it in the template.
8. Creates and updates the `.plist` files. See also Section B.3, "Converting Preferences for iOS."
   a. Creates the preferences `.plist` files from the metadata files. These files create the preferences that are available from the iOS Settings application.
   b. Updates the `root.plist` file with the application name, the bundle identifier, the version, and the flag that determines the use of pre-rendered icons. The application name, bundle identifier, and version are retrieved from the `adfmf-application.xml` file and the iOS Options page, shown in Figure 16–10, "Setting the iOS Options".
9. Runs the xcodebuild utility with the correct flags for the chosen deployment. For example, a debug build deployed to an iOS simulator (Version 5.0) is as follows:

   ```
   xcodebuild clean install -configuration Debug -sdk iphonesimulator5.0
   ```

10. Deploys the newly built application to the proper location.
    a. Copies the iOS application bundle to a unique application directory, as required for deployment to the iOS simulator.
    b. Runs the `xcrun` command to package the application into the `.ipa` file. This command creates an `.ipa` that has a folder entitled `Payload`, which contains the application. It also adds the provisioning profile and signs the application with the developer's name. An example of the `xcrun` command is as follows:

       ```
       /usr/bin/xcrun -sdk iphoneos PackageApplication -v "%PROJECT_
       BUILDDIR%/SampleApp.app" -o "SampleApp.ipa" --sign "iPhone Developer: John
       Doe (A1BC2DEFGH)" --embed "/Users/jdoe/Desktop/my_profile.mobileprovision"
       ```

11. If this is a deployment to a device, JDeveloper moves the `.ipa` file to the `iTunes Media/Automatically Add to iTunes` directory.

16.5 Deploying Feature Archive Files (FARs)

To enable re-use by ADF Mobile view controller projects, application features—typically, those implemented as ADF Mobile AMX or Local HTML— are bundled into an archive known as a Feature Archive (FAR). A FAR is essentially a self-contained collection of application feature elements that can be consumed by ADF Mobile applications, such as icon images, resource bundles, HTML, JavaScript, or other implementation-specific files. (A FAR can also contain Java classes, though these classes must be compiled.) Example 16–1 illustrates the contents of a FAR, which includes a single `adfmf-feature.xml` file and a `connections.xml` file. For more information on `connections.xml`, see the "connections.xml" and "Reusing Application Components" sections in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Example 16–1 Contents of a Feature Archive File

```
connections.xml (or some form of connection metadata)

  META-INF
     adfm.xml
     adfmf-feature.xml
     MANIFEST.MF
     task-flow-registry.xml

  oracle
    application1
      mobile
         Class1.class
         DataBindings.cpx
         pageDefs
           view1PageDefs

  model
     adfc-mobile-config.adfc.diagram
     ViewController-task-flow.adfc.diagram

  public_html
     adfc-mobile-config.xml
       index.html
       navbar-icon.html
       springboard-icon.html
     view1.amx
     ViewController-task-flow.xml
```

Working with Feature Archive files is comprised of the following tasks:

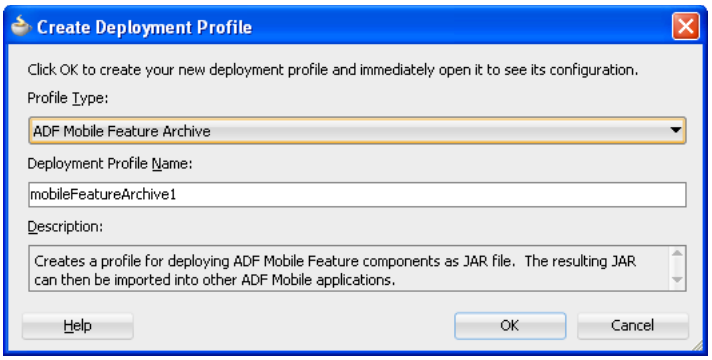
1. Creating a Feature Archive file—You create a Feature Archive by deploying a feature application as a library JAR file.
2. Using the Feature Archive file when creating an ADF Mobile application—This includes importing FARs and re-mapping the imported connection.
3. Deploying an ADF Mobile application that includes features from FARs—This includes unpacking the FAR to a uniquely named folder within the deployment template.

> **Note:** ADF Mobile generates FARs during the deployment process. You only need to deploy a view controller project if you use the FAR in another application.

16.5.1 How to Create a Deployment Profile for a Feature Archive

Use the Create Deployment Profile dialog, shown in Figure 16–9.

Figure 16–26 Create ADF Mobile Feature Dialog



Before you begin:

Create the appropriate connections for the application. See the "Naming Considerations for Connections" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

How to create a deployment profile for a Feature Archive:

1. Right-click a view controller project, choose **New**, then **Deploy**, and then **New Deployment Profile**.

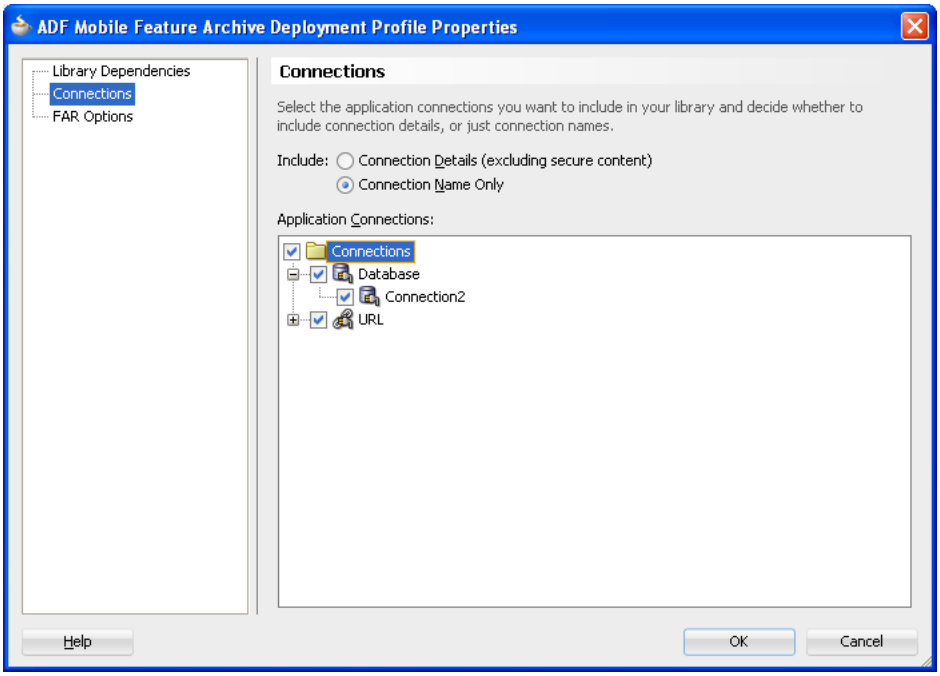
   Note: You do not need to create a separate, application-level deployment profile.

2. Select **ADF Mobile Feature Archive** in the Create Deployment Profile dialog.
3. Enter a profile name, or accept the default, and then click **OK**.

   Note: Name the profile appropriately. Otherwise, you may encounter problems if you upload more than one application feature with the same archive name. For more information, see Section 5.12.3, "What You May Need to Know About Enabling the Reuse of Feature Archive Resources."

4. Select the connections that you want to include in the Feature Archive JAR file, as shown in Figure 16–27.

Figure 16–27 Selecting a Connection for the FAR

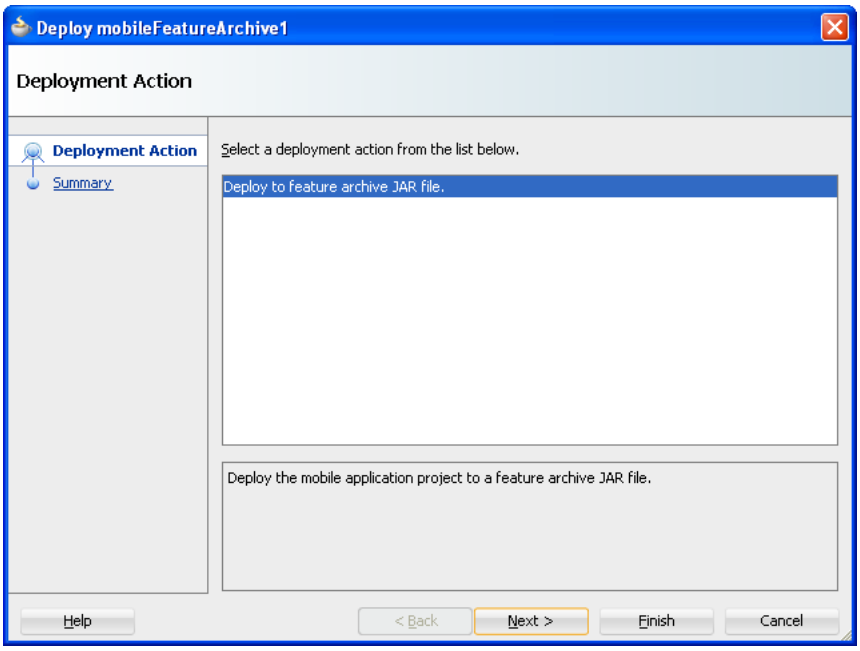


5. Click **Next**, review the options, and then click **Finish**.

16.5.2 How to Deploy the Feature Archive Deployment Profile

The Deployment Actions dialog enables you to deploy the FAR as a JAR file. This dialog, shown in Figure 16–28, includes only one deployment option, **Deploy to feature archive JAR file**.

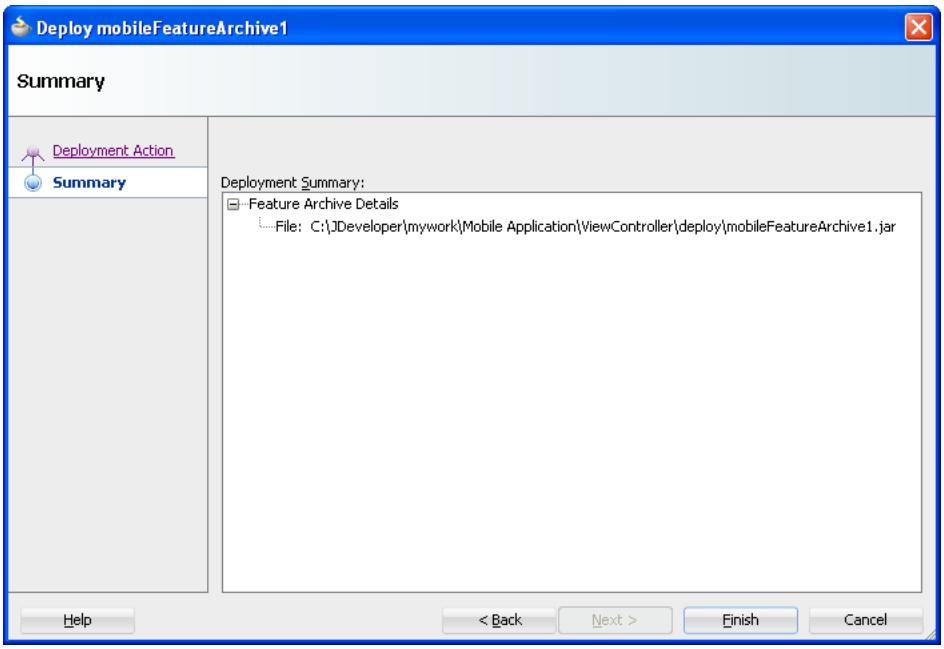
Figure 16–28 Deployment Actions



How to deploy the Feature Archive deployment profile:

1. Right-click the view controller project and then select the Feature Archive deployment profile.
2. Click **Finish**. The Summary page, shown in Figure 16–29, displays the full path of where the Feature Archive file's JAR path is deployed.

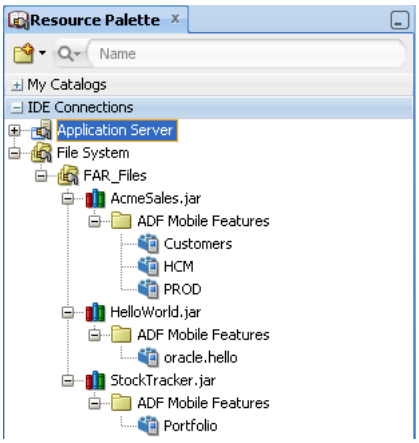
Figure 16–29 Deployment Summary Page



16.5.3 What Happens When You Deploy a Feature Archive File Deployment Profile

After you complete the deployment action dialog, ADF Mobile creates a library JAR in the path shown in the Summary page. To make this JAR available for consumption by other applications, you must first make it available through the Resource Palette, shown in Figure 16–30 (and described in Section 5.12.1, "How to Use FAR Content in an ADF Mobile Application") by creating a connection to the location of the Feature Archive JAR. Figure 16–30 shows Feature Archives that can be made available to an ADF Mobile application through a file system connection. For more information, see "Working with the Resource Palette" in *Oracle Fusion Middleware User's Guide for Oracle JDeveloper*.

Figure 16–30 Deployed Feature Archive JARs in the Resource Palette



16.6 Deploying ADF Mobile Applications from the Command Line

You can deploy iOS or Android deployment files from JDeveloper without starting the JDeveloper IDE using the OJDeploy command line tool. Command line deployment can serve as a tool for testing, as well as a means of deploying applications using a script.

After you have created iOS or Android deployment files using Deployment Profile Properties editor, you can use OJDeploy to deploy applications to iOS simulators and iOS-powered devices (through iTunes), or as iOS bundles (`.ipa` files), or Feature Archive JAR files. Likewise, OJDeploy enables you to deploy applications to both Android emulators and Android-powered devices, or deploy them as Android application package files (`.apk` files) or as Feature Archive JAR files. For information on OJDeploy, see "Deploying from the Command Line" in *Oracle Fusion Middleware User's Guide for Oracle JDeveloper*.

Note: To use OJDeploy on a Mac, add the following line to the `ojdeploy.conf` file:

```
SetSkipJ2SDKCheck true
```

This file is located at: *`jdev_install`*`/jdeveloper/jdev/bin`

16.6.1 Using OJDeploy to Deploy ADF Mobile Applications

The following commands enable you to deploy ADF Mobile deployment profiles:

- `deployToDevice`—Deploys an application to iOS- or Android-powered devices. For iOS applications, this command is used in debugging scenarios where the application is deployed to a device using iTunes. For more information, see Section 16.4.5, "How to Distribute an iOS Application to iTunes."
- `deployToSimulator`—Deploys an application to an iOS simulator or Android emulator. You can only deploy an ADF Mobile application to an iOS simulator on an Apple computer.
- `deployToPackage`—Deploys an iOS application as an `.ipa` file or an Android application as an `.apk` file. You can only package an application as an `.ipa` file on an Apple computer.
- `deployToFeatureArchive`—Deploys a Feature Archive to a JAR file.

You use these commands in conjunction with the `ojdeploy` command line tool, OJDeploy's arguments, and its options as follows:

```
ojdeploy deployToSimulator -profile <profile name> -workspace <jws file location>
```

Note: OJDeploy commands and arguments are case-sensitive.

