Oracle® Fusion Middleware

Mobile Client Developer's Guide for Oracle Application Development Framework

11*g* Release 1 (11.1.1.5.0)

E14826-02

April 2011

ORACLE®

Oracle Fusion Middleware Mobile Client Developer's Guide for Oracle Application Development Framework 11*g* Release 1 (11.1.1.5.0)

E14826-02

Primary Authors: John Bassett, Liza Rekadze

Contents

3 Introduction to the ADF Mobile Client Sample Application

4 Getting Started with ADF Mobile Client

5 Developing the ADF Mobile Client Data Model

6 Creating the ADF Mobile Client User Interface

7 Extending ADF Mobile Client Applications with Java

8 Deploying ADF Mobile Client Components

9 Synchronizing ADF Mobile Client Data and Transactions

10 Testing and Debugging ADF Mobile Client Applications

Preface

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

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible to all users, including users that are disabled. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Accessibility standards will continue to evolve over time, and Oracle is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For more information, visit the Oracle Accessibility Program Web site at http://www.oracle.com/accessibility/.

Accessibility of Code Examples in Documentation

Screen readers may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, some screen readers may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation

This documentation may contain links to Web sites of other companies or organizations that Oracle does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/support/contact.html or visit http://www.oracle.com/accessibility/support.html if you are hearing impaired.

Audience

This document is intended for developers of mobile client applications for smartphones and mobile devices.

Related Documents

For more information, see the following:

- *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 Database Lite Developer's Guide*
- *Oracle Database Lite SQLite Mobile Client Guide*
- *Oracle Fusion Middleware Installation Guide for ADF Mobile Transaction Replay Service*
- *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*
- *Oracle Fusion Middleware Java API Reference for Oracle ADF Mobile Client*

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.

1 Introduction to ADF Mobile Client

This chapter introduces ADF Mobile client—an Oracle JDeveloper extension for developing rich applications that run natively on mobile devices and smartphones.

This chapter includes the following sections:

- Section 1.1, "Introduction to ADF Mobile Client"
- Section 1.2, "Infrastructure Requirements for Developing ADF Mobile Client Applications"
- Section 1.3, "Run-Time Architecture of ADF Mobile Client"
- Section 1.4, "Data and Transaction Synchronization"
- Section 1.5, "Introduction to ADF Mobile Client Application Development"

1.1 Introduction to ADF Mobile Client

ADF Mobile client lets you develop high-performance mobile applications that do not require a browser and that provide full functionality even when disconnected from the server.

ADF Mobile client is built upon the ADF Mobile platform and lets you extend an existing Oracle Application Development Framework (Oracle ADF) application to create a new application that will run on any platform that ADF Mobile client supports (see Section 1.5.4, "Supported Devices and the Supported Database"). ADF Mobile client does the following:

- Enables access to features of a mobile device or smartphone.
- Uses APIs of a mobile device or smartphone for improved performance.
- Uses UI of a mobile device or smartphone to provide applications with the native look-and-feel.

ADF Mobile client applications use Oracle Database Lite Mobile Server and Oracle ADF Mobile transaction replay service to synchronize data between back-end business data sources and the mobile device or smartphone.

1.1.1 Understanding Differences Between ADF Mobile Client Applications and ADF Faces Web Applications

The operation of an ADF Mobile client application is substantially different from the operation of an ADF Faces Web application. ADF Faces applications adhere to the standard JSF lifecycle for processing an HTTP request and response. In Web applications, these lifecycle phases help define at various points in the processing of an HTTP request which actions are valid, which values are initialized, and so forth. It is

also common for an HTML page to be discarded in the browser, with a completely new page then generated by the server and sent again to the browser (though this is changing with Web 2.0).

In contrast, ADF Mobile client implements an application model where the front end and back end reside on the same device. As it is inefficient to discard the UI every time a page transition occurs, or every time a value is submitted to the database, ADF Mobile client compresses the JSF lifecycle into a dynamic, always-on, event-driven model. This operating model is less like the traditional HTTP request-and-response model, and more like the Web 2.0 model, because a screen is created once and the user interface elements on it are updated dynamically based on the underlying data.

The ADF Mobile client consists of the following three basic layers:

- ADF Business Components: This layer operates very similarly to the ADF Business Components layer in an ADF Faces application. You use the JDeveloper business component design time to create entity objects and view objects, apply view criteria for query customization, and so on. You can also generate Java classes for your entity and view objects and program against a subset of the `oracle.jbo` package.
- ADF Model (JSR-227): This layer in ADF Mobile client is also very similar to ADF Faces: views are exposed through data controls; action bindings, method bindings, and iterator bindings are all available through the existing JDeveloper design time. However, it is this layer where some differences start to emerge. The current ADF Model layer is designed to support the JSF lifecycle among other things. Since the lifecycle does not exist in ADF Mobile client, some of the lifecycle-oriented constructs do not apply to ADF Mobile client applications.
- UI: This layer of ADF Mobile client applications is new, but it retains a few key concepts from ADF Faces. ADF Mobile client pages are defined by MCX files, which are similar to JSPX files yet greatly simplified and abstracted in order to support cross-platform development. Many of the basic components that are present in ADF Faces, such as the Input Text, Output Text, and Select One Choice are included in the MCX format. In addition, the components are "wired" to the model layer with JSF Expression Language (EL). Understanding how EL expressions transfer data between the model and UI layers is key to understanding the difference between ADF Faces applications and ADF Mobile client applications.

  In an ADF Faces application, the server processes the JSPX file during the *render response* phase and replaces EL expressions with actual values. This works for ADF Faces because a new page can be generated and sent to the browser each time a value changes. With ADF Mobile client, however, a page is usually created just once. In order to provide data updates to the controls that are populated with EL expressions, EL *event sources and sinks* are created for every EL expression on a page. Whenever the underlying value of an EL expression changes, any event sinks for that expression will automatically receive the update. The events transfer data from the business component layer to the UI layer when a record iterator moves, and the events transfer data from the UI layer to the business component layer when a user updates a value on a form. For more information and examples, see Section 6.14.4, "EL Events."

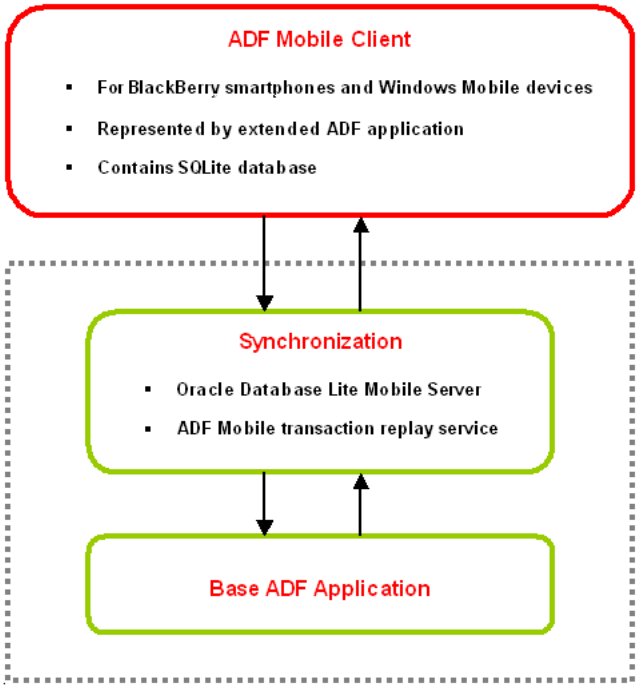
1.2 Infrastructure Requirements for Developing ADF Mobile Client Applications

A number of physical and logical components are required when developing mobile applications using ADF Mobile client.

The main functional component areas of ADF Mobile client are the mobile client itself and the related synchronization technologies. On the server side, ADF Mobile client application data is processed in the same way that data from regular ADF applications is processed, ensuring data integrity and similarity of business processes.

Very simplistically, the ADF Mobile client infrastructure can be expressed, as shown in Figure 1–1.

Figure 1–1 ADF Mobile Client Infrastructure



As the preceding illustration demonstrates, the synchronization technologies enable interactions between the existing ADF application (also known as the base application) including its database and the extended ADF Mobile client application.

For additional overview information on ADF Mobile client, see the following:

- Section 1.3, "Run-Time Architecture of ADF Mobile Client"
- Section 1.4, "Data and Transaction Synchronization"
- Section 1.5, "Introduction to ADF Mobile Client Application Development"

ADF Mobile client is compatible with Oracle ADF (see "Introduction to Oracle ADF" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*). The mobile client shares most of the features and functionality of Oracle ADF by supporting many ADF components, such as view objects, Expression Language (EL), UI components, and so on. An ADF Mobile client application is always an extension of an existing ADF application. You apply your existing ADF development skills to develop mobile client applications.

For more information on what Oracle ADF shares with ADF Mobile client, see the following:

- Section 1.5.5, "Supported ADF Components and Attributes"
- Section 1.5.6, "Support for ADF View Objects"
- Section 1.5.7, "Support for Expression Language"

The synchronization technologies employed by ADF Mobile client may be hosted on the same server as the base ADF application. When connectivity is available, the data from the mobile client application's SQLite database can be synchronized with the base application's data.

For more information on synchronization, see Section 1.4, "Data and Transaction Synchronization."

When planning the development of a mobile client application, even though requirements vary depending on the platform for which you are developing, you generally need the following:

- Oracle JDeveloper
- Oracle Database
- An existing ADF application
- Oracle JDeveloper extension for ADF Mobile client
- A mobile device or device simulator
- Oracle Database Lite Mobile Server for data synchronization
- ADF Mobile transaction replay service for business logic and transaction synchronization. In addition, the transaction replay service requires a running instance of a J2EE Server configured for ADF

For more information on prerequisites, see Section 2.2, "Prerequisites for Developing ADF Mobile Client Applications."