Table 16–7 lists the OJDeploy arguments that you use to modify the ADF Mobile deployment commands.

Tip: Using the `-help` option with any command (such as `ojdeploy deployToSimulator -help`) retrieves usage and syntax information.

Table 16–7 OJDeploy Arguments for ADF Mobile Deployments

Argument	Description
`-profile`	The name of the Android or iOS deployment profile. For example: `ojdeploy deployToSimulator -profile iosDeployProfile ...`
`-workspace`	The full path to the ADF Mobile application file (`.jws`). For example: `... -workspace /usr/jsmith/mywork/Application1/Application1.jws`
`-project`	For the `deployToFeatureArchive` command, you must provide the name of the project (that is, a view controller project) that contains the Feature Archive deployment profile. For example: `ojdeploy deployToFeatureArchive -profile farProfileName -project ViewController ...`

In addition to the arguments listed in Table 16–7, you can also use OJDeploy options described in the "Command Usage" section of *Oracle Fusion Middleware User's Guide for Oracle JDeveloper*.

Note: The following options are not supported:

- `-forcerewrite`
- `-nocompile`
- `-nodatasources`
- `-nodepdendents`
- `-outputfile`
- `-updatewebxmlejbrefs`

Table 16–8 provides examples of how to use the OJDeploy options with the ADF Mobile deployment commands.

Table 16–8 OJDeploy Options for ADF Mobile Deployments

Option	Description
`-clean`	Deletes all files from the project output directory before compiling. For example: `ojdeploy deployToSimulator -profile iosDeployProfile -workspace /usr/jsmith/jdeveloper/mywork/Application1.jws -clean`
`-stdout, -stderr`	Redirects the standard output and error logging streams to a file for each profile and project. For example: `ojdeploy deployToSimulator -profile iosDeployProfile -workspace /usr/jsmith/jdeveloper/mywork/Application1.jws -clean -stdout /usr/jsmith/stdout/stdout.log -stderr /usr/jsmith/stderr/stderr.log`

17

ADF Mobile Application Security

This chapter provides an overview of the security framework within ADF Mobile and also describes how to configure ADF Mobile applications to participate in security.

This chapter includes the following sections:

17.1 Introduction to Security for an ADF Mobile Application

The ADF Mobile runtime authenticates end users when a secured application feature has been activated, such as when it is about to be displayed within the web view or when the operating system returns the application feature to the foreground. In these instances, ADF Mobile determines whether access to the application feature requires user authentication and then challenges the user with a login page. Only when the user successfully enters valid credentials does ADF Mobile render the intended web view, UI component, or application page.

ADF Mobile uses JavaScript, a PhoneGap plugin, and native PhoneGap command handlers that collectively handle the interaction with the login page, the navigation between application features, and interactions with the Oracle Identity Connect's IDM Mobile SDK, whose classes and protocols verify user credentials. The native PhoneGap command handler methods are executed through the PhoneGap plugins. The methods' results are sent back to corresponding JavaScript callback functions that navigate to either the requested application feature or to the web view that initiated the authentication process.

> **Note:** The entire login process is executed within ADF Mobile. Embedded Java is not required.

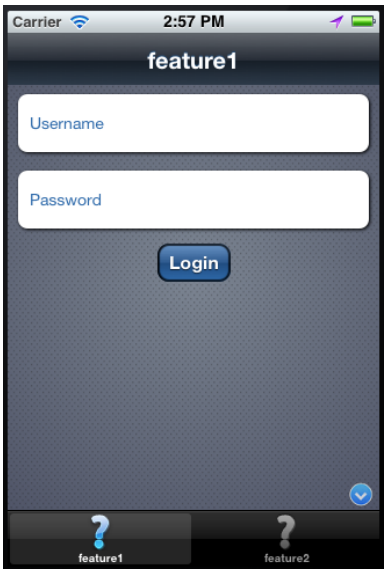
An ADF Mobile application uses either the default page or a customized login page that is written in HTML.

17.2 Introduction to the User Login Process

Login is required when a `credential` attribute for the application feature definition (in the `adfmf-feature.xml` file) is designated as either remote or local and the user has not been authenticated within the timeout period. From the end-user perspective, the login process is as follows:

1. ADF Mobile presents a web view of a login page, shown in Figure 17–1. The login page illustrates the name of the application feature that requires authentication, *feature1*.

Figure 17–1 The Login Page



> **Note:** As described in Section 17.4.10, "What Happens When You Designate a Custom Login Page," ADF Mobile provides not only a login page, but also supports a custom login page.

2. The user enters a user name and password and then clicks **OK**.

> **Note:** ADF Mobile does not allow multiple users for the same application without restarting that application. If a user attempts to login to an application after a previous user was logged out because of a timeout, ADF Mobile prompts the user to enter the initial user's name and password or to restart the application. A user can restart an application by logging off, because logging off typically terminates an application automatically. You must enable end users to terminate the application by logging off from the login screen.
>
> For information on calling methods of the `AdfmfJavaUtilities` class to enable users to log off of an application without it terminating, see Section 17.4.11, "What You May Need to Know About Login Pages."

3. If the user name and password are verified, ADF Mobile displays the intended web view, page, or UI component.

4. ADF Mobile presents challenges to the user name and password until the user logs in successfully. When users cannot login, they can only navigate to another application feature.

> **Note:** Authentication times out when a predefined time period has passed since the last activation of an application feature. ADF Mobile only renews the timer for the idle timeout when one of the application features that uses the connection to the authentication server has been activated.

17.3 Introduction to Authentication

ADF Mobile authenticates against an authentication server, such as Oracle Access Management (OAM) Identity Server.

> **Note:** ADF Mobile can authenticate against any basic authentication server.

When ADF Mobile handles the authentication against a remote server, the flow of security is as follows:

1. ADF Mobile presents the user with the default login page, similar to the one shown in Figure 17–1, or with a custom login page. The login page displays for applications that have been deployed because of the security-related parameters configured in the `adfmf-application.xml` and `adfmf-feature.xml` files. For more information, see Section 17.4, "Configuring Security for ADF Mobile Applications."
2. The Oracle Identity Connect IDM Mobile SDK APIs handle the authentication for both remote authentication servers and for authentication against the credential store on the device (which may or may not store a saved user object). Depending on the success or failure of the authentication, the APIs return either a failure or a valid user object to ADF Mobile.
3. If the login succeeds, ADF Mobile receives an OAM token used in the cookie. ADF Mobile sets the cookie into each login connection.

> **Note:** When at least one secured application feature includes the `user.roles` or `user.privileges` constraints, an end user must authenticate against an application login server when the application starts. Otherwise, the default application feature displays. If a default application feature participates in security, then the end user is prompted for authentication against the login connection associated with the application feature. Instead of authenticating, an end user can also navigate to another application feature. For more information, see Section 14.2.3.1, "User Constraints."

4. Oracle Identity Connect IDM Mobile SDK APIs save the credentials in the device's credential store.
5. If the login fails, the login page remains, thereby preventing users from continuing.

17.4 Configuring Security for ADF Mobile Applications

Security is configured both at the application feature and mobile application levels using the overview editors for the `adfmf-feature.xml` and `adfmf-application.xml` files. At the application feature level, you designate a security requirement for any application feature that requires authentication against a remote login server, or against a credential store on the device that houses the user's login credentials for a remote login server. In the former case, a mobile application may require user authentication against Oracle Access Manager used by Oracle ADF Fusion web applications. After the user is authenticated against the authentication server within the same application session (that is, within the lifecycle of the application execution), the authentication context is stored locally and subsequent authentication is executed against this local authentication context. A subsequent authentication does not attempt to contact the authentication server if the local authentication context is sufficient to authenticate the user. Although a connection to the remote login server is required for the initial authentication, continual access to the server is not required for local authentication. In addition, authentication against a local credential store can be faster than authentication against a remote login server.

Designating security for an application feature results in JDeveloper populating a `credential` attribute to the application feature's `<adfmf:feature>` element. When you embed such an application feature in a mobile application, you then configure login and logout server connection information and designate whether users are presented with the default login page provided by ADF Mobile or with a customized page. These requirements are noted by the attributes for the `<adfmf:login>` element.

Login is required when the credential attribute for the application feature definition (in the `adfmf-feature.xml` file) is either designated as remote or local (as described in Section 17.4.1, "How to Enable Application Features to Require Authentication") and when the user has not been authenticated within a timeout period set in the Create ADF Mobile Login Connection dialog, shown in Figure 17–7. See also Section 17.1, "Introduction to Security for an ADF Mobile Application."

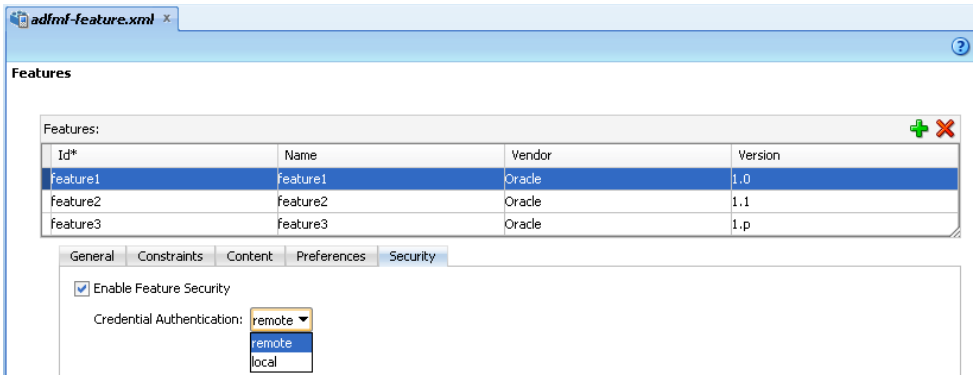
Note: You must define at least one connection to the application login server for an application feature that participates in security remotely through a login server or locally using a stored set of credentials. The absence of a defined connection to an application login server results in an invalid configuration. As a result, the application will not function properly.

17.4.1 How to Enable Application Features to Require Authentication

Configuring security for an ADF Mobile application begins at the application feature level, where you designate which application features require users to authenticate against a login server or to a local credential store. You must define each application feature to participate in security.

The Credentials tab of the `adfmf-feature.xml` overview editor, shown in Figure 17–2, enables you to designate which application features participate in security and the type of authentication they require.

Figure 17–2 Designating User Credentials Options for an Application Feature

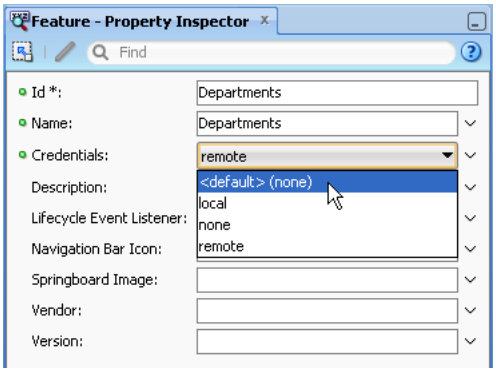


To designate user access for an application feature:

1. Select an application feature or click **Add** to add an application feature.
2. Click **Enable Feature Security** for any application feature that requires login and then select one of the following authentication options:
   - **local**—Select if the application will allow users to authenticate against locally stored credentials on the device. After the user's first successful authentication to a remote server, ADF Mobile persists the credentials locally within a credential store in the device. These credentials are used for subsequent access to the application feature. See also Section 17.4.9, "What You May Need to Know about Web Service Security."
   - **remote**—Select if the application requires authentication with a remote login server, such as Oracle Access Manager (OAM) Identity Server or a secured web application. Authentication against the remote server is required each time a user logs in. If the device cannot contact the server, then a user cannot login to the application despite a previously successful authentication.

In addition to the overview editor shown in Figure 17–2, you can configure the security options in the Property Inspector, shown in Figure 17–3.

Figure 17–3 Designating User Credential Options in the Property Inspector



In addition to the local and remote options available in the overview editor, the Property Inspector also provides the **none** and **<default> (none)** options for application features that do not participate in security. Selecting **none** clears the **Enable Feature Security** option in the overview editor and updates `<adfmf:feature>` with `credentials="none"`. Selecting **<default> (none)** also

clears **Enable Feature Security** and removes the `credentials` attribute from `<admf:feature>`, which has no `credentials` attribute by default.

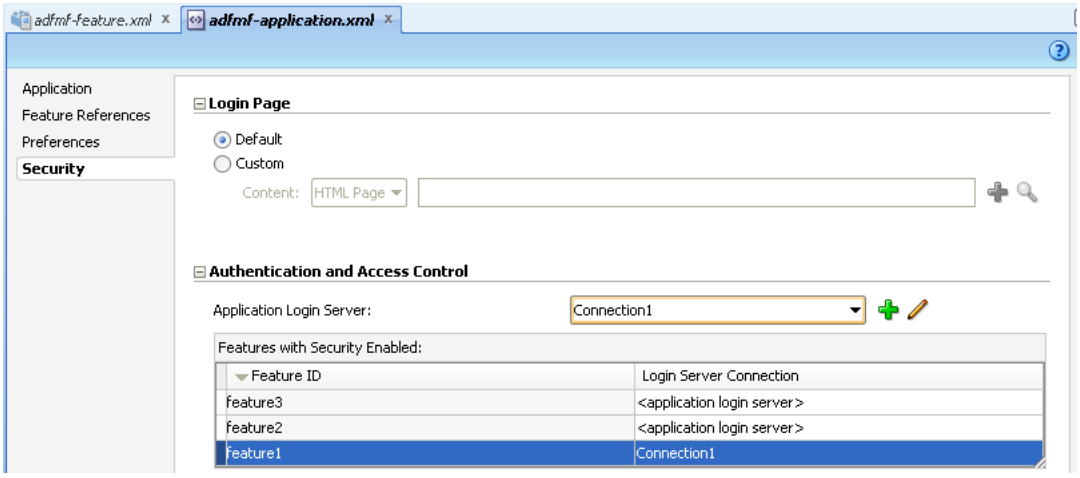
17.4.2 How to Designate the Login Page

After you designate security for the application features, you use the Security page of the `adfmf-application.xml` overview editor, shown in Figure 17–4, to configure the login page as well as create and assign a connection to the login server for each of the application features that participate in security. All of the application features listed in this page have been designated in the `adfmf-feature.xml` file as requiring security. Typically, a group of application features are secured with the same login server connection, enabling users to open any of these applications without ADF Mobile prompting them for subsequent logins. In some cases, however, the credentials required for the application features can vary, with one set of application features secured by one login server and another set secured by a second login server. To accommodate such situations, you can define any number of connections to a login server for an ADF Mobile application. In terms of the `adfmf-application.xml` file, the authentication server connections associated with the feature references are designated using the `loginConnRefId` attribute as follows:

```
<adfmf:featureReference id="feature1" loginConnRefId="Connection_1"/>
<adfmf:featureReference id="feature2" loginConnRefId="Connection2"/>
```

ADF Mobile applications can be authenticated against any standard login server that supports basic authentication over HTTP or HTTPS. ADF Mobile also supports authentication against Oracle Identity Management. You can also opt for a custom login page for a specific application feature.