1.3 Run-Time Architecture of ADF Mobile Client

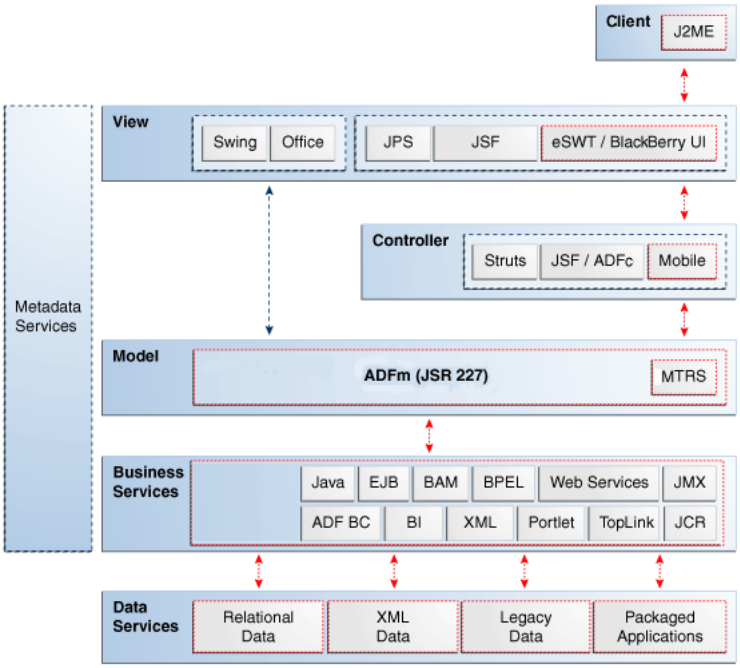
ADF Mobile client consists of the following parts:

- **View** expressed as a mobile device UI
- **Controller** powered by the ADF Mobile framework
- **Model** that includes the following:
  - local relational database
  - mobile ADF Business Components (BC4J) layer
- **Java runtime**

> **Note:** ADF Mobile client's model-view-controller stack resides on a mobile device or smartphone 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 JSR 227.

Figure 1–2 shows the ADF Mobile client architecture (in red) and how it fits into the overall ADF architecture.

Figure 1–2 ADF Mobile Client Run-Time Architecture Within ADF Architecture

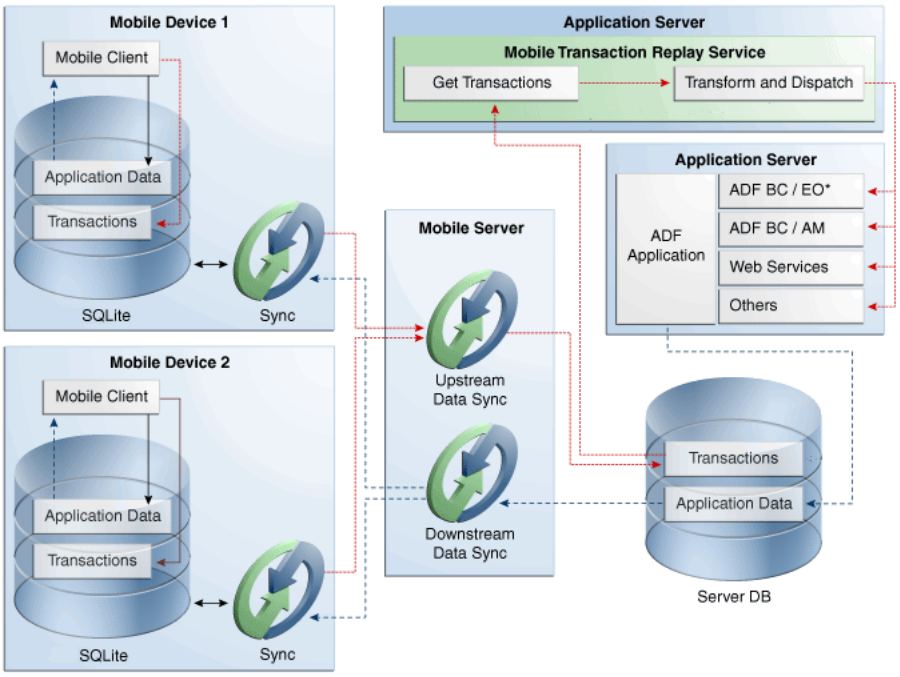


1.4 Data and Transaction Synchronization

Data integrity is the main goal of any data-processing application. Because ADF Mobile client enables functionality when the client is disconnected from the server and the main data source, it relies on synchronization technologies to preserve integrity of the application data. When data on the client is modified, a transaction is created to encapsulate those modifications. When connectivity becomes available, this transaction is recorded and replayed, triggering the update of data on the server.

Figure 1–3 shows how ADF Mobile client's run-time synchronization mechanism functions.

Figure 1–3 ADF Mobile Client Run-Time Synchronization



Note: The Entity Object Transaction Execution is supported by default in V1.

Note: The transaction replay service can reside on the same application server as an ADF application.

1.4.1 Data Synchronization

ADF Mobile client provides local data storage on the mobile device or smartphone by synchronizing a subset of application data down to the device. This frees you from the need to provide the data synchronization code for your application.

The mobile client enables the table-level data synchronization from server to client through the use of Oracle Database Lite Mobile Server.

For more information, see the following:

- Chapter 9, "Synchronizing ADF Mobile Client Data and Transactions"
- Section 9.3, "Initiating Data Synchronization"

1.4.2 Transaction Synchronization

Typically, server-side business logic cannot be duplicated or approximated on the client. To ensure data integrity, ADF Mobile client synchronizes device-side data transactions upstream via ADF Mobile transaction replay service. These transactions are then replayed through the ADF BC (Entity Object) layer of the base ADF application.

Basically, the transaction replay service enables the mobile client applications running on clients to access and execute business logic components on the server or on the base

application. It can replay transactions against a variety of back-end services from ADF Business Component (BC) entity object operations, ADF BC view operations, ADF BC application modules operations and even other interfaces such as web services, Java classes and other elements that can be bound using JSR 227.

The transaction replay service is implemented using J2EE technologies and modeled on the Service Oriented Architecture (SOA). It is independent of the type of the disconnected client, synchronization technologies, application server, and database platform.

The transaction replay service infrastructure resides on the server side of the enterprise application and accepts transactions posted from one or more disconnected clients to a transaction replay service-specific database table. The transactions in this table are configured to map with available Java methods, and are subsequently posted (or replayed) to the enterprise application on behalf of the disconnected client. Using the mapped Java methods for transaction replay allows the implementation of complete server-side business logic for remotely collected data. Server-side business logic can include validation, deferred event triggers, and so on.

Disconnected client transactions that can be replayed using the transaction replay service are dependent on the type of application, as well as on your or your administrator's configurations. You can configure the transaction replay service for use with your disconnected solution (see Section 5.8.4, "How to Create a Transaction Replay Type"). Application administrators can monitor the transaction status and performance of the transaction replay service while in use with a disconnected solution.

The following are examples of transactions that can be replayed:

- Creation of a new contact or account object record
- Updates to existing records

1.5 Introduction to ADF Mobile Client Application Development

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

ADF Mobile client offers you a declarative approach to application development through the use of drag-and-drop components, business logic wizards, and other productivity aids. This enables the following:

- Declarative UI design
- Declarative page navigation
- Declarative data binding

> **Note:** Even though the ADF Mobile client application is defined as XML with components expressed as XML tags, you have an option to customize it using Java code.

For more information, see Section 4.2, "About Developing an ADF Mobile Client Application."

1.5.1 Application Architecture

Your ADF Mobile client application will always extend an existing (base) ADF application:

- Your entity objects will be copies of a subset of the entity objects of the base application.
- Your view objects will either be based on entity objects, or they will be represented by static view objects.

For more information, see Section 5.2, "Extending an ADF Application to Mobile Client."

You use a wizard to generate your mobile client application's business components from the base ADF application's business components.

The following is a potential architecture for your ADF Mobile client application:

- A database-bound, mostly disconnected application that consists of several views bound to data in tables stored in a local database on the device. Data is synchronized from a server database to the mobile device or smartphone. This data synchronization is the primary communication mechanism between the server application along with its data sources and the ADF Mobile client application. The application functionality is largely supported without the use of network connectivity.

For more information, see Chapter 5, "Developing the ADF Mobile Client Data Model."

1.5.2 Typical Development Stages

As with most application development, you perform the following activities when building your ADF Mobile client application:

- Gather requirements
- Design
- Develop
- Test
- Deploy

A disconnectable mobile application is generally created as an extension of a server application. Although the mobile application should not mimic the server/desktop application in its *presentation*, its underlying data schema will mimic the server's data schema because it is a subset of that server schema.

It is assumed that you have a server-side database with an application schema already in place. The steps you take to build your ADF Mobile application will generally occur as follows:

1. **Design mobile tasks.** Consider the tasks a mobile user will be performing, keeping in mind that handheld usage is different from that of a laptop or desktop machine. How will the 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. Can peripheral input devices be used (barcode scanners or cameras) to simplify processing? Design your mobile client application to be a genuine assistant to the mobile worker, not just a new data collection tool.
2. **Design a mobile data schema.** The tasks you are bringing to the mobile application will have some server-side data representation, and likely they will have some server validation code as well. Use that information as a starting point for creation of your mobile application. Examine your server schema and identify the subset of tables and columns that will need to be available in the handheld application.

3. **Set up your work environment.** Install the necessary applications and complete the required setup for development and deployment. For more information, see Chapter 2, "Setting Up the ADF Mobile Client Environment."
4. **Create your mobile client application using JDeveloper**:
   - Build your mobile client business services (edit entity objects, create mobile views).
   - Develop mobile client task flows.
   - Design and refine the user interface.

   For more information, see Chapter 5, "Developing the ADF Mobile Client Data Model" and Chapter 6, "Creating the ADF Mobile Client User Interface."
5. **Deploy the application to a mobile device, smartphone, device emulator, or smartphone simulator**:
   - Set up data synchronization.
   - Create database connections.
   - Set up the mobile device, smartphone, or simulator.
   - Synchronize the data.
   - Deploy the client runtime.
   - Deploy the mobile application.

   Note: With ADF Mobile client, it is required that you deploy to the mobile device, smartphone, device emulator, or smartphone simulator before doing any testing and debugging. A mobile client application cannot be run until you deploy it.

   For more information, see Chapter 8, "Deploying ADF Mobile Client Components."
6. **Test and debug the application:**
   - Test synchronization.
   - Test and debug the application.
   - Optimize performance.

   For more information, see Chapter 10, "Testing and Debugging ADF Mobile Client Applications."
7. **Deploy the application to users.**

   For more information, see the section about ADF Mobile client deployment in *Oracle Fusion Middleware Administrator's Guide for Oracle Application Development Framework*.

1.5.3 The Lifecycle of a View

Unlike ADF, the mobile client does not represent a lifecycle-driven system: once its view (page) is created, the flow of data from the UI to the model and back is immediate and continuous. In other words, a page is live as long as it is displayed, which makes Expression Language (EL) expressions live as well. For more information on EL, see Section 1.5.7, "Support for Expression Language."

For more information on ADF lifecycle, see "Understanding the Fusion Page Lifecycle" chapter of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

For more information on the JSF extended lifecycle, see "Understanding the JSF and ADF Faces Lifecycles" chapter of *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

1.5.4 Supported Devices and the Supported Database

ADF Mobile client supports BlackBerry smartphones and Windows Mobile devices. The supported database for these devices is SQLite (see Section 1.5.4.1, "What You May Need to Know About SQLite").

1.5.4.1 What You May Need to Know About SQLite

SQLite is a relational database management system (RDBMS) specifically designed for embedded applications. SQLite databases are entirely self-contained in a single, portable file that is binary-compatible across a diverse range of computer architectures and operating systems.

SQLite has the following characteristics:

- It is embedded.
- It is ACID-compliant (atomicity, consistency, isolation, durability).
- It is contained in a small library.
- It is weakly-typed: any object can be stored in any column, regardless of how that column was declared.
- It does not officially support foreign key constraints, although triggers can be used as a workaround.
- It does not support `RIGHT OUTER JOIN`s.

Your application can either call the SQLite library dynamically through function calls, or make it part of the application itself by creating a link to the library.

For more information, see `http://www.sqlite.org`

1.5.5 Supported ADF Components and Attributes

ADF Mobile client supports many ADF components and attributes. Refer to the following information, organized by layer:

- Model layer. See Section 5.1.1, "Support for the Core ADF Business Components."
- Binding layer. See the following:
  - Section 6.15, "Understanding Binding Layer Components"
  - Section 6.3.2.3, "Adding Data Controls to the View"
  - Section 1.5.7, "Support for Expression Language"
- UI layer. See Chapter 6, "Creating the ADF Mobile Client User Interface."

Although ADF Mobile client is compatible with ADF, there is a number of differences in every layer that exist due to the inherent differences between a mobile application and a Web application.

1.5.6 Support for ADF View Objects

As stated in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*, a view object is an Oracle ADF component that encapsulates an SQL query and simplifies working with its results.

View objects can access the database directly for read operations.

ADF Mobile client provides support for ADF view object functionality, with limited support for database field types.

> **Note:** With ADF Mobile client, you cannot create view objects that are based on SQL statements.

For more information, see the following:

- "Introduction to View Objects" chapter of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects"
- Section 5.3.8, "View Accessor Support for Entity Objects and View Objects"
- Section 5.4, "The Entity Object and View Object Extension"

1.5.7 Support for Expression Language

You use the Expression Language (EL) to enable data binding. For an overview of the use of EL with Oracle ADF, see "Using ADF Model in a Fusion Web Application" chapter in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

ADF Mobile client supports the use of compound EL expressions. For more information, see Section 6.14, "Understanding EL Support."

ADF objects exist within different scopes, such as the application scope, session scope, page flow scope, and so on (see "About Object Scope Lifecycles" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*). ADF Mobile client, however, only supports the application scope. EL expressions defined in the application scope namespace are available for the life of the application. With the mobile client, you can define an application scope on one view of an application, and then reference it on another.

For more information, see the following:

- Section 1.5.3, "The Lifecycle of a View."
- Section 6.14, "Understanding EL Support."

2

Setting Up the ADF Mobile Client Environment

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

This chapter includes the following sections:

2.1 Introduction to the ADF Mobile Client Environment

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

2.2 Prerequisites for Developing ADF Mobile Client Applications

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

- What You Need to Get Started With the ADF Mobile Client Sample Application
- What You Need to Create an ADF Mobile Client Application
- What You Need to Deploy an ADF Mobile Client Application to a Development Environment

2.2.1 What You Need to Get Started With the ADF Mobile Client Sample Application

Before you start working with the mobile client sample application (see Chapter 3, "Introduction to the ADF Mobile Client Sample Application"), ensure that you have the following components installed:

- Oracle JDeveloper (see Section 2.3, "Setting Up JDeveloper")
- Oracle JDeveloper extension for ADF Mobile client (see Section 2.3, "Setting Up JDeveloper")
- Oracle Database (Standard or Enterprise Edition) (see Section 2.4, "Setting Up Oracle Database")
- Oracle Database Lite (see Section 2.5, "Setting Up Oracle Database Lite")
- Fusion Order Demo Mobile Client application (see Section 2.9, "Setting Up the Fusion Order Demo Mobile Client Application")
- The sample application deployed to a smartphone, mobile device, or emulator (see Section 3.4, "Running the Fusion Order Demo Mobile Client Application")

In addition:

- If Windows Mobile is your target platform, the following is required:
  - Java Runtime Environment (JRE) in the form of Oracle Java Micro Edition Connected Device Configuration HotSpot Implementation (Oracle Java ME CDC HI)
  - Windows Mobile SDK.

    Note that this software is required if you are planning to use a Windows Mobile device emulator only.
  - Windows Mobile device or device emulator

  For more information, see Section 2.6, "Setting Up Development Tools for Windows Mobile Platform."
- If BlackBerry is your target platform, the following is required:
  - BlackBerry JDE
  - BlackBerry smartphone or simulator.

    Note that BlackBerry JDE download includes a number of smartphone simulators. You only need to install a separate, standalone simulator if the JDE does not include a simulator for your target device.
  - BlackBerry Desktop Software

  For more information, see Section 2.7, "Setting Up Development Tools for BlackBerry Platform."

2.2.2 What You Need to Create an ADF Mobile Client Application

Before you start creating a mobile client application (see Chapter 4, "Getting Started with ADF Mobile Client"), ensure that you have the following components installed:

- Oracle JDeveloper (see Section 2.3, "Setting Up JDeveloper")
- Oracle JDeveloper extension for ADF Mobile client (see Section 2.3, "Setting Up JDeveloper")
- Oracle Database (Standard or Enterprise Edition) (see Section 2.4, "Setting Up Oracle Database")

- Oracle Database Lite (see Section 2.5, "Setting Up Oracle Database Lite")
- An existing ADF application: because you are extending an application to a mobile device, you cannot begin application development by creating a standalone mobile client application. An ADF application must first exist and stand as the base (server) application for the mobile client. The view objects and entity objects used by the mobile client application are likewise based on the view objects and entity objects of the base application. For more information, see Section 4.5, "Extending the Base Application for the Mobile Client Application."
- Synchronization initiated using mSync (see Section 9.3, "Initiating Data Synchronization").

In addition:

- If Windows Mobile is your target platform, the following is required:
  - Java Runtime Environment (JRE) in the form of Oracle Java Micro Edition Connected Device Configuration HotSpot Implementation
  - Windows Mobile SDK

    Note that this software is required if you are planning to use a Windows Mobile device emulator only
  - Windows Mobile device or device emulator

  For more information, see Section 2.6, "Setting Up Development Tools for Windows Mobile Platform."
- If BlackBerry is your target platform, the following is required:
  - BlackBerry JDE
  - BlackBerry smartphone or simulator.

    Note that BlackBerry JDE download includes a number of smartphone simulators. You only need to install a separate, standalone simulator if the JDE does not include a simulator for your target device.
  - BlackBerry Desktop Software

    Note that this software is required if you are planning to use a BlackBerry smartphone only.

  For more information, see Section 2.7, "Setting Up Development Tools for BlackBerry Platform."

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

Before you start deploying your mobile client application (see Chapter 4, "Getting Started with ADF Mobile Client"), ensure that you have the following components installed:

- All components listed in Section 2.2.2, "What You Need to Create an ADF Mobile Client Application"
- The mobile client application
- Various database connections and login credentials
- The server application's model project deployed as an ADF library. This library contains ADF Business Components from the base ADF application that you are

extending to the mobile client application (see Section 4.3, "Deploying the Model Project of the Base Application as an ADF Library").

In addition, if you are deploying to Windows Mobile devices, install the following:

- Microsoft ActiveSync 4.5 or Microsoft Windows Mobile Device Center 6.1 or later: this component connects a mobile device or emulator to your computer and transfers your application to the device or emulator once it is running (see Section 2.6.3, "How to Connect the Mobile Device or Emulator").
- Microsoft Device Emulator 3.0: this component represents a program that can emulate Window Mobile devices (see Section 2.6.2, "How to Install and Set Up the Windows Mobile Emulator").
- One or more Microsoft emulator images that enable functionality of Microsoft Device Emulator (see Section 2.6.2, "How to Install and Set Up the Windows Mobile Emulator").

For more information, see Section 2.6, "Setting Up Development Tools for Windows Mobile Platform."

2.2.4 What You May Need to Know About Pre-Installed Components

There is a number of additional components that are part of the mobile client environment setup with the purpose of the application development and deployment. However, these components do not require explicit installation as they are installed by default during installation of other components. These pre-installed components include the following:

- SQLite database: this component comes with Oracle Database Lite and is preinstalled on BlackBerry smartphones. For Windows Mobile devices, the required files are installed as part of the synchronization and runtime installation. Moreover, the sample application includes an SQL script to automatically generate the database.
- SQLite Mobile Client installed on the mobile device or emulator (see Section 2.8, "Installing SQLite Mobile Sync Client on BlackBerry Smartphone or Simulator"): this component comes with Oracle Database Lite.
- BlackBerry Desktop Manager pack: this component is included in the BlackBerry Desktop Software installation (see Section 2.7.2, "How to Install BlackBerry Desktop Software").
- Microsoft Device Emulator Manager: this component is installed as part of Microsoft Device Emulator 3.0 package. The Device Emulator Manager simulates an ActiveSync connection. It connects the emulator for file synchronization using Microsoft ActiveSync (see Section 2.6.2, "How to Install and Set Up the Windows Mobile Emulator").
- Oracle Java Micro Edition Connected Device Configuration HotSpot Implementation: this component is included in the ADF Mobile client extension as the Windows Mobile JVM (see Section 2.6.5, "How to Install Java Runtime Environment on the Mobile Device or Emulator").
- Oracle Database Lite Mobile Server: this component is a part of Oracle Database Lite and enables data synchronization (see Section 2.5, "Setting Up Oracle Database Lite").
- BlackBerry Mobile Data Service Simulator (MDS simulator): this component installs with the BlackBerry JDE and represents the BlackBerry network access

server (see Section 2.7.1.1, "What You May Need to Know About BlackBerry Mobile Data Service Simulator").

2.3 Setting Up JDeveloper

Setting up your ADF Mobile client development environment begins with Oracle JDeveloper and its ADF Mobile client extension.

Before you develop an application or run the ADF Mobile client sample application, you must perform the following steps:

1. Download and install Oracle JDeveloper.

   For more information, see *Oracle Fusion Middleware Installation Guide for Oracle JDeveloper*.