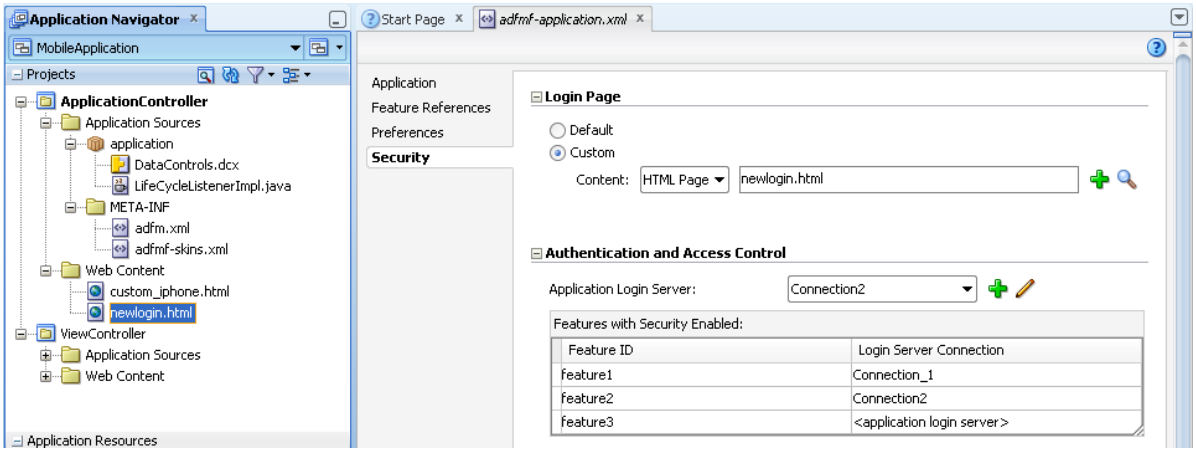
Figure 17–4 The Security Page



Before you begin:

If the ADF Mobile application uses a custom login page, add the file to the `public_html` directory of the application controller project (`JDeveloper\mywork\`*`Application`*`\ApplicationController\public_html`) to make it available from the **Web Content** node in the Application Navigator, as shown in Figure 17–5. See also Section 5.9.2, "What You May Need to Know About Selecting External Resources."

Figure 17–5 Adding a Custom Login Page



To designate the login page:

1. Click **Security**.
2. Select the login page type:
   - **Default**—The default login page used for all of the selected embedded application features. For more information, see Section 17.4.11.1, "The Default Login Page." The default login page is provided by ADF Mobile.
   - **Custom**—Select the format (HTML) and then click **Browse** to retrieve the path location of the file within the application controller project. Alternatively, click **New** to create an HTML page within the application controller project. For more information, see Section 17.4.11.2, "The Custom Login Page" and Section 17.4.11.3, "Creating a Custom Login HTML Page for an iOS-Powered Device."

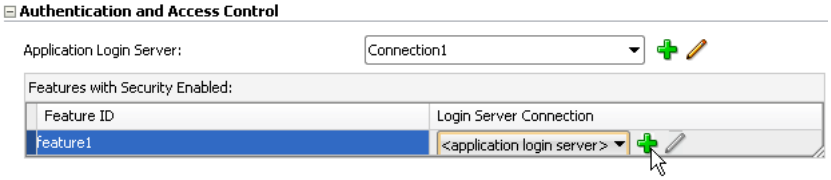
     Tip: Rather than retrieve the location of the login page using the **Browse** function, you can drag the login page from the Application Navigator into the field.

To set the connection to the remote server:

You must first create the authentication-related connections as follows:

1. In the Features with Security Enabled table, select the application feature's Server Connection field and then click **Add**, as shown in Figure 17–6.

Figure 17–6 Adding a Server Connection



Note: You must define the application login server connection and assign it to the default application feature (if the default application feature is secured). Also, the credentials used for the application login server are also used to retrieve user and roles and services through the Access Control Service (ACS). See also Section 17.4.6, "What You May Need to Know About the Access Control Service."

---

2. Complete the Authentication tab of the Create ADF Mobile Login Server Connection dialog as follows:

   - **Name**—Enter a name for the connection.
   - **Login URL**—Enter the login URL for the authentication server.
   - **Logout URL**—Enter the logout URL for the authentication server.
   - **Idle Timeout**—Enter the time for an application feature to remain idle after ADF Mobile no longer detects the activation of an application feature. After this period expires, the user is timed-out of all the application features that are secured by the login connection. ADF Mobile then presents the user with a login page. By default, ADF Mobile presents the user a login page when an application remains idle for 300 seconds (five minutes).

     Note: ADF Mobile authenticates against the local credential store after an idle timeout, but does not perform this authentication after a session timeout.

   - **Session Timeout**—Enter the time, in seconds, that a user can remain logged in to an application feature. After the session expires, ADF Mobile prompts the user with a login page if the idle timeout period has not expired. By default, a user session lasts for 28,800 seconds (eight hours).

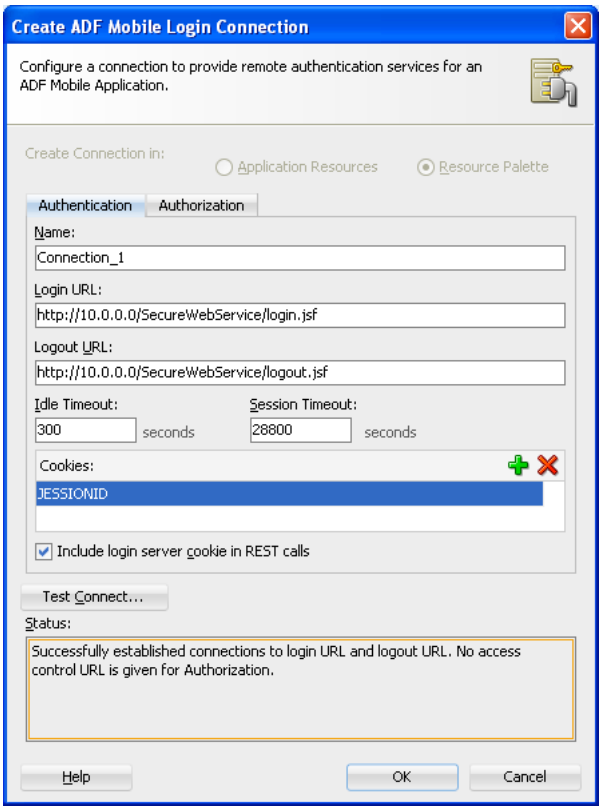
     Note: You must edit `connections.xml` to set the number of failed login attempts allowed to a user. For more information, see Section 17.4.4, "What You May Need to Know About Configuring ADF Mobile to Respond to Unsuccessful Login Attempts."

   - **Cookies**—Use for a remote login for a secured application feature. Enter a list of cookie names that enable basic authentication after they are stored. Because ADF Mobile stores the cookies in a browser, users need only log in once. These cookies and the credentials stored within them are provided by a system administrator. The cookie names are case-sensitive.
   - **Include login server cookie in REST calls**—For application features using remote authentication (that is, the `<adfmf:feature>` element includes `credentials="remote"` as described in Section 17.4.1, "How to Enable Application Features to Require Authentication"), select to enable a REST web service to retrieve authorized user data stored on a login server by using the login server-generated user session cookie. For more information, see Section 17.4.8, "What You May Need to Know About Injecting Cookies into REST Web Service Calls."

Note: To enable cookies to be injected in the REST web service call:

- The application feature must use remote authentication; ADF Mobile does not support injecting cookies for application features that store user credentials locally.
- The domain entered for **Login URL** must be identical to the domain of the REST web service end point.
- The cookies entered expected by the REST web service, which are entered into **Cookies** field, must be identical to the ones returned by the login server when the user logs in to an application feature.

Figure 17–7 The Create ADF Mobile Login Connection Dialog



Note: Figure 17–7 shows a cookie for JSESSIONID, which is used for HTTP basic authentication against Oracle WebLogic Server. Because other authentication mechanisms require different cookies, you can obtain the cookie name through network tracing or through the HTTP Analyzer, described in the "Monitoring HTTP Using the HTTP Analyzer" in *Oracle Fusion Middleware User's Guide for Oracle JDeveloper*.

Tip: To authenticate against a resource other than OAM, configure both **Login URL** and **Logout URL** as the URL of the authentication resource, and enter JSESSIONID as the cookie.

3. Complete the Authorization tab, shown in Figure 17–8, of the Create ADF Mobile Login Server Connection dialog if the application feature contains any component that is secured, such as an expense report application that includes a submit button configured with an EL expression that only renders to `true` if it is clicked by a user who is granted a role of *manager*. Completing the fields in this tab enables the retrieval of the specific user roles that are checked by an application feature.

   The access control granted by the application login server is based on the evaluation of the `user.roles` and `user.privileges` constraints configured for an application feature, as described in Section 14.2.3.1, "User Constraints." For example, to allow only a user with *manager_role* role to access an application feature, you must define the `<adfmf:constraints>` element `adfmf-feature.xml` with the following:

```
<adfmf:constraint property="user.roles"
                  operator="contains"
                  value="manager_role"/>
</adfmf:constraints>
```

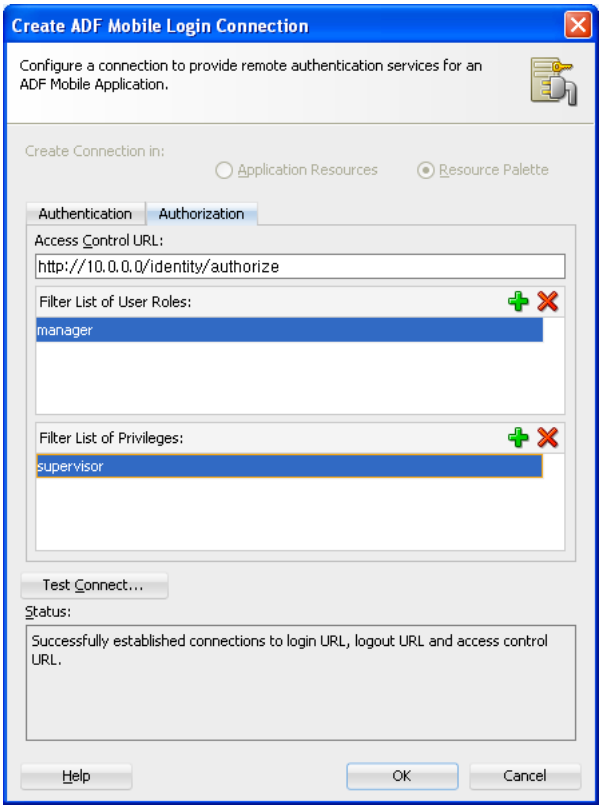
   At the start of application, the RESTful web service known as the Access Control Service (ACS) is invoked for the application login server connection and the roles and privileges assigned to the user are then fetched. ADF Mobile then challenges the user to login against the application login server connection.

   ADF Mobile evaluates the constraints configured for each application against the retrieved user roles and privileges and makes only the application features available to the user that satisfy all of the associated constraints.

   Complete the authorization requirements as follows:

   - **Access Control URL**—Enter the URL that is the endpoint for the Access Control Service, as shown in Figure 17–8.
   - **Filter List of User Roles**—Enter the user roles checked by the application feature. Because there may be thousands of user roles and privileges defined in a security system, use the manifest provided by the application feature developer that lists the roles specific to the application feature to create this list.
   - **Filter List of Privileges**—Enter the privileges checked by the application feature.

Figure 17–8 Configuring Access Control



4. Click **OK**.

Note: By default, all secured application features share the same connection, which, as shown in Figure 17–4, is denoted as *<application login server>*. The Property Inspector for a Feature Reference notes this default option in its Login Server Connection dropdown menu as `<default> application login server`. You can select other connections that are defined for the ADF Mobile application using the Create ADF Mobile Login Connection dialog.

17.4.3 What Happens When You Create an ADF Mobile Connection

ADF Mobile aggregates all of the connection information in the `connections.xml` file (located in the Application Navigator's Application Resources panel under the **Descriptors** and **ADF META-INF** nodes). This file, shown in Example 17–1, can be bundled with the application or can be hosted for the Configuration Service. In the case of the latter, ADF Mobile checks for the updated configuration information each time an application starts.

Example 17–1 ADF Mobile Connections Defined in connections.xml

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<References xmlns="http://xmlns.oracle.com/adf/jndi">
  <Reference name="Connection_1"
             className="oracle.adf.model.connection.adfmf.LoginConnection"
             adfCredentialStoreKey="Connection_1"
             partial="false"
             manageInOracleEnterpriseManager="true"
```

```
            deployable="true"
            xmlns="">
      <Factory className="oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
      <RefAddresses>
         <XmlRefAddr addrType="adfmfLogin">
            <Contents>
               <login url="http://10.0.0.0/SecuredWebServicelogin/login.jsf"/>
               <logout url="http://10.0.0.0/SecuredWebServicelogout/logout.jsf"/>
               <accessControl url="http://10.0.0.0/Identity/Authorize"/>
               <idleTimeout value="300"/>
               <sessionTimeout value="28800"/>
               <cookieNames>
                  <cookie name="JSESSIONID"/>
               </cookieNames>
               <injectCookiesToRESTHttpHeader value="true"/>
               <userObjectFilter>
                  <role name="manager"/>
                  <privilege name="account manager"/>
                  <privilege name="supervisor"/>
                  <privilege name=""/>
               </userObjectFilter>
            </Contents>
         </XmlRefAddr>
      </RefAddresses>
   </Reference>
</References>
```

For more information, see the "connections.xml" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

17.4.4 What You May Need to Know About Configuring ADF Mobile to Respond to Unsuccessful Login Attempts

You can set the maximum number of failed login attempts allowed for a user by adding the `max_retries` attribute to the `connections.xml` file, as shown in Example 17–2. If you do not configure this attribute, then ADF Mobile allows ten retries. The application closes automatically after a user exceeds the set number of login attempts.

By default, ADF Mobile grants a user three unsuccessful login attempts before it clears the user's locally stored credentials and contacts the remote login server for subsequent login attempts. By adding the `maxFailuresBeforeCredentialCleared` attribute to the `connections.xml` file, as shown in Example 17–2, you can change the number of login attempts that ADF Mobile allows a user before it purges the locally stored credentials.

Note: ADF Mobile clears locally stored user credentials even when the application feature is configured to use local authentication in the `adfmf-feature.xml` file.

Example 17–2 Configuring the Maximum Retries in connections.xml

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<References xmlns="http://xmlns.oracle.com/adf/jndi">
  <Reference name="MyAuth"
           className="oracle.adf.model.connection.adfmf.LoginConnection"
           adfCredentialStoreKey="MyAuth"
           partial="false"
```

```
          manageInOracleEnterpriseManager="true"
          deployable="true" xmlns="">
        <Factory className="oracle.adf.model.connection.adfmf.LoginConnectionFactory"/>
        <RefAddresses>
            <XmlRefAddr addrType="adfmfLogin">
                <Contents>
                    <login url="http://10.0.0.0:7777/SecuredWeb1-ViewController-context-root/
                                faces/view1.jsf"/>
                    <logout url="http://10.0.0.0:7777/SecuredWeb1-ViewController-context-root/
                                 faces/view1.jsf"/>
                    <accessControl url="http://myhost.us.example.com:7777/
                                       UserObjects/jersey/getUserObjects" />
                    <idleTimeout value="10"/>
                    <sessionTimeout value="36000"/>
                    <cookieNames>
                        <cookie name="JSESSIONID"/>
                    </cookieNames>
                    <max_retries value="6"/>
                    <maxFailuresBeforeCredentialCleared value="3"/>
                </Contents>
            </XmlRefAddr>
        </RefAddresses>
  </Reference>
</References>
```

17.4.5 What Happens in JDeveloper When You Configure Security

Selecting the **Enable Feature Security** option in the `adfmf-feature.xml` overview editor and then selecting the subsequent type of authentication updates the `<adfmf:feature>` element with a defined `credentials` attribute. For example:

```
<adfmf:feature id="feature1" name="feature1" credentials="remote">
<adfmf:feature id="feature2" name="feature2" credentials="local">
```

> **Note:** Application features that do not require security (that this, those for which these security options have not been selected) are available to all users of the ADF Mobile application. These applications either have no `credentials` attribute defined (the default setting, which is present prior to selecting **Enable Feature Security** in the `adfmf-feature.xml` oveview editor, shown in Figure 17–2) or include `credentials="none"` (an attribute that is populated automatically when you clear **Enable Feature Security**).

After an application feature has been designated to participate in security, JDeveloper updates the Features With Security Enabled table with a corresponding feature reference, shown in Figure 17–4. If each of the referenced application features authenticate against the same login server connection defined in the `connections.xml` file, JDeveloper updates the `adfmf-application.xml` file with a single `<adfmf:login>` element defined with a `defaultConnRefId` attribute (such as `<adfmf:login defaultConnRefId="Connection_1">`). For application features configured to use different login server connections defined in the `connections.xml` file JDeveloper updates each referenced application feature with a `loginConnReference` attribute (`<adfmf:featureReference id="feature2" loginConnRefId="Connection2"/>`). For more information, see Section 17.4.1,

"How to Enable Application Features to Require Authentication." See also *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*.

17.4.6 What You May Need to Know About the Access Control Service

The Access Control Service (ACS) is a RESTful web service that enables users to download their user roles and privileges through a single HTTP `POST` message. This is a request message, one which returns the roles or privileges (or both) for a given user. It can also return specific roles and privileges by providing lists of required roles and privileges. The request message is comprised of the following:

- Request header fields: `If-Match`, `Accept-Language`, `User-Agent`, `Authorization`, `Content-Type`, `Content Length`.
- A request message body: A request for user information.
- The requested JSON object that contains:
  - `userId`—The user ID.
  - `filterMask`—A combination of `"role"` and `"privilege"` elements are used to determine if either the filters for user role or privilege should be used.
  - `roleFilter`—A list of roles used to filter the user information.
  - `privilegeFilter`—A list of privileges used to filter the user information.

> **Note:** If all of the roles should be returned, then do not include the `"role"` element in the `filterMask` array.
>
> If all of the privileges should be returned, then do not include the `"privilege"` element in the `filterMask` array.

Example 17–3 illustrates an HTTP `POST` message and identifies a JSON object as the payload, one that requests all of the filters and roles assigned to a user, John Smith.

Example 17–3 The ACS Request for User Roles and Privileges

```
Protocol: POST
Authoization: Basic xxxxxxxxxxxx
Content-Type: application/json

{
        "userId": "johnsmith",
        "filterMask": ["role", "privilege"],
        "roleFilter": [ "role1", "role2" ],
        "privilegeFilter": ["priv1", "priv2", "priv3"]
}
```

The response is comprised of the following:

- A response header with the following fields: `Last-Modfied`, `Content-Type`, and `Content-Length`.
- A response message body that includes the user information details.
- The returned JSON object, which includes the following:
  - `userId`—the ID of the user.

- `roles`—A list of user roles, which can be filtered by defining the `roleFilter` array in the request. Otherwise, the response returns an entire list of roles assigned to the user.
- `privileges`—A list of the user's privileges, which can be filtered by defining the `privilegeFilter` array in the request. Otherwise, the response returns an entire list of privileges assigned to the user.

Example 17–4 illustrates the returned JSON object that contains the user name and the roles and privileges assigned to the user, John Smith.

Example 17–4 The Returned JSON Object

```
Content-Type: application/json

{
        "userId": "johnsmith",
        "roles": [ "role1" ],
        "privileges": ["priv1", "priv3"]
}
```

Note: There are no login screens for web services; user access is instead enabled by ADF Mobile, which automatically adds credentials to the header of the web service. For more information, see Section 10.5.3, "What You May Need to Know About Credential Injection."

17.4.7 What Happens When You Enable Cookie Injection into REST Web Service Calls

Each time an ADF Mobile application requests an REST web service, ADF Mobile's security framework enables the transport layer of the REST web service to check if cookie injection is enabled for the login connection associated with the URL endpoint of the REST web service. (That is, the `connections.xml` file must include `<injectCookiesToRESTHttpHeader value="true"/>`, shown in Example 17–1.) If the connection allows cookie injection, and if the domains for the login server and the REST web service endpoint are identical, then the security framework retrieves the cookies stored when a user has logged in to an application feature and returns a string in the form of `"<cookie name1>=<cookie value1>;<cookie name2>=<cookie value2>;...<cookie nameN>=<cookie valueN>"`. The security framework creates a cookie header and injects this string into it only if the domain configured for the login server is identical URL endpoint of the REST web service. Further, the string of stored cookies, which are expected by the REST web service, must match those returned from the login server.

Note: ADF Mobile constructs the cookie string by calling the IDM Mobile SDK API, which returns cookies by name from a platform-specific cookie store. The IDM Mobile SDK manages the cookies returned by authentication servers, the names of which are defined in the `connections.xml` file.

17.4.8 What You May Need to Know About Injecting Cookies into REST Web Service Calls

After a user has been successfully authenticated by an ADF Mobile application, the login server creates the security context for the user and generates a cookie that tracks the user session. If you selected the **Include login server cookie in REST call** option in the Create ADF Mobile Login Connection dialog, shown in Figure 17–2, you instruct ADF Mobile to retrieve this user session cookie sent by the login server and then inject it into the HTTP header of the REST web service that originated from the ADF Mobile application. Propagating the cookie to the web service call enables the retrieval of the user's security context, which is stored on the login server, and enables the ADF Mobile application to use the REST web service to access the application data authorized for the user. After the user session cookie expires, ADF Mobile challenges the user for credentials and then re-authenticates the user. A user that has been re-authenticated can continue to access the authorized application data through the REST web service call.

17.4.9 What You May Need to Know about Web Service Security

There are no login screens for web services; user access is instead enabled by ADF Mobile injecting credentials into the header of the web service. Web services gain access to application data using the locally stored credentials persisted by ADF Mobile after the first successful login to the authentication server. The name of the local credential store is reflected by the `adfCredentialStoreKey` attribute of the login server connection (such as `adfCredentialStoreKey="Connection_1"` in Example 17–1). To enable a web service to use this credential store, the name defined for the `credentialStoreKey` attribute of a SOAP or REST web service connection must match the name defined for the login server's `adfCredentialStoreKey` attribute.

> **Note:** Because there is no overview editor for the `connections.xml` file, you can use the Property Editor to update the `<Reference>` element's `credentialStoreKey` attribute with the name configured for `adfCredentialStoreKey` attribute of the login server connection. Alternatively, you can add or update the attribute using the Source editor.

For more information, see Section 10.5.3, "What You May Need to Know About Credential Injection."

17.4.10 What Happens When You Designate a Custom Login Page

When you add a custom login page for a selected application feature, JDeveloper adds the `<adfmf:login>` element and populates its child `<adfmf:LocalHTML>` element, as shown in Example 17–5. As with all `<adfmf:LocalHTML>` elements, its `url` attribute references a location within the `public_html` directory.

Example 17–5 The Login Element

```
<adfmf:login defaultConnRefId="Connection_1">
    <adfmf:localHTML url="newlogin.html"/>
  </adfmf:login>
```

For more information on the <adfmf:localHTML> element, see *Oracle Fusion Middleware Tag Reference for Oracle ADF Mobile*. See also Section 5.9.2, "What You May Need to Know About Selecting External Resources."

17.4.11 What You May Need to Know About Login Pages

The entry point for the authentication process to an application feature is the `activate` lifecycle event, described at Section 5.6.2, "Timing for Mobile Application Events." Every time an application feature is activated (that is, the `activate` event handler for the application feature is called), the application feature login process is executed. This process navigates to the login page (which is either the default or a custom login page) where it determines if user authentication is needed. Before the process navigates to the login page, however, the originally intended application feature must be registered with ADF Mobile. When authentication succeeds, the login page retrieves the originally intended destination from ADF Mobile and navigates to it.

17.4.11.1 The Default Login Page

The default user interface is an HTML file that provides a login button and input text fields for the user name and password.

17.4.11.2 The Custom Login Page

The custom login user interface can be implemented in a login page specified in the `adfmf-application.xml` file The user authentication mechanism and navigation control are identical to the default login page.

ADF Mobile provides JavaScript APIs for accessing relevant security features. Because you implement a custom login page as an HTML page, you can enable ADF Mobile to process user commands by adding the call to the `adf.mf.api.invokeSecurity` method, which is described in Table 17–1.

Table 17–1 The adf.mf.api.invokeSecurityMethod

Security Method	Return Value	Parameters Passed	Function
`adf.mf.api.invokeSecurityMethod ("login", username, password)`	None	`username`, `password`	Authenticates the user and provides appropriate program flow execution according to the authentication result.

To enable logging out, call the `logout` method of the `AdfmfJavaUtilities` class as follows. For example:

```
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
...
   AdfmfJavaUtilities.logout();
```

You can invalidate the credentials (either remote or stored locally) that are specific to an application using the `clearPasswordCredential` method of the `adfmfJavaUtilities` class as illustrated in Example 17–6.

Example 17–6 Clearing User Credentials

```
import oracle.adfmf.framework.api.AdfmfJavaUtilities;
...
   AdfmfJavaUtilities.clearPasswordCredential(adfCrendentialStorykey, username);
```

The `adfCredentialStorykey` parameter is returned as a `String` from the value defined for the `adfCredentialStoreKey` parameter in the `connections.xml` file. The method's `username` parameter is returned as a `String` and represents the user whose credentials will be removed from the credential store.

By clearing this set of credentials, users can maintain their sessions in other secured application features or open a non-secured application feature without exiting the ADF Mobile application.

For more information on the `AdfmfJavaUtilties` class and the usage of the `key` parameter, see *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile*.

17.4.11.3 Creating a Custom Login HTML Page for an iOS-Powered Device

You can create the custom login page using the `adf.login.iphone.html` page, an artifact that is generated as part of deployment.

Before you begin:

Deploy an ADF Mobile application that uses the default login page.

To create a custom login page:

1. Copy the default login page, `adf.login.iphone.html` (located in *`application_folder`*`/deploy/`*`deployment profile name`*`/temporary_xcode_project/www/adf.login.iphone.html`) to a location within the application controller project's `public_html` directory, such as `JDeveloper\mywork\`*`Application Name`*`\ApplicationController\public_html`.
2. Rename the `adf.login.iphone.html` file.
3. In the Security page for the overview editor for the `adfmf-application.xml` file, select **Custom** and then click **Browse** to retrieve the location of the login page.

17.5 Adding Private Certificates

As described in Section 4.2.2.1, "About the Application-Level Resources," JDeveloper creates the `cacerts` certificate file within the Application Resources `Security` folder (located at `JDeveloper\mywork\Mobile Application\resources\Security\cacerts`). This file identifies a set of certificates from well-known and trusted sources to JVM 1.4 and enables deployment. For an application that requires custom certificates (such as in cases where RSA cryptography is not used), you must add private certificates before deploying the application.

Before you begin:

Refer to Java SE Technical Documentation (`http://download.oracle.com/javase/index.html`) for information on the `cacerts` file and how to use the keytool utility.

To add private certificates:

1. Create a private certificate. For example, create a certificate file called `new_cert`.
2. Add the private certificate to the application as follows:
   a. Create a copy of the seeded `cacerts` file (`cp cacerts cacerts.org`).

b. Use the Java SE keytool utility to add certificates to a `cacerts` file. Example 17–7 illustrates adding certificates to a `cacerts` file called `new_cert`.

Example 17–7 Adding a Certificate Using the keytool Utility

```
keytool -importcert
        -keystore cacerts
        -file new_cert
        -storepass changeit
        -noprompt
```

Example 17–7 illustrates how to add a single certificate. Repeat this procedure for each certificate. Table 17–2 lists the keytool options

Table 17–2 Options For Adding Certificates

Option	Description
`-importcert`	Imports a certificate.
`-keystore` *`cacerts file`*	Identifies the file location of the imported certificate.
`-file` *`certificate file`*	Identifies the file containing the new certificate.
`-storepass changeit`	Provides a password for the `cacerts` file. By default, the password is *changeit*.
`-noprompt`	Instructs the keytool not to ask the user (through `stdin`) whether to trust the certificate or not.

c. Visually inspect the contents of the new `cacerts` file to ensure that all of the fields are correct. Use the following command:

```
keytool -list -v -keystore cacerts | more
```

d. Verify that the certificate is for the given hostname.

Note: The certificate's common name (CN) must match the hostname exactly.

---

e. Ensure that the customized certificate file is located within the `Security` directory (`JDeveloper\mywork\`*`Mobile Application`*`\resources\Security`) so that it can be read by JVM 1.4.

3. Deploy the application.

Note: During deployment, if a certificate file exists within the `Security` directory, ADF Mobile copies it into the Android or Xcode template project, replacing any default copies of the `cacerts` file.

---

4. Validate that you can access the protected resources over SSL.

18

Testing and Debugging ADF Mobile Applications

This chapter provides information on testing and debugging ADF Mobile applications developed for both iOS and Android platforms.

This chapter includes the following sections:

18.1 Introduction to Testing and Debugging ADF Mobile Applications

Before you start any testing and debugging of your ADF Mobile application, you have to deploy it to one of the following:

- iOS-powered device
- iOS-powered device simulator
- Android-powered device
- Android-powered device emulator

You cannot run the ADF Mobile application until it is deployed. For more information, see Chapter 16, "Deploying ADF Mobile Applications."

To test and debug an ADF Mobile application, you generally take the following steps:

1. Test the application's infrastructure, such as a splash screen, application feature navigation, authentication, and preferences, ensuring that all declared application features are available.
2. If the application includes ADF Mobile AMX content, test this application feature's logic, page flows, data controls, and UI components.
3. Make changes to the application as necessary.
4. Reconnect the mobile device or restart the simulator, and then deploy and run the application for further testing and debugging.