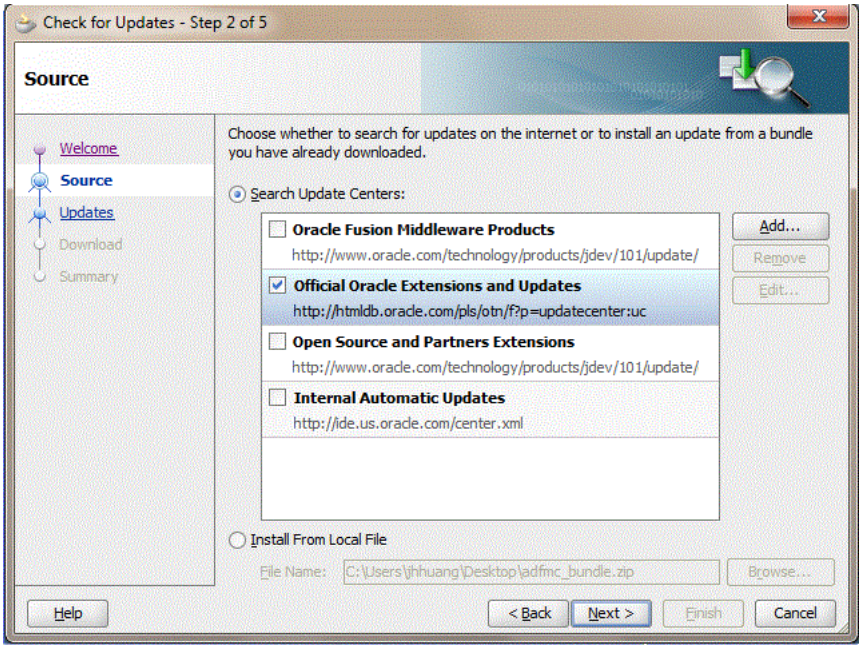
2. Download the Fusion Order Demo Mobile client application ZIP file.

   For more information, see "Introduction to the ADF Sample Application" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

3. Download the ADF Mobile client extension as follows:

   1. In JDeveloper, choose **Help**, then **Check for Updates**, and then click **Next**.
   2. In the **Source** page that Figure 2–1 shows, select **Official Oracle Extensions and Updates**, and then click **Next**.

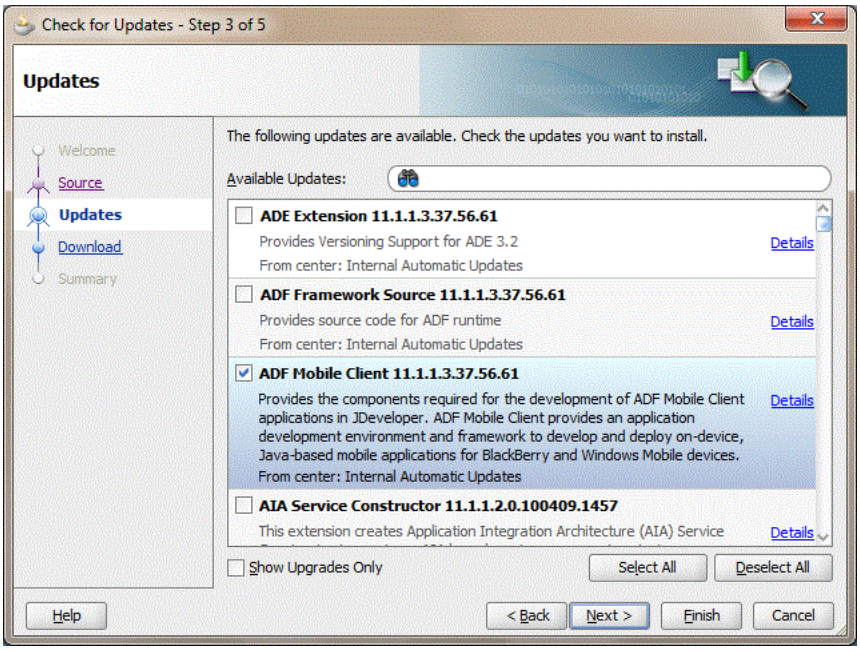
Figure 2–1 Selecting the ADF Mobile Client Extension



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.

3. In the **Updates** dialog that Figure 2–2 shows, select the **ADF Mobile Client** update.

Figure 2–2 Selecting the ADF Mobile Client Components



4. Click **Next**, and then click **Finish**.

2.4 Setting Up Oracle Database

While the ADF Mobile client application does not require a database, the server-side application with which it synchronizes does. For this reason, you must install Oracle Database.

This database hosts the application data and the Oracle Database Lite Mobile Server repository schema. For more information, see Section 2.5, "Setting Up Oracle Database Lite."

> **Note:** Since Oracle Database Lite Mobile Server requires either the Standard or Enterprise Edition of Oracle Database, you cannot use the Express Edition.

Because there is no local application database when you first create the mobile client application, you must be able to access the same Oracle Database that houses the data used by the server-side application you are extending.

Download Oracle Database from
`http://www.oracle.com/technology/software/products/database/index.html`

2.5 Setting Up Oracle Database Lite

If your mobile client application requires data to be synchronized from a server database, you need to install Oracle Database Lite by downloading it from

http://www.oracle.com/technology/software/products/lite/index.html

Oracle Database Lite installation provides the following tools necessary for developing ADF Mobile client applications:

- Mobile Database Workbench (MDW), which is a development tool for creating publications and publication items. You can also use MDW to view and modify publications and publication items after application data has been published to the device. For more information, see *Oracle Database Lite Developer's Guide*.
- Mobile Sync Client (mSync), which is a small footprint application that resides on a mobile device and enables you to synchronize published application data between Oracle databases, handheld devices, and desktop and laptop computers.
- Oracle Database Lite Mobile Server, which is a server-side component that works in conjunction with Oracle Database and mSync to synchronize data between multiple client devices and computers.
- Oracle Database Lite 10g MDK. For more information, see *Oracle Database Lite Developer's Guide*.

Note: After installing any component of Oracle Database Lite, you must remove the following two entries from your `PATH` environment variable, and then restart your system:

1. `<OLITE_HOME>\jre\1.4.2\bin`
2. `<OLITE_HOME>\jre\1.4.2\bin\client`

For more information, see the following:

- *Oracle Database Lite Getting Started Guide*
- Chapter 9, "Synchronizing ADF Mobile Client Data and Transactions"

2.6 Setting Up Development Tools for Windows Mobile Platform

In addition to general-purpose tools listed in Section 2.2.2, "What You Need to Create an ADF Mobile Client Application," getting ready for development of an ADF Mobile client application for the Windows Mobile platform involves installation and configuration of the following tools:

- A Windows Mobile device (see Section 2.6.1, "How to Set Up the Windows Mobile Device") or device emulator (see Section 2.6.2, "How to Install and Set Up the Windows Mobile Emulator")
- A connection utility (see Section 2.6.3, "How to Connect the Mobile Device or Emulator")
- Oracle Database Lite on the mobile device or emulator (see Section 2.6.4, "How to Install the Oracle Database Lite Client on the Mobile Device or Emulator")
- Java Runtime Environment (JRE) (see Section 2.6.5, "How to Install Java Runtime Environment on the Mobile Device or Emulator")

2.6.1 How to Set Up the Windows Mobile Device

ADF Mobile client supports the following Windows Mobile devices:

- Windows Mobile 6.0 Professional

- Windows Mobile 6.1 Professional
- Windows Mobile 6.5 Professional

To set up your mobile device, you simply have to connect it to the USB port of the computer running JDeveloper. At this point, either Microsoft ActiveSync or Microsoft Windows Mobile Device Center establishes a connection from the mobile device to your development computer (see Section 2.6.3, "How to Connect the Mobile Device or Emulator").

Note: Only a single Windows Mobile device can be plugged in at any given time. If ActiveSync or Microsoft Windows Mobile Device Center does not automatically initiate a connection to the device, check to make sure no other device is attached and no emulator is running.

Also try selecting or deselecting the **Enable advanced network functionality** option on your device. You access this option through **Settings >USB to PC** menu.

2.6.2 How to Install and Set Up the Windows Mobile Emulator

During development, you can use the mobile device emulator instead of an actual device (see Section 2.2, "Prerequisites for Developing ADF Mobile Client Applications") to test your application.

ADF Mobile client supports the following device emulators:

- Windows Mobile 6 Professional
- Windows Mobile 6.1 Professional
- Windows Mobile 6.5 Professional

Note: The current ADF Mobile client release does not provide support for Windows Mobile Standard and Windows Mobile 7 (Windows Phone 7) device emulators.

Before you begin:

Download and install one of the following Windows Mobile SDK packages:

- Windows Mobile 6 Professional SDK from Microsoft download site at `http://www.microsoft.com/downloads/en`
- Windows Mobile 6.5 Developer Tool Kit from Microsoft download site at `http://www.microsoft.com/downloads/en`

Note: Since there is no Windows Mobile 6.1 SDK, you should download Windows Mobile 6 Professional SDK for development using Windows Mobile 6.1 Professional device emulator.

To complete the set-up:

- Download and install Microsoft Device Emulator 3.0 from Microsoft download site at `http://www.microsoft.com/downloads/en`

For information on how to use Microsoft Device Emulator, search http://msdn.microsoft.com site for "Step by Step: Using Microsoft Device Emulator In-Depth in Your Application Development Experience".

- Download one or more Microsoft emulator images that enable functionality of Microsoft Device Emulator 3.0:
  - For Windows Mobile 6: http://www.microsoft.com/downloads/en
  - For Windows Mobile 6.1: http://www.microsoft.com/downloads/en
  - For Windows Mobile 6.5: http://www.microsoft.com/downloads/en

Figure 2–3 shows a Windows Mobile device emulator.

Figure 2–3 Windows Mobile Device Emulator

After you install the Windows Mobile device emulator, you have to connect it to your computer (see Section 2.6.3, "How to Connect the Mobile Device or Emulator").

2.6.3 How to Connect the Mobile Device or Emulator

To connect your development computer to the mobile device or emulator and enable transfer of your mobile client application to the mobile device or emulator once it is running, you use one of the following utilities:

- Microsoft ActiveSync 4.5: This installation is required for Microsoft Windows XP and 2000 systems and is available from Microsoft download site (see also http://www.microsoft.com/windowsphone/en-ca/apps/65-downloads.aspx).
- Microsoft Windows Mobile Device Center 6.1 or later: This installation is required for Microsoft Windows Vista and 7 systems and is available from Microsoft download site (see also http://www.microsoft.com/downloads/en).