For more information, see the following:

- Section 18.3, "Debugging ADF Mobile Applications"

- Section 18.2, "Testing ADF Mobile Applications"

18.2 Testing ADF Mobile Applications

There are two approaches to testing an ADF Mobile application:

1. Testing on a mobile device: this method always provides the most accurate behavior, and is also necessary to gauge the performance of your application. However, you may not have access to all the devices on which you wish to test, making device testing inconclusive.
2. Testing on a mobile device emulator or simulator: this method usually offers better performance and faster deployment, as well as convenience. However, even though a device emulator or simulator closely approximates the corresponding physical device, there might be differences in behavior and limitations on the capabilities that can be emulated.

Typically, a combination of both approaches yields the best results.

18.2.1 How to Perform Accessibility Testing on iOS-Powered Devices

You should use a combination of the following methods to test the accessibility of your ADF Mobile application developed for iOS-powered devices:

- Testing with the Accessibility Inspector on an iOS-powered device simulator.

  For detailed information, see the "Testing the Accessibility of Your iPhone Application" section in *Accessibility Programming Guide for iOS*, which is available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

- Testing with the VoiceOver on an iOS-powered device.

  For more information, see the "Using VoiceOver to Test Your Application" section in *Accessibility Programming Guide for iOS*, which is available through the iOS Developer Library (`http://developer.apple.com/library/ios/navigation/`).

18.3 Debugging ADF Mobile Applications

JDeveloper is equipped with debugging mechanisms that allow you to execute a Java program in debug mode and use standard breakpoints to monitor and control execution of an Oracle ADF application. For more information, see the section on debugging applications in *Oracle Fusion Middleware User's Guide for Oracle JDeveloper*.

Since an ADF Mobile application cannot be run inside JDeveloper, the debugging approach is different: you can use the JDeveloper debugger to connect to a Java Virtual Machine (JVM) 1.4 instance on a mobile device or simulator and control the Java portions of your deployed ADF Mobile application.

The following are the steps you need to take to use JDeveloper to debug the Java code in your ADF Mobile application:

1. Configure the project properties for remote debugging. This step involves creating an ADF Model debugging configuration as follows:
   a. From the JDeveloper's main menu, click **Application > Project Properties** to open the Project Properties dialog.
   b. In the Project Properties dialog, select the **Run/Debug/Profile** node.

c. Using the **Run Configurations** pane of the Run/Debug/Profile dialog, create a new configuration, or click **Edit** to modify an existing one.

d. In the **Edit Run Configuration > Launch Settings** dialog, select **Remote Debugging**.

e. Expand the Tools Settings node of the Edit Run Configuration dialog, then expand **Debugger**, and then select **Remote**.

f. In the **Edit Run Configuration > Remote** dialog that Figure 18–1 shows, perform the following configurations:

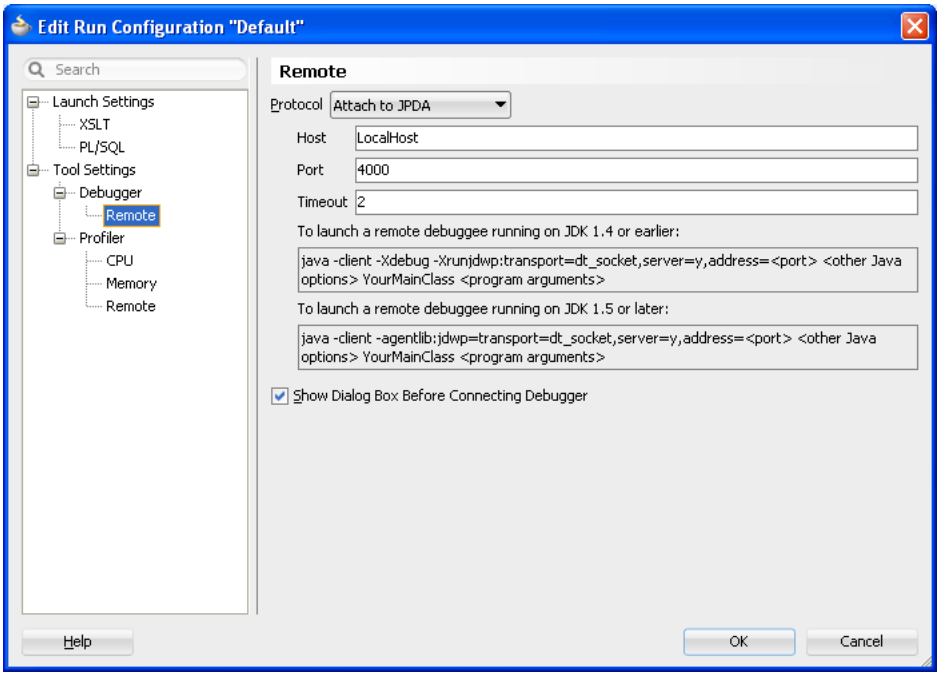
- Set the protocol to **Attach to JPDA**.

- Set the host to one of the following: 1) For simulator or emulator debugging, set to **localhost**; 2) For the device debugging, ensure that your development computer can access that device over the network (you may use the `ping` command to test network access), and then enter the device's IP address.

- Set the port to the appropriate port number.

- Set the timeout to **2**.

Figure 18–1 Configuring Remote Debugging



2. Deploy the application to a mobile device or simulator (see Section 18.3.1, "How to Debug on iOS Platform" or Section 18.3.2, "How to Debug on Android Platform").

3. Specify the following debugging parameters in the `cvm.properties` file:

```
java.debug.enabled=true
java.debug.port=4000
```

The port number must match the one you set in JDeveloper.

For more information, see Section 18.3.4, "How to Enable Debugging of Java Code and JavaScript."

4. Redeploy the application to the mobile device or simulator.

5. Launch the application in a mobile device or simulator by clicking the application icon.
6. Start the debugger from JDeveloper as follows:
    1. Use the debug icon on JDeveloper's main menu to select the run configuration (the one that you defined for your project in step 3).
    2. Confirm the debugger **Attach to JPDA Debugger** dialog.

> **Note:** To avoid timeout (20 seconds), start the debugger immediately after launching the application on the mobile device or simulator.

If you use the mobile device for debugging, you have to connect through WiFi.

For additional information, see the following:

- Section 16.1.1.2, "Deployment of the JVM 1.4 Libraries"
- "Testing and Debugging ADF Components" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

18.3.1 How to Debug on iOS Platform

To debug an ADF Mobile application on the iOS platform using JDeveloper, follow the generic debugging procedure described in Section 18.3, "Debugging ADF Mobile Applications."

For information on how to configure an iOS-powered device or simulator and how to deploy an ADF Mobile application for debugging, see the following:

- Section 16.4.1, "How to Deploy an iOS Application to an iOS Simulator"
- Section 16.4.2, "How to Deploy an Application to an iOS-Powered Device"
- Section 16.4.4.2, "Registering an Apple Device for Testing and Debugging"

18.3.2 How to Debug on Android Platform

To debug an ADF Mobile application on the Android platform using JDeveloper, follow the generic debugging procedure described in Section 18.3, "Debugging ADF Mobile Applications."

For information on how to configure an Android-powered device or emulator and how to deploy an ADF Mobile application for debugging, see Section 16.3.1, "How to Deploy an Android Application to an Android Emulator."

When you debug Java code, either on an Android-powered device connected through USB or on an Android-powered device emulator, you need to forward the TCP port by executing the following command on a terminal:

- For the device debugging:

```
adb -d forward tcp:<host port> tcp:<target port>
```

- For the emulator debugging:

```
adb -e forward tcp:<host port> tcp:<target port>
```

For example, executing `adb -d forward tcp:4510 tcp:4510` forwards the device TCP port 4510 to the host TCP port 4510. Upon execution, the debugging settings in the `cvm.properties` file (see Section 18.3.4, "How to Enable Debugging of Java Code and JavaScript") should be defined as follows:

```
java.debug.enabled=true
java.debug.port=4510
```

> **Note:** If the connection is made through Wi-Fi, ensure that this connection is correct. It is recommended to place both the debugger and target on the same network without the use of the virtual private network (VPN).

18.3.3 How to Debug the ADF Mobile AMX Content

If your ADF Mobile application includes the ADF Mobile AMX content, after you configure the device or emulator, you can set breakpoints, view the contents of variables, and inspect the method call stack just as you would when debugging a web-based ADF Faces application. For more information, see the "Testing and Debugging ADF Components" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

> **Note:** You can only debug your Java code and JavaScript (see Section 18.3.4, "How to Enable Debugging of Java Code and JavaScript"). Debugging of EL expressions or other declarative elements is not supported.

18.3.4 How to Enable Debugging of Java Code and JavaScript

A `cvm.properties` file allows you to specify startup parameters for the JVM and web views of ADF Mobile to enable debugging of the Java code and JavaScript. The `cvm.properties` file is automatically created and placed in your project's `Descriptors/META-INF` directory (see Section 18.5, "Using and Configuring Logging").

You can use the following debugging properties in the `cvm.properties` file:

- `java.debug.enabled`: Enables or disables Java debugging for ADF Mobile. Valid values are `true` and `false`.

  > **Caution:** When `java.debug.enabled` is set to `true`, the JVM waits for a debugger to establish a connection to it. Failure of the debugger to connect will result in the failure of the ADF Mobile AMX application feature to load.

- `java.debug.port`: Specifies the port to be used during debugging. The valid value is an integer.
- `javascript.debug.enabled`: Enables or disables JavaScript debugging when the application is running in the device simulator. Valid values are `true` and `false`.
- `javascript.debug.feature`: Specifies the application feature that is to trigger the activation of JavaScript debugging in ADF Mobile. The format of the value is `featureId:port` . The port must be specified (it is initially set to a placeholder value).

The contents of the `cvm.properties` file may be similar to the following:

```
java.debug.enabled=true
```

```
java.debug.port=8000
javascript.debug.enabled=true
javascript.debug.feature=products:8888
```

After the `cvm.properties` file has been configured to debug JavaScript, you can navigate to the following URL to see a listing of all the loaded pages that can be debugged in ADF Mobile:

```
http://localhost:9999
```

> **Caution:** Problems may arise if you debug JavaScript on a computer shared by multiple instances of an iOS-powered device simulator: if multiple instances of an iOS-powered device simulator are running, you might not be able to connect to your specific instance, which will prevent the debugging page from displaying.

For information on how to use JDeveloper to debug the Java code, see Section 18.3, "Debugging ADF Mobile Applications."

18.4 Using the Debug Mode

You use the application's deployment profile to specify either the release or debug execution mode for your ADF Mobile application. The debug mode allows for inclusion of special debugging libraries and symbols at compile time.

Figure 18–2 shows how to set the debug mode option on Android.

Figure 18–2 Setting Debug Mode for Android

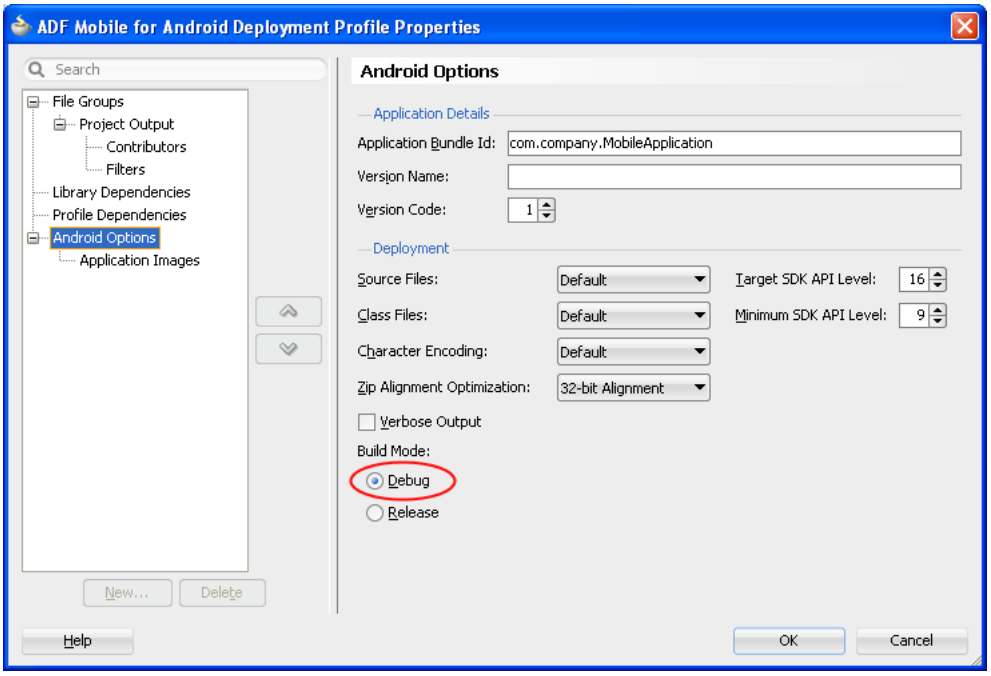
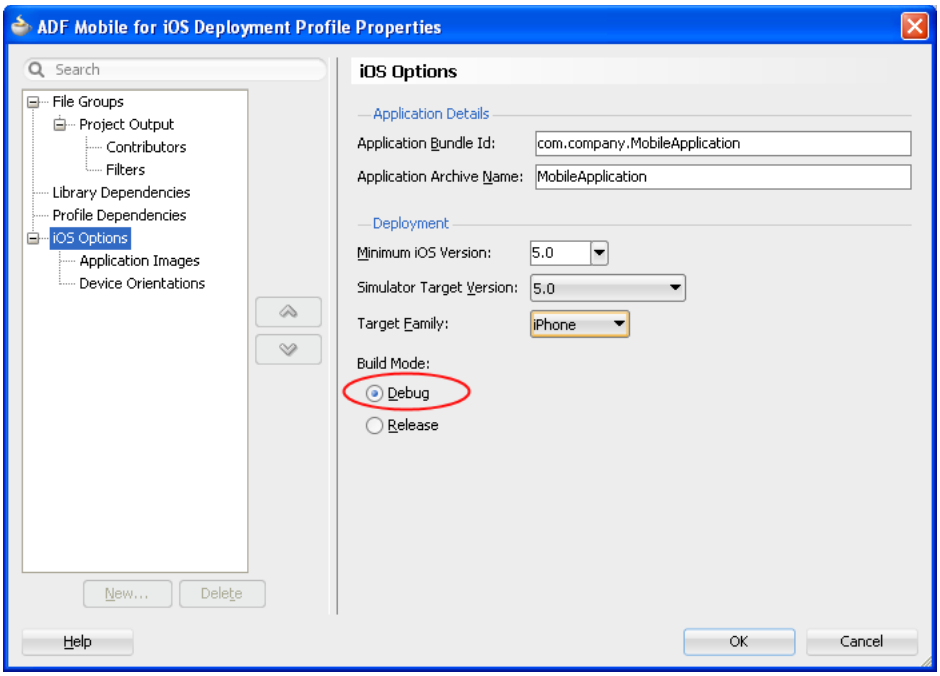


Figure 18–3 shows how to set the debug mode option on iOS.