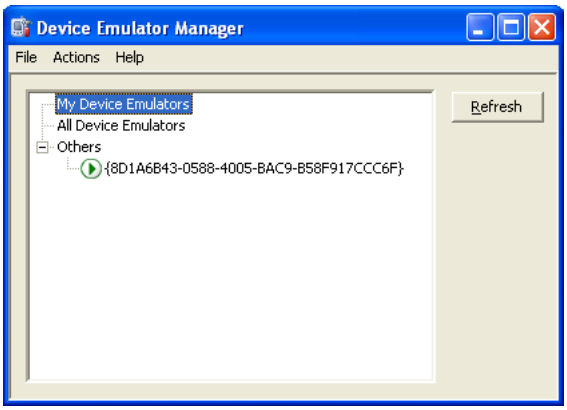
Once installed, this utility starts automatically when you plug in the mobile device or cradle the emulator.

To connect to the Windows Mobile device emulator using Microsoft ActiveSync:

1. From the Windows Start menu, select **Program Files** > **Windows Mobile 6 SDK** > **Standalone Emulator Images**, and then select an emulator.
2. From the Windows Start menu, choose **Program Files** > **Windows Mobile 6 SDK** > **Tools** > **Device Emulator Manager**.

3. From the Device Emulator Manager, click **Refresh**, and then select the preferred simulator, as Figure 2–4 shows.

Figure 2–4 The Device Emulator Manager

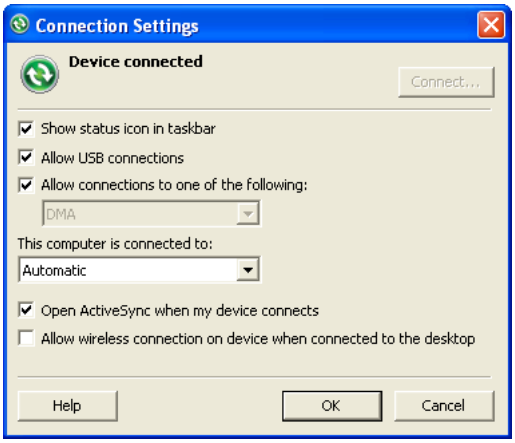


4. Choose **Actions**, and then select **Cradle** from the context menu. ActiveSync will automatically appear upon successful connection.

> **Note:** If you are prompted by ActiveSync to establish a partnership, you can dismiss this dialog by clicking Cancel.

5. Ensure that **Allow connections to one of the following** and **DMA** are both selected, as Figure 2–5 shows, and then click **OK**.

Figure 2–5 Setting the Connection in Microsoft ActiveSync



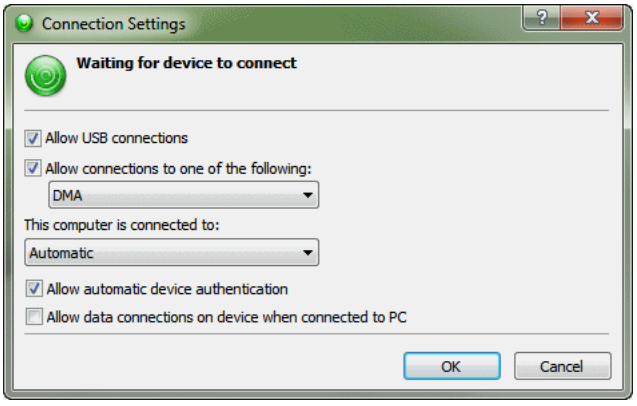
To connect to the Windows Mobile device emulator using Windows Mobile Device Center:

1. From the Windows Start menu, select **Program Files** > **Windows Mobile Device Center** to open the **Windows Mobile Device Center** dialog (see Figure 2–6).

Figure 2–6 Windows Mobile Device Center

2. Select **Mobile Device Settings**, and then click **Connection Settings** (see Figure 2–7).

Figure 2–7 Connections Settings Dialog



3. On the **Connection Settings** dialog, select **Allow connections to one of the following**, and then select **DMA**.
4. Click **OK**.

2.6.4 How to Install the Oracle Database Lite Client on the Mobile Device or Emulator

You must install the Oracle Database Lite client on the mobile device or emulator in order to synchronize data and access SQLite databases.

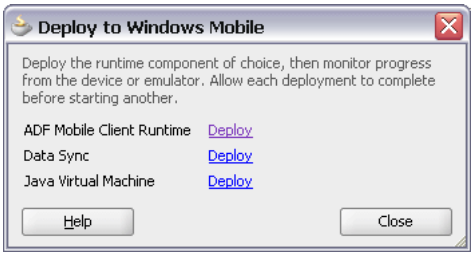
Before you begin:

Ensure that your computer and mobile device are connected (see Section 2.6.3, "How to Connect the Mobile Device or Emulator").

To install the Oracle Database Lite client:

1. In JDeveloper, from the main menu select **Tools** > **Deploy ADF Mobile Client Runtime** > **to Windows Mobile device/emulator**.
2. On the **Deploy to Windows Mobile** dialog that Figure 2–8 shows, select **Data Sync > Deploy**, and then click **OK**.

Figure 2–8 Deploy to Windows Mobile Dialog



3. To complete the installation, follow the instructions that appear on the screen of your device or emulator.

2.6.5 How to Install Java Runtime Environment on the Mobile Device or Emulator

To run ADF Mobile client applications, you must have Oracle Java Micro Edition Connected Device Configuration HotSpot Implementation (Oracle Java ME CDC HI) installed on the mobile device or emulator.

Before you begin:

Ensure that your computer and mobile device or emulator are connected (see Section 2.6.3, "How to Connect the Mobile Device or Emulator").

To install JRE:

1. In JDeveloper, from the main menu select **Tools** > **Deploy ADF Mobile Client Runtime** > **to Windows Mobile device/emulator**.
2. On the **Deploy to Windows Mobile** dialog that Figure 2–8 shows, select **Java Virtual Machine > Deploy**, and then click **OK**.
3. To complete the installation, follow the instructions that appear on the screen of your device or emulator.

> **Note:** If you are using the Window Mobile device emulator, you must install JRE to its internal main memory. JRE does not function properly if installed to the emulator's storage card.

2.6.6 What You May Need to Know About Limitations of Windows Mobile Platform Usage

There is a number of limitations that are associated with the usage of Windows Mobile platform in the current release of ADF Mobile client:

- Windows Mobile Standard and Windows Mobile 7 (Windows Phone 7) device emulators are not supported.
- At any given time, only a single Windows Mobile device or emulator can be connected to your development computer. If ActiveSync or Microsoft Windows Mobile Device Center does not automatically initiate a connection to the device, you have to ensure no other device is attached and no emulator is running. You may also try selecting or deselecting the **Enable advanced network functionality** option on your device. This option is accessible through the **Settings >USB to PC** menu.
- The Window Mobile JVM does not function properly if installed to the device emulator's storage card. Instead, it must be installed to the internal main memory of the emulator.

2.7 Setting Up Development Tools for BlackBerry Platform

In addition to general-purpose tools listed in Section 2.2.2, "What You Need to Create an ADF Mobile Client Application," getting ready for development of an ADF Mobile client application for BlackBerry platform involves installation and configuration of the following:

- BlackBerry JDE (see Section 2.7.1, "How to Install BlackBerry JDE")
- BlackBerry Desktop Software (see Section 2.7.2, "How to Install BlackBerry Desktop Software")
- BlackBerry smartphone (see Section 2.7.3, "How to Set Up a BlackBerry Smartphone") or simulator (see Section 2.7.4, "How to Set Up a BlackBerry Smartphone Simulator")

2.7.1 How to Install BlackBerry JDE

BlackBerry JDE includes all of the Java libraries and development tools that you need to build applications for a BlackBerry smartphone.

ADF Mobile client provides support for BlackBerry JDE 5.0 and 6.0.

To install BlackBerry JDE:

Download BlackBerry JDE from
`http://na.blackberry.com/eng/developers/javaappdev/javadevenv.jsp`

> **Note:** Due to certain limitations, if you are using Microsoft Windows Vista or 7 system, you should download BlackBerry JDE 6.0. For more information, see Section 2.7.8, "What You May Need to Know About Limitations of BlackBerry Platform Usage."

Figure 2–9 shows a BlackBerry smartphone simulator.

Figure 2–9 BlackBerry Smartphone Simulator Screen

2.7.1.1 What You May Need to Know About BlackBerry Mobile Data Service Simulator

The BlackBerry Mobile Data Service (MDS) Simulator is a part of the BlackBerry JDE and simulates a BlackBerry Enterprise Server for the simulator, providing network access for the smartphone simulator.

For data security and network bandwidth efficiency, RIM provides server components (BES and BIS) that act as a gateway between a BlackBerry smartphone and corporate networks or the Internet. BlackBerry MDS Simulator is capable of simulating these server components on your development computer for testing purposes.

2.7.2 How to Install BlackBerry Desktop Software

BlackBerry Desktop Software connects physical BlackBerry smartphones to your development computer and enables you to install applications to those smartphones.

Note: If you are planning to work with smartphone simulators only, this software is not required.

To install BlackBerry Desktop Software:

1. Download BlackBerry Desktop Software from the download site.
2. Run the installer and follow its on-screen instructions.

2.7.3 How to Set Up a BlackBerry Smartphone

In your mobile client application development and deployment, you can use either the BlackBerry smartphone itself, or its simulator (see Section 2.7.4, "How to Set Up a BlackBerry Smartphone Simulator"). If you are planning to use the smartphone, you simply need to connect it to your computer in order to establish a link between the two devices.

2.7.4 How to Set Up a BlackBerry Smartphone Simulator

In your mobile client application development and deployment, you can use either the BlackBerry smartphone itself (see Section 2.7.3, "How to Set Up a BlackBerry

Smartphone") 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.

If you are planning to use one of the smartphone simulators included with JDE, you do not need to download and install a separate smartphone simulator. In case JDE does not contain a simulator for your target BlackBerry smartphone and you need to download a standalone simulator, see the how-to section of the ADF Mobile page on Oracle Technology Network (OTN) at `http://www.oracle.com/technetwork/developer-tools/adf/overview/adf-mobile-096323.html` for instructions on installing these simulators.

To set up the BlackBerry smartphone simulator:

1. Start the BlackBerry Mobile Data Service (MDS) simulator by double-clicking the `run.bat` file, which is typically located in `C:\Program Files\Research In Motion\BlackBerry JDE <version number>\MDS` directory.

   Note: For information on how to configure MDS simulator to work behind a proxy, see Section 2.7.6, "How to Configure Proxy Settings."