Figure 18–3 Setting Debug Mode for iOS



For more information, see the following:

- Section 16.2.3, "How to Create an Android Deployment Profile"
- Section 16.2.4.1, "Defining the iOS Build Options"
- Section 16.4.4.1, "Creating iOS Development Certificates"

18.5 Using and Configuring Logging

For your ADF Mobile application, you can enable logging on Android and iOS platforms, through JavaScript and embedded code, all using a single configuration with the log output, including the output produced by `System.out.println` and `System.err.println` statements, directed to a single file.

The default ADF Mobile's logging process is as follows:

- The logging begins at application startup.
- The existing log file from the previous application run is deleted, so only the contents of the current run are available.
- When you are running your application on an iOS-powered device simulator, all logging output is sent to the console, which you can access through the `Application/Utilities` directory on your development computer.

  When you are running your application on an iOS-powered device, the console output is redirected to an `application.log` file that is placed in the `Documents/logs` directory of your application.

  On Android, the output is forwarded to a text file with the same name as the application. The output file location is `/sdcard`. If this location is not present or is configured as read-only, the log output is rerouted to the application's writable data directory.

- The `logging.properties` file is created in `/.adf/META-INF` directory and added to the application under the Application Resources (see Section 4.2.2.1, "About the Application-Level Resources"). In this file, it is defined that all loggers use the `com.sun.util.logging.ConsoleHandler` and `SimpleFormatter`, and the log level is set to `SEVERE`. You can edit this file to specify different logging behavior (see Section 18.5.1, "How to Configure Logging Using the Properties File").

> **Note:** In your ADF Mobile application, you cannot use loggers from the `java.util.logging` package.

ADF Mobile loggers are declared in the `oracle.adfmf.util.Utility` class as follows:

```
public static final String APP_LOGNAME = "oracle.adfmf.application";
public static final Logger ApplicationLogger = Logger.getLogger(APP_LOGNAME);

public static final String FRAMEWORK_LOGNAME = "oracle.adfmf.framework";
public static final Logger FrameworkLogger = Logger.getLogger(FRAMEWORK_LOGNAME);
```

The logger that you are to use in your ADF Mobile application is the `ApplicationLogger`.

You can also use methods of the `oracle.adfmf.util.logging.Trace` class.

For more information, see Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile.

18.5.1 How to Configure Logging Using the Properties File

Example 18–1 shows the `logging.properties` file that you use to configure logging.

Example 18–1 logging.properties File

```
# default - all loggers to use the ConsoleHandler
.handlers=com.sun.util.logging.ConsoleHandler
# default - all loggers to use the SimpleFormatter
.formatter=com.sun.util.logging.SimpleFormatter

oracle.adfmf.util.logging.ConsoleHandler.formatter=
             oracle.adfmf.util.logging.PatternFormatter
oracle.adfmf.util.logging.PatternFormatter.pattern=
             [%LEVEL%-%LOGGER%-%CLASS%-%METHOD%]%MESSAGE%

#configure the framework logger to only use the adfmf ConsoleHandler
oracle.adfmf.framework.useParentHandlers=false
oracle.adfmf.framework.handlers=oracle.adfmf.util.logging.ConsoleHandler
oracle.adfmf.framework.level=SEVERE

#configure the application logger to only use the adfmf ConsoleHandler
oracle.adfmf.application.useParentHandlers=false
oracle.adfmf.application.handlers=oracle.adfmf.util.logging.ConsoleHandler
oracle.adfmf.application.level=SEVERE
```

The `oracle.adfmf.util.logging.ConsoleHandler` plays the role of the receiver of the custom formatter.

The `oracle.adfmf.util.logging.PatternFormatter` allows the following advanced formatting tokens that enable log messages to be printed:

- `%LEVEL%`—the logging level.
- `%LOGGER%`—the name of the logger to which the output is being written.
- `%CLASS%`—the class that is being logged.
- `%METHOD%`—the method that is being logged.
- `%TIME%`—the time the logging message was sent.
- `%MESSAGE%`—the actual message.

The following logging levels are available:

- `SEVERE`: this is a message level indicating a serious failure.
- `WARNING`: this is a message level indicating a potential problem.
- `INFO`: this is a message level for informational messages.
- `FINE`: this is a message level providing tracing information.
- `FINER`: this level indicates a fairly detailed tracing message.
- `FINEST`: this level indicates a highly detailed tracing message.

> **Caution:** When selecting the amount of verbosity for a logging level, keep in mind that by increasing the verbosity of the output at the `SEVERE`, `WARNING`, and `INFO` level negatively affects performance of your application.

The logger defined in the `logging.properties` file matches the logger obtained from the `oracle.adfmf.util.Utility` class (see Section 18.5, "Using and Configuring Logging"). The logging levels also match. If you decide to use the logging level that is more fine-grained than `INFO`, you have to change the `ConsoleHandler`'s logging level to the same level, as Example 18–2 shows.

Example 18–2 Setting Very Fine-Grained Logging Level

```
oracle.adfmf.util.logging.ConsoleHandler.formatter=
             oracle.adfmf.util.logging.PatternFormatter
oracle.adfmf.util.logging.ConsoleHandler.level=FINEST
oracle.adfmf.util.logging.PatternFormatter.pattern=
             [%LEVEL%-%LOGGER%-%CLASS%-%METHOD%]%MESSAGE%
```

18.5.2 How to Use JavaScript Logging

JavaScript writes the output to the `console.log` or `.error/.warn/.info`. This output is redirected into the file through the `System.out` utility.

You customize the log output by supplying a message. The following JavaScript code produces `"Message from JavaScript"` output:

```
<script type="text/javascript" charset="utf-8">
   function test_function() { console.log("Message from JavaScript"); }
</script>
```

To make use of the properties defined in the logging file, you need to use the `adf.mf.log` package and the `Application` logger that it provides.

The following logging levels are available:

- `adf.mf.log.level.SEVERE`
- `adf.mf.log.level.WARNING`
- `adf.mf.log.level.INFO`
- `adf.mf.log.level.CONFIG`
- `adf.mf.log.level.FINE`
- `adf.mf.log.level.FINER`
- `adf.mf.log.level.FINEST`

To trigger logging, use the `adf.mf.log.Application` logger's `logp` method and specify the following through the method's parameters:

- the logging level
- the current class name as a String
- the current method as a String
- the message string as a String

Example 18–3 shows how to use the `logp` method in an ADF Mobile application.

Example 18–3 Using Logging Method

```
adf.mf.log.Application.logp(adf.mf.log.level.WARNING,
                            "myClass",
                            "myMethod",
                            "My Message");
```

Upon execution of the `logp` method, the following output is produced:

```
[WARNING - oracle.adfmf.application - myClass - myMethod] My Message
```

18.5.3 How to Use Embedded Logging

Embedded logging uses the `com.sun.util.logging.Logger`, as illustrated in Example 18–4.

Example 18–4 Using Embedded Logging

```
import com.sun.util.logging.Level;
import com.sun.util.logging.Logger;
import oracle.adfmf.util.logging.*;
...
   Utility.ApplicationLogger.logp(Level.WARNING,
                                  EmbeddedClass.class.getName(),
                                  "onTestMessage",
                                  "embedded warning message 1");
   Logger.getLogger(Utility.APPLICATION_LOGNAME).logp(Level.WARNING,
                                                      this.getClass().getName(),
                                                      "onTestMessage",
                                                      "embedded warning message 2");
   Logger.getLogger("oracle.adfmf.application").logp(Level.WARNING,
                                                     this.getClass().getName(),
                                                     "onTestMessage",
                                                     "embedded warning message 3");
```

The preceding code produces the following output:

```
[WARNING - oracle.adfmf.application - EmbeddedClass - onTestMessage] embedded warning message 1
[WARNING - oracle.adfmf.application - EmbeddedClass - onTestMessage] embedded warning message 2
[WARNING - oracle.adfmf.application - EmbeddedClass - onTestMessage] embedded warning message 3
```

Part VII

Appendixes

Part VII contains the following appendixes:

- Appendix A, "Troubleshooting"
- Appendix B, "Converting Preferences for Deployment"
- Appendix C, "ADF Mobile Application Usage"
- Appendix D, "Parsing XML"
- Appendix E, "ADF Mobile Sample Applications"

A

Troubleshooting

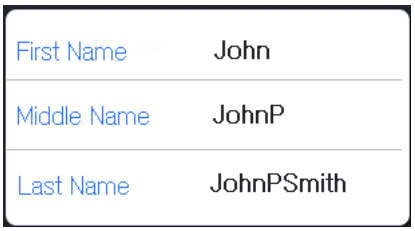
This appendix describes problems with various aspects of ADF Mobile applications, as well as how to diagnose and resolve them.

A.1 Problems with Input Components on iOS Simulators

Issue:

On ADF Mobile applications deployed to iOS simulators, text entered into one `<amx:inputText>` component field becomes attached to the beginning of the text entered in subsequent field when navigating from one field to another using a mouse. For example, on a page with First Name, Middle Name, and Last Name input text fields, if you enter *John* in the First Name field, then click the Middle Name field, and enter P, the text displays as *JohnP*. Likewise, when you click the Last Name field, and enter Smith, the text in that field displays as *JohnPSmith*, as shown in Figure A–1.

Figure A–1 Text Values Concatenate in Subsequent <amx:inputText> fields



Note: This behavior only occurs on iOS simulators and in web pages, not on actual devices.

Solution:

Use the keyboard on the simulator to traverse the input text fields rather than the mouse.

B

Converting Preferences for Deployment

This appendix describes how ADF Mobile converts user preferences during deployment.

This document includes the following sections:

- Section B.1, "Naming Patterns for Preferences"
- Section B.2, "Converting Preferences for Android"
- Section B.3, "Converting Preferences for iOS"

B.1 Naming Patterns for Preferences

Conversion of ADF Mobile application preferences to a mobile-platform representation occurs when a deployment target is invoked. Following conversion, the naming pattern described in Table B–1 ensures that each preference can be uniquely identified on the mobile platform. Each preference element in the `adfmf-application.xml` and `adfmf-feature.xml` files must be uniquely identified within the scope of its sibling elements prior to deployment.

The following are examples of identifier values:

- `application.gen.gps.trackGPS`
- `feature.f0.gen.gps.trackGPS`

Table B–1 describes how to generate fully qualified preference identifiers.

Table B–1 ADF Mobile Naming Patterns for Preferences

Expression	Description	Syntax
`PreferenceIdentifier`	Represents an identifier value of a preference element that has been converted to a mobile platform representation.	`ApplicationPreferences | FeaturePreferences`

Table B–1 (Cont.) ADF Mobile Naming Patterns for Preferences

Expression	Description	Syntax

`ApplicationPreferences`	Use this expression to build a preference identifier value that is generated from the `adfmf-application.xml` file.	*`application`*`.ApplicationElementPath`
		`ApplicationElementPath` represents a dot-separated list of `id` attribute values beginning with the top-most parent element, `<adfmf:preferences>`, and ending with the element that is to be identified. In the following segment from the `adfmf-application.xml` file, this generated identifier is shown in the comment as `application.gen.gps.trackGPS`.
		`<adfmf:preferences>`
		`  <adfmf:preferenceGroup id="gen">`
		`  <adfmf:preferenceGroup id="gps">`
		`  <!-- The mobile-platform identifier would be "application.gen.gps.trackGPS" -->`
		`  <adfmf:preferenceBoolean id="trackGPS"/>`
		`  </adfmf:preferenceGroup>`
		`  </adfmf:preferenceGroup>`
		`</adfmf:preferences>`

`FeaturePreferences`	Use this expression to build a preference identifier value that is generated from the `adfmf-feature.xml` file.	*`feature`*`.FeatureElementPath`
		`FeatureElementPath` represents a dot-separated list of `id` attribute values beginning with `<adfmf:feature>`, the top-most parent element, and ending with the element that is to be identified. In the following segment from the `adfmf-feature.xml` file, this generated identifier is displayed in the comment as `feature.f0.gen.gps.trackGPS`.
		`<adfmf:feature id="f0">`
		`  <adfmf:preferences>`
		`  <adfmf:preferenceGroup id="gen">`
		`  <adfmf:preferenceGroup id="gps">`
		`  <!-- The mobile-platform identifier would be "feature.f0.gen.gps.trackGPS" -->`
		`  <adfmf:preferenceBoolean id="trackGPS"/>`
		`  </adfmf:preferenceGroup>`
		`  </adfmf:preferenceGroup>`
		`  </adfmf:preferences>`
		`</adfmf:feature>`

The `<adfmf:preferences>` element cited in the code examples in Table B–1 does not have an `id` attribute and is therefore not represented in any preference identifiers.

B.2 Converting Preferences for Android

The ADF Mobile deployment uses XML and XLS to transform the user preference pages defined at both the application feature and ADF Mobile application-level into the following three XML documents:

- `preferences.xml`
- `arrays.xml`
- `strings.xml`

B.2.1 Preferences.xml

This file contains the transformed preferences from both of the `adfmf-feature.xml` and `adfmf-application.xml` files.

B.2.1.1 Preferences Element Mapping

Table B–2 shows the mapping of ADF Mobile's preference definitions to Android template preferences, and Android native preferences:

Table B–2 Mapping ADF Mobile Preferences to Android Preferences

ADF Mobile Preference Definition	Template Custom or Android Native Preference Definition (Used by Deployment)	Android Native Preference Definition (Not used by Deployment)
`<adfmf:preferenceBoolean>`	`oracle.adfmf.preferences.AdfMFPreferenceBoolean`	`CheckBoxPreference`
`<adfmf:preferenceNumber>`	`oracle.adfmf.preferences.AdfMFPreferenceText`	`EditPreferenceText`
`<adfmf:preferenceText>`	`oracle.adfmf.preferences.AdfMFPreferenceText`	`EditTextPreference`
`<adfdmf:preferenceList>`	`oracle.adfmf.preferences.AdfMFPreferenceList`	`ListPreference`
`<adfmf:PreferenceGroup>`	`PreferenceCategory`	`PreferenceCategory`
`<adfmf:PreferencePage>`	`PreferenceScreen`	`PreferenceScreen`

B.2.1.2 Preference Attribute Mapping

The `Preferences.xml` file contains references to string resources contained in both `strings.xml` and `arrays.xml`. The Android SDK defines the syntax for resources in XML files as *@[<package_name>:]<resource_type>/<resource_name>*. This file contains references to string values as well as the name and value pairs of list preferences. The XSL constructs the following for the strings and list preferences:

- `<package_name>` is the name of the package in which the resource is located (not required when referencing resources from the same package). This component of the reference will not be used.
- `<resource_type>` is the R subclass for the resource type. This component will have a value of `string` if constructing a string reference or `array` if constructing a list preference.
- `<resource_name>` is the `android:name` attribute value in the XML element. The value for this component will be the value of the `<PreferenceIdentifier>_title` when specifying the `android:title` attribute (see Section B.1, "Naming Patterns for Preferences." for the definition of `<PreferenceIdentifier>`.

Table B–3 and Table B–4 show the mapping of ADF Mobile attributes for a given ADF Mobile preference to the Android preference.