2. Start the smartphone simulator.

   To activate the default simulator, double-click the `defaultSimulator.bat` file that is typically located in `C:\Program Files\Research In Motion\BlackBerry JDE <version number>\simulator\` directory.

2.7.5 How to Configure JDeveloper for BlackBerry Development

After you install the following components, you have to configure JDeveloper to make use of them:

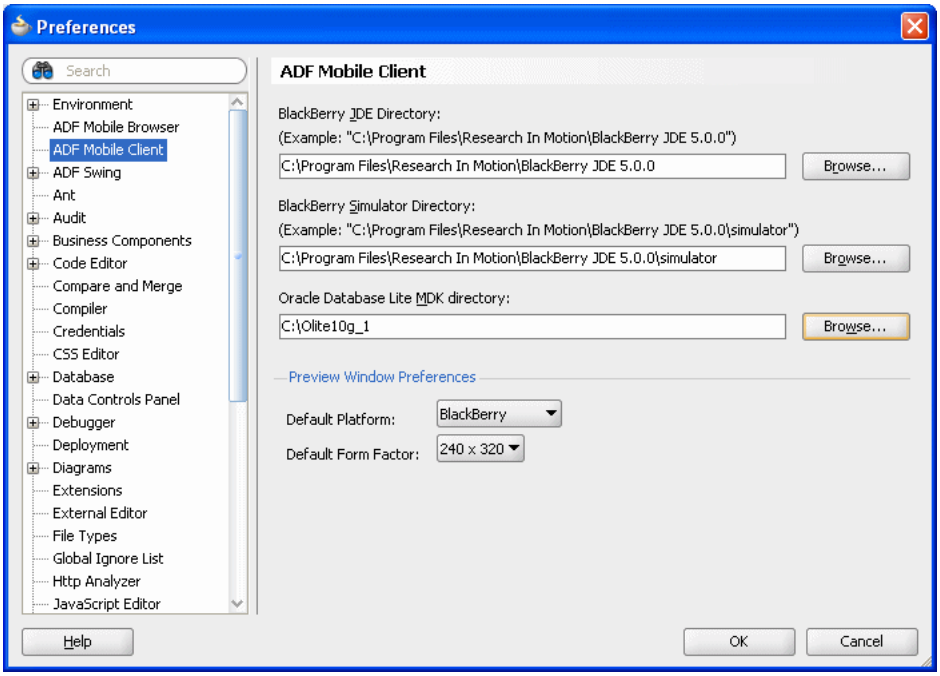
- BlackBerry JDE (see Section 2.7.1, "How to Install BlackBerry JDE")
- Optionally: standalone BlackBerry smartphone simulators, if the JDE does not contain a simulator for your target BlackBerry smartphone (see Section 2.7.4, "How to Set Up a BlackBerry Smartphone Simulator")
- Optionally: Oracle Database Lite 10g MDK, if your application requires data synchronization (see Section 2.5, "Setting Up Oracle Database Lite")

To configure JDeveloper for BlackBerry development:

1. In JDeveloper, select **Tools** > **Preferences** > **ADF Mobile Client** from the main menu.
2. Set the directory of the BlackBerry JDE. Because the default BlackBerry simulator installs with the JDE, the location of the BlackBerry simulator is populated automatically when you enter the location for the BlackBerry JDE.
3. If you use a simulator that is not part of BlackBerry JDE, then enter the location of this simulator.
4. Enter the root location of the MDK installation.
5. Click **OK**.

   Figure 2–10 shows the locations set for BlackBerry JDE, the included simulator, and Oracle Database Lite MDK installation directory.

Figure 2–10 Set BlackBerry JDE Location for Deployment



2.7.6 How to Configure Proxy Settings

You might need to configure network proxy settings by modifying the `rimpublic.property` file that is typically located in the `C:\Program Files\Research In Motion\BlackBerry JDE <version number>\MDS\config` directory. The configuration settings vary depending on the proxy server that is required to access the network, and may include any of the following:

```
[HTTP HANDLER]
application.handler.http.logging = true
application.handler.http.CookieSupport = true
application.handler.http.AuthenticationSupport = true
application.handler.http.AuthenticationTimeout = 3600000
application.handler.http.device.connection.timeout = 120000
application.handler.http.server.connection.timeout = 120000
application.handler.http.proxyEnabled = true
application.handler.http.proxyAutoConfig = true
application.handler.http.proxyAutoConfigURL=http://wpad.us.oracle.com/wpad.dat
application.handler.http.proxyAutoConfig.ScriptCacheTime=0
```

For more information on how to configure MDS simulator to work behind a proxy, see BlackBerry Developers Knowledge Base page at
http://www.blackberry.com/knowledgecenterpublic

2.7.7 What You May Need to Know About BlackBerry File Browser System

The BlackBerry file browser may not display the true path to a file. For instance, `/Media Card` is displayed for `/SDCard` directory, and `/Device Memory` is displayed for `/store` directory. Therefore, when referencing paths on the BlackBerry file system, either in code or in configuration files (such as `adf-config.xml` and `connections.xml`), ensure that you use the actual directory names.

2.7.8 What You May Need to Know About Limitations of BlackBerry Platform Usage

Even though the mobile client supports both BlackBerry JDE 5.0 and 6.0, there is a number of issues that are associated with the usage of these environments:

- BlackBerry JDE 5.0: It is not possible to close a BlackBerry smartphone simulator from the UI on Microsoft Windows Vista or 7 systems. The only means available to terminate the simulator is to use the End Task utility of the Microsoft Windows Task Manager on the `fledge.exe` file to force stop the program. When this happens, the state of the simulator is not saved, resulting in the loss of information as the data is not saved on the simulator between the restarts. In addition, exceptions occur and the simulator is left in a faulty state.
- BlackBerry JDE 6.0: The standalone synchronization of data is not possible. To transfer data to the smartphone, you can only rely on the integrated synchronization that is invoked by the application when data is not found on the smartphone. For more information, see Chapter 9, "Synchronizing ADF Mobile Client Data and Transactions."

Table 2–1 summarizes compatibility of BlackBerry components and specific Microsoft Windows platforms.

Table 2–1 Platform Compatibility

Microsoft Windows platform	BlackBerry JDE	BlackBerry Desktop Software	BlackBerry smartphone simulator
32-bit editions of Microsoft Windows XP, Vista, and 7	5.0 and 6.0	5.0 and 6.0	Included with JDE 5.0 and 6.0
64-bit editions of Microsoft Windows XP, Vista, and 7	6.0	6.0	Included with JDE 6.0

2.8 Installing SQLite Mobile Sync Client on BlackBerry Smartphone or Simulator

To synchronize data, you must install the SQLite Mobile Sync client on BlackBerry smartphone or simulator.

Before you begin:

Ensure that your computer and the smartphone or simulator are connected (see Section 2.7.3, "How to Set Up a BlackBerry Smartphone" and Section 2.7.4, "How to Set Up a BlackBerry Smartphone Simulator").

To install the SQLite Mobile Sync client to a smartphone:

In JDeveloper, from the main menu select **Tools** > **Deploy ADF Mobile Client Runtime** > **to BlackBerry device**.

To install the SQLite Mobile Sync client to a simulator:

In JDeveloper, from the main menu select **Tools** > **Deploy ADF Mobile Client Runtime** > **to BlackBerry simulator**.

2.9 Setting Up the Fusion Order Demo Mobile Client Application

For information, see Chapter 3, "Introduction to the ADF Mobile Client Sample Application."

3

Introduction to the ADF Mobile Client Sample Application

The ADF Mobile client sample application, also referred to as the Fusion Order Demo Mobile Client application, is a modified version of the Fusion Order Demo. It demonstrates the extension of the Fusion application technology to mobile platforms. This application is a companion to this guide and is used as an example throughout to illustrate points and provide code examples.

Before you examine the individual components of the sample application in depth, you may find it helpful to install and become familiar with the functionality of the Fusion Order Demo Mobile Client application.

This chapter includes the following sections:

- Section 3.1, "About the Fusion Order Demo Mobile Client Application"
- Section 3.2, "Installing the Fusion Order Demo Schema"
- Section 3.3, "Overview of the Fusion Order Demo Mobile Client Application Schema."
- Section 3.4, "Running the Fusion Order Demo Mobile Client Application"
- Section 3.5, "Taking a Look at the Fusion Order Demo Mobile Client Application"

For information on the Oracle ADF Fusion Order Demo application, see "Introduction to the ADF Sample Application" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

3.1 About the Fusion Order Demo Mobile Client Application

The sample application enables you to browse and edit orders for electronic devices. For more information, see Section 3.5.2, "Runtime Components."

The sample application requires an existing installation of Oracle JDeveloper 11*g* and an Oracle database. For a complete list of tasks for preparing your environment for the Fusion Order Demo Mobile Client application, see Chapter 2, "Setting Up the ADF Mobile Client Environment."

3.2 Installing the Fusion Order Demo Schema

You can download the Fusion Order Demo application from the Oracle Technology Network (OTN) web site.

To download the demo and install the FOD schema to your database:

1. Navigate to Oracle Technology Network (OTN) `http://www.oracle.com/technetwork/index.html` and search for "Fusion Order Demo Sample Application."
2. Download the ZIP file to a local directory.
3. Start Oracle JDeveloper 11*g* and from the main menu choose **File** then **Open**.
4. In the Open dialog, browse to the location where you extracted the ZIP file to in Step 1 and select **Infrastructure.jws** from the **infrastructure** directory. Click **Open**.
5. In the Application Navigator, expand **MasterBuildScript** and then **Resources**, and double-click **build.properties**.
6. In the editor, modify the properties shown in Table 3–1 for your environment.

Table 3–1 Properties Required to Install the Fusion Order Demo Application

Property	Description
`jdeveloper.home`	The root directory where you have Oracle JDeveloper 11*g* installed. For example: `C:/JDeveloper/11/jdeveloper`
`jdbc.urlBase`	The base JDBC URL for your database in the format `jdbc:oracle:thin:@<`*`yourhostname`*`>`. For example: `jdbc:oracle:thin:@localhost`
`jdbc.port`	The port for your database. For example: `1521`
`jdbc.sid`	The SID of your database. For example: `ORCL`
`db.adminUser`	The administrative user for your database. For example: `system`
`db.demoUser.tablespace`	The table space name where FOD users will be installed. For example: `USERS`

7. From the JDeveloper main menu, choose **File > Save All**.
8. In the Application Navigator, under the **Resources** node, right-click **build.xml** and choose **Run Ant Target > buildAll**.
9. In the Enter Property dialog, enter the password for the database system user and click **Continue**.

   Once you enter the password, the Ant build script creates the FOD users and populates the tables in the FOD schema. In the Apache Ant - Log window, you will see a series of SQL scripts and finally:

   ```
   buildAll:

   BUILD SUCCESSFUL
   Total time: nn minutes nn seconds
   ```

   For more information on the demo schema and scripts, see the `README.txt` file in the `MasterBuildScript` project.

3.2.1 Mounting the Sample Client Database

In addition to installing the FOD database, you also need to install client databases for both Windows Mobile devices and BlackBerry smartphones. The sample client database is delivered on a simulated SD card that is included in the sample application ZIP file, `ADFMCSampleApp.zip`. The simulated cards are located within ADFMCSampleApp at `WindowsMobileSDCard` and `BlackBerrySDCard`.

3.2.1.1 How to Mount the Sample Client Database on Windows Mobile Devices

Use the Windows Mobile simulator to mount the simulated SD card, `WindowsMobileSDCard`, that contains the sample client database.

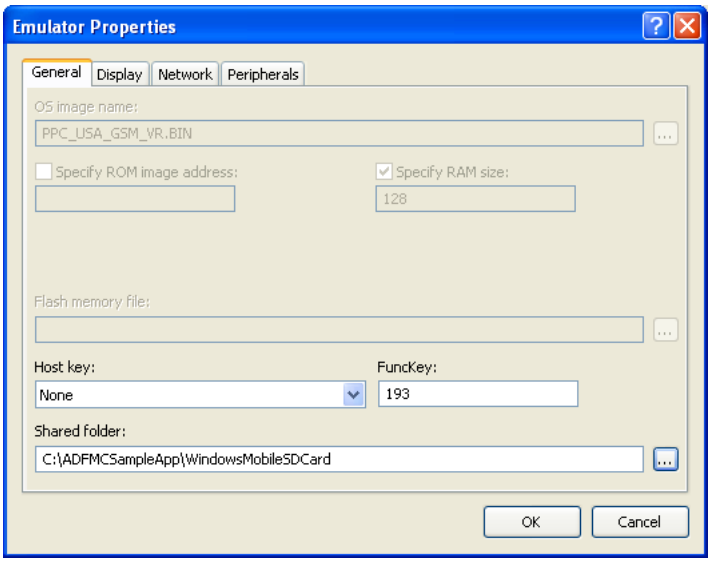
Before you begin:

Create the sample database that represents the database used by the server application as described in Section 3.2, "Installing the Fusion Order Demo Schema," download and extract `ADFMCSampleApp.zip` on your computer, and install the Windows Mobile device simulator. You must start and connect the Windows Mobile simulator to the computer as described in Chapter 2, "Setting Up the ADF Mobile Client Environment."

To mount the client database on a Windows Mobile device simulator:

1. On the Windows Mobile simulator, choose **File** > **Configure**.
2. In the Shared Folder folder, browse to WindowsMobileSDCard in ADFMCSampleApp, as shown in Figure 3–1.
3. Click **OK**.

Figure 3–1 Mounting the Simulated SD Card on a Windows Mobile Device Simulator



3.2.1.2 How to Mount the Sample Client Database on a BlackBerry Smartphone Simulator

Use the BlackBerry smartphone simulator to mount the sample client database that is contained on the simulated SD card, `BlackBerrySDCard`.

Before you begin:

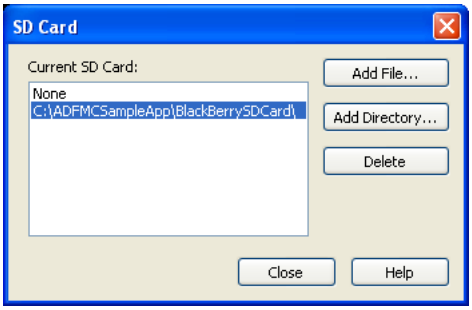
Create the sample database that represents the database used by the server application as described in Section 3.2, "Installing the Fusion Order Demo Schema," download and extract `ADFMCSampleApp.zip` on your computer, and install BlackBerry Java Development Environment (JDE 5.0).

Start the BlackBerry smartphone simulator to the computer as described in Chapter 2, "Setting Up the ADF Mobile Client Environment."

To mount the client database on a BlackBerry Smartphone simulator:

1. On the BlackBerry smartphone simulator, select **Simulate** > **Change SD Card**.
2. Click **Add Directory** and then browse to, and select, the **BlackBerrySDCard** folder. Click **OK**.
3. Select BlackBerrySDCard as the current SD card as shown in Figure 3–2. Click **Close**.

Figure 3–2 Mounting the Simulated SD Card to a BlackBerry Smartphone Simulator



3.3 Overview of the Fusion Order Demo Mobile Client Application Schema

The schema of the Fusion Order Demo Mobile Client application consists of the following core tables:

- `PERSONS`: This table stores all the users who interact with the system, including customers, staff, and suppliers. The first and last name, e-mail address, and person type code of each user is stored. A user is uniquely identified by an ID. Other IDs provide foreign keys to tables with address information and, membership information (for customers).
- `ORDERS`: This table represents activity by specific customers. When an order is created, the date of the order, the total amount of the order, the ID of the customer who created it, and the status of the order are all recorded. After the order is fulfilled, the order status and order shipped date are updated. All orders are uniquely identified by a sequence-assigned ID.
- `ORDER_ITEMS`: For each order, there may be many order items recorded. The unit price and quantity of each order item are recorded. The order line item and its order ID uniquely identify each order item.
- `PRODUCTS_BASE`: This table stores all of the products available in the store. For each product, the name and cost are recorded. All products are uniquely identified by a sequence-assigned ID. The image of the product and its description are stored in separate tables, which each reference the product ID.

Queue-based snapshots (described in "Manage Snapshots on SQLite Mobile Client" in *Oracle Database Lite SQLite Mobile Client Guide*) create a database file called `OSE_<database name>.db`. This database file contains the following tables:

- `OSE$DATAQ`: The data queue for both In Queue and Out Queue records. This queue is used for all snapshots and contains both In and Out Queue records. The TRID column is positive when the record is an Out Queue record. When you synchronize with queue-based snapshots enabled, new data from the client is uploaded from the `OSE$DATAQ` queue table and new data from the Oracle database is downloaded into this queue. For more information, see "SQLITE QUEUES" in *Oracle Database Lite SQLite Mobile Client Guide*.
- `OSE$BLOBQ`: A BLOB queue
- `OSE$TABLES`: A snapshot registry
- `OSE$TRANS`: A transactions registry
- `OSE$TRESQ`: A table that contains transaction sequences per publication.

In addition to these tables, SQLite replicates the following:

- `C$INDEXES`: A table containing replicated index information.
- `C$SCRIPTS`: A table containing SQL scripts published on the server.
- `C$WSEQ_CLIENTS`: Simulates sequences

3.4 Running the Fusion Order Demo Mobile Client Application

Before you run the demo application on a BlackBerry Smartphone or a Windows Mobile device, you must first do the following:

1. Open the demo application in JDeveloper.
2. Install simulators and other required development tools for the target platforms.
3. Connect and start the simulators.
4. Deploy the ADF Mobile client runtime components on BlackBerry smartphones and Windows Mobile devices.
5. Deploy the demo application to the simulators.
6. Run the demo application.

For more information on Oracle Lite Mobile Development Kit (MDK), Mobile Sync (mSync), and SQLite Mobile Client, see Chapter 2, "Setting Up the ADF Mobile Client Environment."

3.4.1 How to Run the Demo Application on a Windows Mobile Device Emulator

You run the demo application after you have deployed it to a Windows Mobile device emulator.

To run the demo application:

1. In JDeveloper, choose **Tools** the **Deploy ADF Mobile Client Runtime** then **to Windows Mobile**.
2. Choose **ADF Mobile Client Runtime** and then click **Deploy**.
3. Monitor the deployment progress on the Windows Mobile device emulator. Deployment must complete before you deploy another ADF Mobile client runtime component.

4. Choose **Java Virtual Machine** and then click **Deploy**.
5. Click **Close**. Verify the deployment by viewing JDeveloper's Deployment-Log file.
6. In the Application Navigator, right-click the Mobile Client project.
7. Choose **Deploy** then choose **ADFMCSampleApp_RC1 to Windows Mobile Device**.
8. On the Windows Mobile simulator, use the File Explorer utility to navigate to **My Device** then **Program Files** and then **MobileFOD**.
9. Select the **MobileFOD** executable file, as shown in Figure 3–3.

Figure 3–3 Selecting the MobileFOD Executable File

Figure 3–4 shows the Fusion Order Demo Mobile Client application's Browse page, which appears after you click the MOBILEFOD executable. For more information, see Section 3.5, "Taking a Look at the Fusion Order Demo Mobile Client Application."

Figure 3–4 The Browse Order Page

3.4.2 Running the Sample Application on a BlackBerry Smartphone Simulator

Before you run the sample application on a BlackBerry Smartphone or simulator, you must download the BlackBerry Java Development Environment (JDE 5.0), the BlackBerry smartphone simulator, and BlackBerry Desktop Manager from BlackBerry (`http://na.blackberry.com`).

3.4.2.1 How to Start the Demo Application on a BlackBerry Smartphone Simulator

You run the demo application after you have deployed it to a BlackBerry smartphone or simulator.

Before you begin:

You must set the location of the JDE and the simulator directory in which JDeveloper places the COD file.

To run the demo application:

1. If needed, stop the BlackBerry smartphone simulator.
2. In JDeveloper, choose **Tools** then **Deploy ADF Mobile Client Runtime** then choose **to BlackBerry Simulator**.
3. Choose **ADF Mobile Client Runtime** and then click **Deploy**.
4. In the Application Navigator, right-click the mobile client project.
5. Choose **Deploy** then **ADFMCSampleApp_BlackBerry to BlackBerry Device**.
6. After the deployment completes, start the BlackBerry smartphone simulator.
7. On the BlackBerry smartphone or simulator, navigate to the **Downloads** folder, as shown in Figure 3–5.

8. Click **ADFMCSampleApp_BlackBerry**.

> **Note:** Do not deploy an application to BlackBerry smartphone simulator while it is running. To load the COD file to the simulator, you must first stop it, then restart it. If you would like to deploy a COD file manually, put it in the simulator location specified in the Mobile Client Preferences page described in Chapter 2, "Setting Up the ADF Mobile Client Environment."