In this table:

- Entries of the form {*X*} (such as `{default}` in Table B–3) indicate the value of an ADF Mobile attribute named *X*.
- Entries having `<PreferenceIdentifier>` indicate the value of the preference identifier, as defined in Section B.1, "Naming Patterns for Preferences."
- Attributes with an asterisk (*) are custom template attributes defined in an ADF Mobile namespace and must appear in the `preferences.xml` in the form `adfmf:<attributeName>`. Otherwise, the attributes are part of the Android

namespace and must appear in the `preferences.xml` as `android:<attributeName>`.

Table B–3 Mapping of ADF Mobile Preference Attributes to Android Preferences

ADF Mobile Attribute Definition	Template Custom or Android Native Preference Attribute	Android Attribute Value	Applies to
`id`	`key`	`<PreferenceIdentifier>`	`AdfMFPreferenceBoolean`, `AdfMFPreferenceText`, `AdfMFPreferenceList`, `PreferenceScreen`, `PreferenceCategory`
`default`	`defaultValue`	`{default}`	`AdfMFPreferenceBoolean`, `AdfMFPreferenceText`, `AdfMFPreferenceList`
`label`	`title`	`@string/<PreferenceIdentifier>___` `title` if the given `{label}` value is not a reference to a string resource bundle. References a string in `strings.xml` having the given `{label}`.	`AdfMFPreferenceBooleanAdfM`, `FPreferenceNumber`, `AdfMFPreferenceText`, `AdfMFPreferenceList`, `PreferenceScreen`, `PreferenceCategory`
`secret`	`password`	`{secret}`	`AdfMFPreferenceText`
`min`	`min*`	`{min}`	`AdfMFPreferenceText`
`max`	`max*`	`{max}`	`AdfMFPreferenceText`
`name`	`entryValues`	`@array/<PreferenceIdentifier>___` `entryValues`	`AdfMFPreferenceList`
`value`	`entries`	`@array/<PreferenceIdentifier>___` `entries`	`AdfMFPreferenceList`

B.2.1.3 Attribute Default Values

The overview editors for the `adfmf-application.xml` and `adfmf-feature.xml` files exclude an attribute name and value from the XML if:

- The attribute type is `xsd:boolean`.
- The attribute value has a `<default>` value option.
- The user specifies `<default>` as the value.

The XSL must know the ADF Mobile attributes that are boolean typed and their corresponding default values. The XSL, then, specifies the appropriate Android or template custom attribute value where has been selected by the user.

Table B–4 indicates what the deployment will specify for the `android:defaultValue` attribute if the ADF Mobile preference being transformed does not contain a `default` attribute:

Table B–4 Transforming Attributes with Non-Default Values

ADF Mobile Preference Element	Android Preference Equivalent	Default Attribute Value
`preferenceBoolean`	`AdfMFPreferenceBoolean`	`false`
`preferenceText`	`AdfMFPreferenceText`	Empty string

Table B–4 (Cont.) Transforming Attributes with Non-Default Values

ADF Mobile Preference Element	Android Preference Equivalent	Default Attribute Value
preferenceList	AdfMFPreferenceList	Empty string

B.2.1.4 Preferences Screen Root Element

The preferences.xml has a root element called <PreferenceScreen> element. The Android template requires that this element have the following XML namespace definition:

```
xmlns:adfmf="http://schemas.android.com/apk/res/<Application Package Name>
```

where <Application Package Name> is the same application package name put in the AndroidManifest.xml file. <Android Package Name> defines the definition for the Android package name specified in the AndroidManifest.xml. For more information, see Section 5.3.1, "How to Set the ID and Display Behavior for a Mobile Application."

The deployment uses the Application Bundle Id value from the Android deployment profile, if it exists. If it does not exist in the profile, the deployment obtains this value from the application display name and Application Id contained in the adfmf-application.xml file. The deployment Java code will pass the value to the XSL document as a parameter.

Example B–1 shows ADF Mobile preferences contained in the adfmf-feature.xml file. Example B–1 shows the corresponding preferences.xml file.

Example B–1 Preferences Defined in the adfmf-feature.xml File

```
<?xml version="1.0" encoding="UTF-8" ?>
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance
xmlns:adfmf="http://xmlns.oracle.com/jdev/adfmf">
   <adfmf:feature id="oracle.hello" name="Hello"
                  icon="oracle.hello/navbar-icon.png"
                  image="oracle.hello/springboard-icon.png">
      <adfmf:content id="Hello.Generic">
         <adfmf:localHTML url="oracle.hello/index.html"/>
      </adfmf:content>
      <adfmf:preferences>
         <adfmf:preferenceGroup id="prefGroup"
                                label="preference group">
            <adfmf:preferenceBoolean id="boolPref"
                                     label="boolPref perference"
                                     default="true"/>
            <adfmf:preferenceNumber id="numPref"
                                    label="numPref preference"
                                    default="1"
                                    min="1"
                                    max="10"/>
            <adfmf:preferenceText id="textPref"
                                  label="textPref preferences"
                                  default="Foo"/>
            <adfmf:preferenceList id="listPref"
                                  label="listPref preference"
                                  default="value2">
               <adfmf:preferenceValue name="name1"
                                      value="value1"/>
               <adfmf:preferenceValue name="name2"
```

```
                                        value="value2"/>
            </adfmf:preferenceList>
         </adfmf:preferenceGroup>
      </adfmf:preferences>
   </adfmf:feature>
</adfmf:features>
```

B.2.2 arrays.xml

This file consists of string-array elements that enumerate the names and values of list preferences that are referenced from the `preferences.xml`file. Each `<preferenceList>` contained in the `adfmf-application.xml` and `adfmf-feature.xml` files will be transformed into two string-array elements, one for the name and one for the values. For example, an ADF Mobile `preferenceList` definition described in Example B–2 results in `<string-array name="feature.oracle.hello.prefGroup.MyList___entry_values">` and `<string-array name="feature.oracle.hello.prefGroup.MyList___entries">` in the `arrays.xml` file shown in Example B–3

Example B–2 PreferenceList Definition in the adfmf-feature.xml File

```
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:adfmf="http://xmlns.oracle.com/jdev/adfmf">

<adfmf:feature id="oracle.hello" name="Hello" icon="oracle.hello/navbar-icon.png"
image="oracle.hello/springboard-icon.png">
...
   <adfmf:preferences>
      <adfmf:preferenceGroup id="prefGroup">
         <adfmf:preferenceList id="MyList" label="My List">
            <adfmf:preferenceValue name="name1" value="value1"/>
            <adfmf:preferenceValue name="name2" value="value2"/>
            <adfmf:preferenceValue name="name3" value="value3"/>
         </adfmf:preferenceList>
      </adfmf:preferenceGroup>
   </adfmf:preferences>
</adfmf:feature>
...
```

Example B–3 Preference Lists Converted to <string-array> Elements in arrays.xml

```
<resources xmlns:android="http://schemas.android.com/apk/res/android"
           xmlns:adfmf="http://schemas.android.com/apk/res/oracle.myandroidapp">

   <string-array name="feature_oracle_hello_prefGroup.MyList___entry_values">
      <item>name1</item>
      <item>name2</item>
      <item>name3</item>
   </string-array>

   <string-array name="feature_oracle_hello_prefGroup.MyList___entries">
      <item>value1</item>
      <item>value2</item>
      <item>value3</item>
   </string-array>
</resources>
```

Example B–4 shows the `<string-arrays>` referenced in `preferences.xml`.

Example B–4 PreferenceList Reference from the preferences.xml File

```
<oracle.adfmf.preferences.AdfMFPreferenceList android:key="feature.oracle.hello.MyList"
android:title="@string/feature_oracle_hello_prefGroup.MyList___title"
android:entries="@array/feature_oracle_hello_prefGroup.MyList___entries"
android:entryValues="@array/feature_oracle_hello_prefGroup.MyList___entry_values" />
```

B.2.3 Strings.xml

The `strings.xml` file, shown in Example B–5, consists of string elements that are referenced by the `preferences.xml` file, as well as any resource bundle references defined in the `adfmf-application.xml` and `adfmf-feature.xml` files. Each string element has a `name` attribute that uniquely identifies the string and the string value.

Example B–5 The strings.xml File

```
<resources xmlns:android="http://schemas.android.com/apk/res/android"
           xmlns:adfmf="http://schemas.android.com/apk/res/oracle.myandroidapp">
...
  <string name="feature.PROD.bundle.FeatureName">Products</string>
  <string name="feature.oracle.hello.prefGroup.MyBooleanPreference___title">My
feature boolean pref</string>
...
</resources>
```

If the source of the string is not a reference to a resource bundle string, the naming convention for the `name` attribute is `<PreferenceIdentifier>___<androidAttributeName>`.

Example B–6 Resource Bundle References Defined in the adfmf-feature.xml File

```
<adfmf:features xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
                xmlns:adfmf="http://xmlns.oracle.com/jdev/adfmf">
   <adfmf:loadBundle basename="mobile.ViewControllerBundle" var="bundle"/>
   <adfmf:feature id="oracle.hello" name="Hello"
                                    icon="oracle.hello/navbar-icon.png"
                                    image="oracle.hello/springboard-icon.png">
   <adfmf:feature id="PROD"
                  name="#{bundle.FeatureName}"
                  icon="openMore.png"
                  image="G.png"
                  credentials="none">
...
      <adfmf:preferences>
         <adfmf:preferenceGroup id="prefGroup">
            <adfmf:preferenceBoolean default="true" id="MyBooleanPreference"
label="My feature boolean pref"/>
         </adfmf:preferenceGroup>
      </adfmf:preferences>
   </adfmf:feature>
```

B.3 Converting Preferences for iOS

The iOS deployer transforms the ADF Mobile preferences listed in Table B–3 to the preference list (`.plist`) file representation required by an iOS Settings application.

Table B–5 ADF Mobile Preferences and Their iOS Counterparts

ADF Mobile Preferences Component	iOS Representation
`<adfmf:preferencePage>`	`PSChildPaneSpecifier`
`<adfmf:preferenceGroup>`	`PSGroupSpecifier`
`<adfmf:preferenceBoolean>`	`PSToggleSwitchSpecifier`
`<adfmf:preferenceList>`	`PSMultiValueSpecifier`
`<adfmf:preferenceText>`	`PSTextFieldSpecifier`
`<adfmf:preferenceNumber>`	`PSTextFieldSpecifier`

For information on the iOS requirement for preference list (`.plist`) files, see the "Implementing Application Preferences" section in *iOS Application Programming Guide*, which is available through the iOS Developer Library (http://developer.apple.com/library/ios/navigation/).

Example B–7 shows an ADF Mobile preferences stanza.

Example B–7 XML Based on the adfmf-application.xml File

```
<adfmf:preferences>
     <adfmf:preferenceGroup id="gen" label="Oracle Way Cool Mobile App">
         <adfmf:preferenceGroup id="SubPage01" label="Child Page">
     </adfmf:preferenceGroup>
</adfmf:preferences>
```

C

ADF Mobile Application Usage

This appendix provides an introductory information on the ADF Mobile user experience.

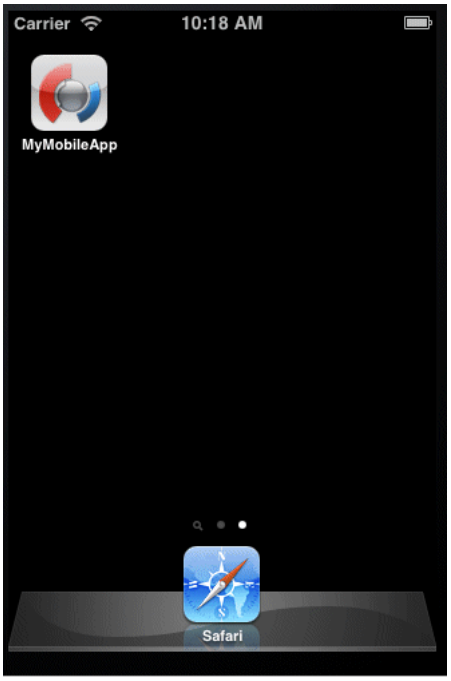
This appendix includes the following sections:

- Section C.1, "Introduction to ADF Mobile Application Usage"
- Section C.2, "Installing the ADF Mobile Application on a Mobile Device"
- Section C.3, "End User Navigation Between Application Features"
- Section C.4, "Setting Preferences"
- Section C.5, "Limitations to the Application Usage"

C.1 Introduction to ADF Mobile Application Usage

After installing an ADF Mobile application (see Section C.2, "Installing the ADF Mobile Application on a Mobile Device"), the end user can start using it by selecting the application icon on their mobile device's home screen (see Figure C–1), which displays the splash screen while the application launches. After the completion of the launch, the end user can navigate between application features (see Section C.3, "End User Navigation Between Application Features"), set preferences (see Section C.4, "Setting Preferences"), and perform all other tasks.

Figure C–1 Application Icon on iPhone



C.2 Installing the ADF Mobile Application on a Mobile Device

The end user can download and install an ADF Mobile application through their regular application provisioning mechanism.

During installation, the application's Preferences are populated with default settings. For information on how to modify the defaults, see Section C.4, "Setting Preferences" and Section C.2.2, "How End Users Install ADF Mobile Applications on Android-Powered Devices".

Removing the ADF Mobile application from a mobile device is not different from uninstalling any other application (see Section C.2.3, "How End Users Uninstall an ADF Mobile Application").

C.2.1 How End Users Install ADF Mobile Applications on iOS-Powered Devices

Users of iOS-powered devices download and install ADF Mobile applications as follows:

- From an enterprise-specific distribution mechanism:
  - using iTunes;
  - deploying through iPhone Configuration Utility;
  - wirelessly, hosted on a web server.
- From Apple's App Store.

C.2.2 How End Users Install ADF Mobile Applications on Android-Powered Devices

In addition to installing ADF Mobile applications available through the application marketplace, the end user can download applications available outside of the application marketplace. It is recommended to search the web for information on how to do this.

C.2.3 How End Users Uninstall an ADF Mobile Application

An ADF Mobile application is removed from the mobile device just like any other application. During the uninstall process, all application data and all external preferences are removed along with the application.

C.3 End User Navigation Between Application Features

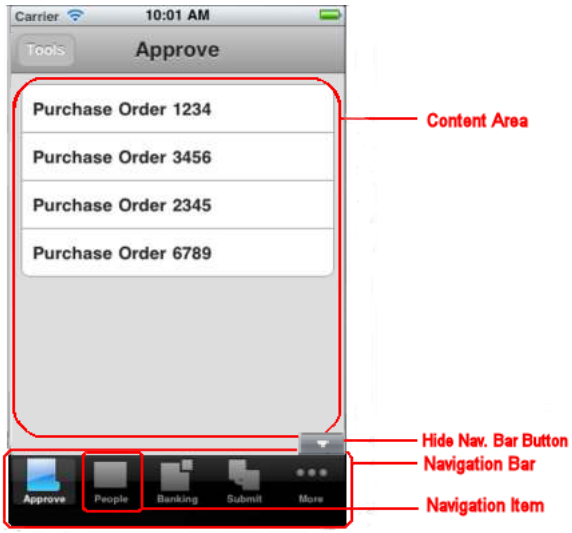
To provide access to each application feature, ADF Mobile applications allow for navigation between enabled application features using either a navigation bar or a springboard.

For information on configuring navigation during the application development, see Section 5.4, "Configuring the Springboard and Navigation Bar Behavior."

C.3.1 How End Users Navigate Between Application Features on iOS-Powered Devices

Figure C–2 shows elements of the ADF Mobile UI displayed on an iPhone.

Figure C–2 UI Elements on iPhone



The UI consists of a navigation bar populated with navigation items (icons). The first navigation item is highlighted to indicate that it is selected.

Note: If the springboard is defined for the application, a Home navigation button represented by an overlay is rendered above the navigation bar, but is not a part of it. This button allows the end user to return to the springboard from the application content:

If the springboard is not specified for the application, the Home icon is not displayed.

For more information, see Section 5.4, "Configuring the Springboard and Navigation Bar Behavior."

The navigation bar in the example that Figure C–2 shows contains six navigation items, and since not all of them can be displayed at the same time due to the space limitations on an iPhone, the fifth icon is represented by the **More** feature. When activated the More feature expands the navigation bar into the mode that lists the remaining navigation items. On an iPad, all navigation items are displayed. The content area above the navigation bar provides the content specific to this particular solution, which is an approval tool for purchase orders and is bundled with the application.

Note: If at least one icon for an application feature is shown on the navigation bar, the end user is presented with Hide and Show buttons that allow to display the navigation bar when it is hidden, and hide when it is shown:

If the application XML file (`adfmf-application.xml`) only defines a single application feature, or if the constraints (or conditions) allow for only a single application feature to be displayed, then the navigation bar is hidden.

The Hide and Show buttons may not be presented to the end user and the navigation bar could be initially hidden if the `adfmf-application.xml` file does not reference any application features to be displayed on the navigation bar. In this case, if the springboard is defined, it will be the only navigation tool for the application.

For more information, see Section C.3.1.2, "What You May Need to Know About Single-Featured Applications."

Figure C–3 shows the iPhone UI after the activation of the More icon and display of the remaining navigation items as a list.

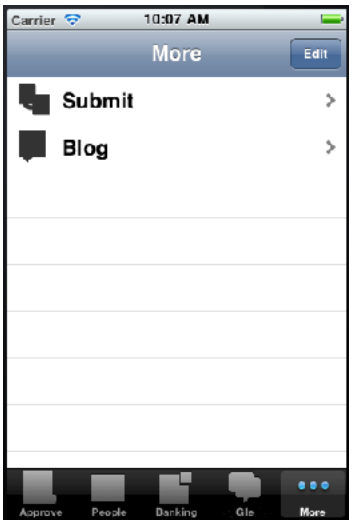
Figure C–3 Display of All Navigation Items on iPhone

To change which navigation items appear on the navigation bar at the application startup, the end user can select **Edit** to enter the **Configure** mode, as Figure C–4 shows.

Figure C–4 "Configure" Mode on iPhone

The Configure mode allows for dragging icons from the content area and dropping them onto the navigation bar. In this example, Submit navigation item was replaced with Gle navigation item on the navigation bar, and Submit is listed under More items, as Figure C–5 shows. To exit the configuration mode, the user selects **Done** to return to the More screen.

Figure C–5 Changing Navigation Bar Display on iPhone



If the end user selects the newly repositioned **Gle** navigation item in the navigation bar, Gle page is displayed in the **content area**, as Figure C–6 shows.

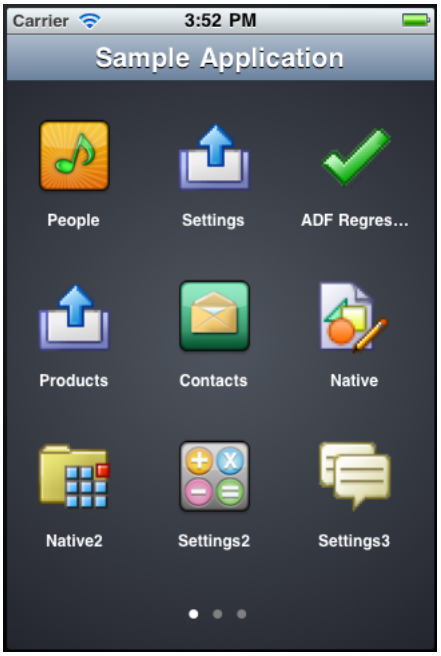
Figure C–6 Activation of Repositioned Navigation Item on iPhone

C.3.1.1 What You May Need to Know About Navigation Using the Springboard

By default, the springboard navigation is disabled in ADF Mobile. It is enabled during development by configuring the `adfmf-application.xml` file (see Section 5.4, "Configuring the Springboard and Navigation Bar Behavior").

If an ADF Mobile application is enabled for navigation using the springboard, the end user is presented a display similar to the one shown in Figure C–7 when the application starts.

Figure C–7 Springboard Display



In the preceding illustration, the default springboard supplied by ADF Mobile is displayed on an iPhone in the default portrait layout (for information on how to create a custom springboard during development, see Section 5.4, "Configuring the Springboard and Navigation Bar Behavior"). There are three pages of the application

content features that are available to the end user, which is indicated by the three dots at the bottom of the screen. The end user is on page one of the three pages, which is denoted by the bright dot in the first position.

To open the second page of features (see Figure C–8), the end user swipes the iPhone screen from right to left, pushing the first page to the left and bringing the second page from the right. As the first page moves left, it fades out, and as the second page moves in, it fades in.

Figure C–8 Springboard - Second Page

In the preceding illustration, the end user is on page two of the three pages, which is denoted by the bright dot in the second position.

When the end user rotates the mobile device to landscape orientation, the springboard icons animate into positions that will better accommodate such change (see Figure C–9).

Figure C–9 Springboard Display in Landscape

Note that the page did not change when the display orientation changed, and the end user is still on page two.

To return to the first page of features while the display is in landscape orientation (see Figure C–10), the end user swipes the iPhone screen from left to right, pushing the second page to the right and bringing the first page from the left.

Figure C–10 Springboard in Landscape - First Page

To view a particular feature (such as Contacts) from page one, the end user touches the Contacts icon or its corresponding text.

On iOS-powered devices, the end user can return to the springboard from any application feature by performing a device shake gesture.

On Android-powered devices, the end user can use a menu item that lets them return to the springboard at any time.

C.3.1.2 What You May Need to Know About Single-Featured Applications

Some applications may have only a single feature, and this feature is not configured to be displayed on a Springboard or navigation bar. In this case, only this single feature is presented to the end user; the special buttons that control the display of the navigation bar or return to the Springboard are not visible.

C.3.2 How to Navigate on Android-Powered Devices

Feature navigation on Android-powered devices is almost identical to the navigation on iOS-powered devices (see Section C.3.1, "How End Users Navigate Between Application Features on iOS-Powered Devices"), with the exception of the More feature: on Android-powered devices, the More feature, when activated, triggers the display of a list of the remaining navigation items. The navigation bar does not change its appearance.

C.4 Setting Preferences

The end user can configure the application preferences in the manner already prescribed by the mobile platform.

For information on configuring preferences during the application development, see Chapter 13, "Enabling User Preferences."

C.4.1 How to Set Preferences on iOS-Powered Devices

The end user can open the Settings application on their iOS-powered device and select ADF Mobile application's Settings icon to access all the settings available for that application. The modified settings take effect upon exiting the Settings application. This is a typical behavior of all applications on iOS-powered devices.

Preferences are populated with default values at startup. These values are defined in the `adf-feature.xml` file. In addition to the standard ways of setting Preferences values, they can be defined as follows:

- By making selection from a list of values.
- As non-readable values (for entering passwords and such).
- As binary values.

Preferences are displayed on cascading pages. Modifiable preferences can be easily distinguished from the ones that cannot be modified.

Preferences can be used to globally set the user credentials (see Chapter 17, "ADF Mobile Application Security").

C.4.2 How to Set Preferences on Android-Powered Devices

Setting Preferences on Android-powered devices does not differ from the same operation on iOS-powered devices (see Section C.4.1, "How to Set Preferences on iOS-Powered Devices"): the Preferences are accessed through the Preferences menu item.

Note: The Preferences menu item does not appear in the menu if there are no preferences defined for the application.

C.5 Limitations to the Application Usage

There is a number of limitations to the usage of various modules of typical ADF Mobile applications.

C.5.1 List View Component Limitations

The following are limitations of which the end user should be aware when using an ADF Mobile AMX List View component (see Section 8.2.7, "How to Use List View and List Item Components"):

1. If a List View component is in edit mode, the end user is only allowed to reorder rows (represented by List Item components) and cannot select or highlight a row.
2. If a List View component is in edit mode and dividers are enabled for the component, the end user should not use them, as this may cause unexpected results.

C.5.2 Data Visualization Components Limitations

With the exception of the geographic map (see Section 8.5.14, "How to Create a Geographic Map Component"), ADF Mobile AMX data visualization components do not support interactivity on Android 2.*n* platform.

C.5.3 Device Back Button Limitations on Android Platform

On Android 4.0.3, the device back button is disabled when an ADF Mobile application is running.

D

Parsing XML

This appendix contains information about libraries that can be used to parse XML.

This appendix includes the following section:

- Section D.1, "Parsing XML Using kXML Library"

D.1 Parsing XML Using kXML Library

kXML, one of the core ADF Mobile libraries, provides API that you can use to parse XML. This library is exposed to the application through the JDK Profiler Interface (JVMPI).

For more information, consult kXML documentation at:

- `http://kxml.sourceforge.net/kxml2`
- `http://kxml.sourceforge.net/kxml2/javadoc`

E

ADF Mobile Sample Applications

This appendix describes the ADF Mobile sample applications.

E.1 Overview of the ADF Mobile Sample Applications

ADF Mobile ships with a set of a sample applications that provide different development scenarios, such as creating the basic artifacts, accessing such device-native features as SMS and e-mail, or performing CRUD (Create, Read, Update, and Delete) operations on a local SQLite database. These applications are in the `PublicSamples.zip` file at the following location within the JDeveloper installation directory of your development computer:

jdev_install`/jdeveloper/jdev/extensions/oracle.adf.mobile/Samples`

To view these applications, extract the `PublicSamples.zip` file to your JDeveloper working directory (typically, this is *User Home Directory*`/jdeveloper/mywork`).

Note: Sample applications that include a default springboard must be extracted to, and opened from, the `Samples` directory. The `adfmf-application.xml` file for these applications must also be updated, as described in Section 5.4.1, "How to Configure Application Navigation," to ensure that the springboard is added to the classpath.

To enable sample applications that use the default springboard run properly:

1. Open the application from the `Samples` directory.
2. In the Applications page of the overview editor for the `adfmf-application.xml` file, change the springboard option from **Default** to **None**.
3. Choose **Default** as the springboard option.
4. Click **Save All**.
5. Deploy the application.

These applications, which are described in Table E–1, are complete. Except where noted otherwise, these applications can be deployed to a simulator after you configure the development environment as described in Chapter 3, "Setting Up the ADF Mobile Environment."

Tip: To get an idea of how to create an ADF Mobile application, review these applications in the order set forth in Table E–1.

Table E–1 ADF Mobile Sample Applications

Recommended Order of Use	Application Name	Description	Additional Resources Required to Run the Sample Application
1	HelloWorld	The "hello world" application for ADF Mobile, which demonstrates the basic structure of the framework. This basic application has a single application feature that is implemented with a local HTML file. Use this application to ascertain that the development environment is set up correctly to compile and deploy an application. See also Section 4.2.2, "What Happens When You Create an ADF Mobile Application."	
2	CompGallery	This application serves as an introduction to the ADF Mobile AMX UI components by demonstrating all of these components. Using this application, you can change the attributes of these components and see the effects of those changes in real time without recompiling and redeploying the application after each change. See generally Chapter 8, "Creating ADF Mobile AMX User Interface."	
3	LayoutDemo	This application demonstrates the user interface layout and shows how to create the various list and button styles that are commonly used in mobile applications. It also demonstrates how to create the action sheet style of a popup component and how to use various chart and gauge components. See Section 8.3, "Creating and Using UI Components" and Section 8.5, "Providing Data Visualization."	This application must be opened from the `Samples` directory. The **Default** springboard option must be cleared in the Applications page of the `adfmf-application.xml` overview editor, then selected again.
4	JavaDemo	This application demonstrates how to bind the user interface to Java beans. It also demonstrates how to invoke EL bindings from the Java layer using the supplied utility classes. See also Section 8.10, "Using Event Listeners" and Section 9.2, "Understanding EL Support."	
5	Navigation	This application demonstrates the various navigation techniques in ADF Mobile, including bounded task flows and routers. It also demonstrates the various page transitions. See also Section 7.2, "Creating Task Flows."	This application must be opened from the `Samples` directory. The **Default** springboard option must be cleared in the Applications page of the `adfmf-application.xml` overview editor, then selected again.

Table E–1 (Cont.) ADF Mobile Sample Applications

Recommended Order of Use	Application Name	Description	Additional Resources Required to Run the Sample Application
6	LifecycleEvents	This application implements lifecycle event handlers on the ADF Mobile application itself and its embedded application features. This application shows you where to insert code to enable the applications to perform their own logic at certain points in the lifecycle. See also Section 5.6, "About Lifecycle Event Listeners."	For iOS, the LifecycleEvents sample application logs data to the Console application, located at Applications-Utilities-Console application.
7	DeviceDemo	This application shows you how to use the DeviceFeatures data control to expose such device features as geolocation, e-mail, SMS, and contacts, as well as how to query the device for its properties. See also Section 9.5, "Using the DeviceFeatures Data Control."	You must also run this application on an actual device, because SMS and some of the device properties do not function on an iOS simulator or Android emulator.
8	GestureDemo	This application demonstrates how gestures can be implemented and used in ADF Mobile applications. See also Section 8.4, "Enabling Gestures."	
9	StockTracker	This application demonstrates how data change events use Java to enable data changes to be reflected in the user interface. It also has a variety of layout use cases, gestures and basic mobile patterns. See also Section 9.7, "Data Change Events."	

Table E–1 (Cont.) ADF Mobile Sample Applications

Recommended Order of Use	Application Name	Description	Additional Resources Required to Run the Sample Application
10	HR	This human resources application is a CRUD application that demonstrates a variety of real-world application techniques. It uses a local SQLite database to store its data. The application persists the data between each startup and is based on the default HR schema that ships with all Oracle databases. See generally Chapter 11, "Using the Local Database." By providing layouts for both iPad and iPhone, this application demonstrates how different types of user interfaces can share the same data model. There are a variety of other patterns demonstrated in the application as well.	
11	Skinning	This application demonstrates how to skin applications and add a unique look and feel by either overriding the supplied style sheets or extending them with their own style sheets. This application also shows how skins control the styling of ADF Mobile AMX UI components based on the type of device. See also Section 5.11, "Skinning ADF Mobile Applications."	
12	PrefDemo	This application demonstrates application-wide and application feature-specific user setting pages. See generally Chapter 13, "Enabling User Preferences"	