Figure 3–5 The BlackBerry Smartphone Downloads Folder

3.5 Taking a Look at the Fusion Order Demo Mobile Client Application

The design time artifacts of the ADF Mobile sample application are visible in Oracle JDeveloper. Its runtime UI consists of screens that are displayed natively on the device outside of a mobile browser.

3.5.1 Design Time Components

After you have opened the projects in Oracle JDeveloper, you can then review the artifacts within each project. The development environment for the Fusion Order Demo Mobile Client application contains a view-controller project named MobileClient and a model project named Model.

3.5.1.1 MobileClient Project

MobileClient project contains the files for the interface, including the backing beans, deployment files, and MCX files (the ADF Mobile client equivalent of the JSPX file).

Figure 3–6 The MobileClient Project

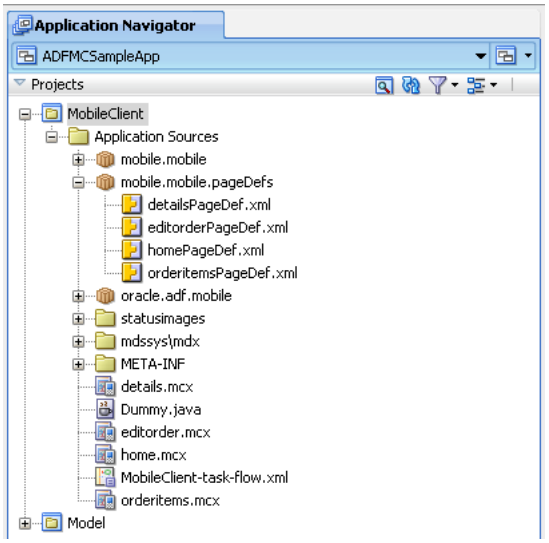


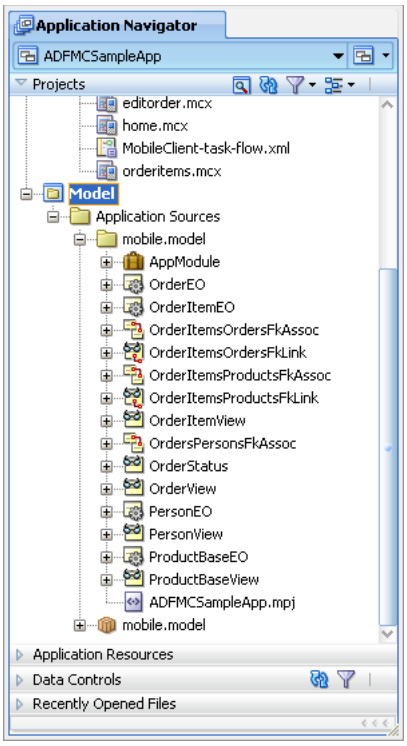
Figure 3–6 shows the MobileClient project and its associated directories. The MobileClient project contains the following directories:

- `Application Sources`: Contains the code used by the mobile client, including the managed and backing beans, property files used for internationalization, the metadata used by ADF Mobile client to display bound data as well as the MCX files and images. This project also includes `MobileClient-task-flow.xml`, the source file for the bounded task flow.
- `META-INF` (subdirectory of `Application Sources`): includes the following:
  - `UiApplication.jad`—the Java Application Descriptor file (JAD) that is required by BlackBerry to deploy applications over the air (OTA).
  - `adfm.xml`—Used at runtime in order to help resolve data bindings.

3.5.1.2 Model Project

The Model project (shown in Figure 3–7) is a data model project that contains the data definitions that allow the product data to be displayed in the sample application.

Figure 3–7 The Model Project



The Model project contains the Application Sources directory, which contains the files used to access the product data. Included are the metadata files used by Oracle Application Development Framework (ADF) to bind the data to the view.

3.5.2 Runtime Components

The Fusion Order Demo Mobile Client application displays the following pages at runtime:

- Browse Orders (`home.mcx`)
- Order Details (`details.mcx`)
- Edit Order (`editorder.mcx`)
- Order Items (`orderitems.mcx`)

3.5.3 Browsing Orders

You start the Fusion Order Demo Mobile Client application by running the `home.mcx` page. This page, which is the entry point of the application, displays on a BlackBerry smartphone or Windows Mobile device as the Browse Orders page. As shown in Figure 3–8, this page contains a dropdown list of order status codes. You use this list in conjunction with the page's **Search** button to retrieve a list of customers whose orders belong to the category that you selected. Clicking a customer name opens the Order Details page (shown in Figure 3–12), which displays information about the customer.

Clicking **Add Order** displays the Add Order page, which is comprised mainly of `inputText` components. For more information about the Order Details page, see Section 3.5.4, "Viewing Order Details." For more information on the Add Order page, see Section 3.5.5, "Editing or Adding an Order."

Figure 3–8 The Browse Order Page

After you select an order status and click **Search** (a `commandButton` component), the application performs a search, executes a query and populates the table. Figure 3–9 shows the results of selecting the *Order being picked* status. As illustrated in Figure 3–9, the table is populated with a status image, the customer name, and read-only information about the date of the order and its total cost. The name is a `commandLink` component that enables you to navigate to the details screen.

Figure 3–9 Records Retrieved by Order Status

The images are dynamic, reflecting the status of the order. Note, for example, that the images reflect the *Order being picked* status in Figure 3–9 and the *Order completed* status in Figure 3–10.

Figure 3–10 Dynamic Images

Where to Find Implementation Details

Following are the sections of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* and *Oracle Fusion Middleware Mobile Client Developer's Guide for Oracle Application Development Framework* that describe how to define queries and create query search forms:

- Laying out ADF Mobile Tables

  The Browse Orders page includes a single-column table which uses `panelGroupLayout` components for the status images and the output text. For more information, see Section 6.8, "Creating and Using Tables."

- Enable navigation through the application

  The demo application provides navigation using `commandLink` components, In Figure 3–10, **Add Order** and the customer names (for example, Nancy Greenberg) are `commandLink` components. For more information, see Section 6.9, "Using Buttons and Links."

- Creating dynamic images

  The page achieves the dynamic, status-related images through an EL expression and a string. For example,

  ```
  <amc:image id="image1"source="/statusimages/#{row.OrderStatusCode}.gif"/>
  ```

  For more information see "Creating EL Expressions" in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework* and also "Displaying Images."

- Creating a static view object

  In the Browse Orders page, the `selectOneChoice` component for the order status is backed by a list binding. The Status and StatusDesc attributes of the OrderStatus view object, a static view object, populate the values for the dropdown list. For more information, see "Populating View Object Rows with Static Data" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- Defining a list of values for selection lists

  Input forms displayed in the user interface can utilize databound ADF Faces selection components to display a list of values (LOV) for individual attributes of the data collection. To facilitate this common design task, Oracle ADF Business Components provides declarative support to specify the LOV usage for attributes in the data model project. For example, in the Fusion Order Demo application, the `selectOneChoice` component displayed in the Browse Orders page is bound to LOV-enabled attributes configured for the OrderStatus view object. For more information about configuring attributes for LOV usage, see "Working with Lists of Values (LOV) in View Object Attribtutes" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- Creating menus for Windows Mobile devices and BlackBerry smartphones

  Windows Mobile applications have two types of menus: The Main menu and the ALT menu. The Main menu describes the right Standard menu button for the form (such as the Menu button in Figure 3–10) and the ALT menu is the left Standard menu button for the form. For BlackBerry, the MAIN menu describes the Full menu for the form. The ALT menu is amalgamated into the Full menu for the form. For more information, see Section 6.11, "Creating and Using Menus."

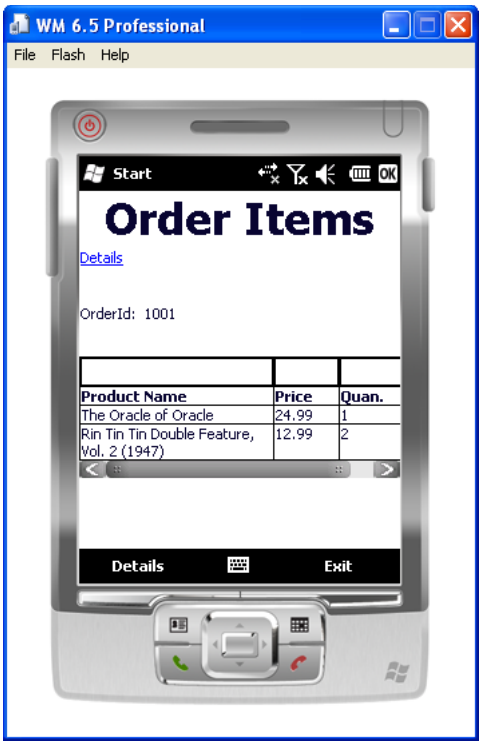
3.5.4 Viewing Order Details

The Order Details page displays information about the customer who placed the order.

Figure 3–11 The Order Details Page

This page contains two navigational links: **Back to Order List**, which you use to return to the result list shown in Figure 3–10 and **Show Order Items**. Clicking **Edit** enables you to edit the order details. For more information, see Section 3.5.5, "Editing or Adding an Order."

Figure 3–12 The Order Items Page

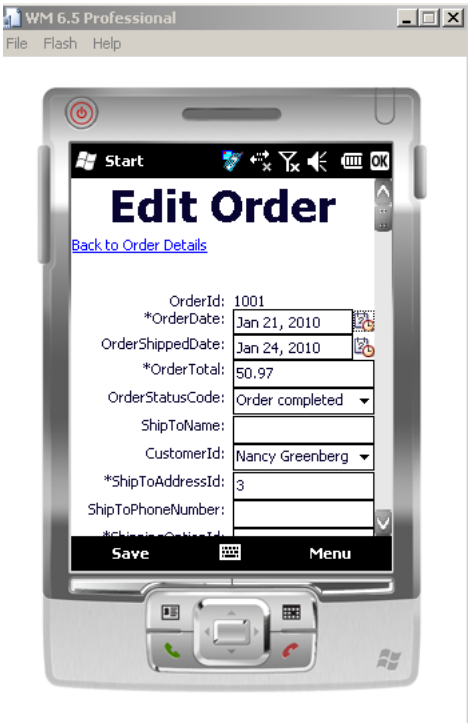


Clicking **Show Order Items** enables you to drill down to a list of ordered items that match the status that you selected in the Browse Orders page. For example, Figure 3–12 shows customer Nancy Greenberg's items that have the status of *Order completed* (shown in Figure 3–10). This page also includes a form that enables you to filter order items. For more information, see Section 3.5.6, "Viewing Ordered Items."

3.5.5 Editing or Adding an Order

Clicking **Edit** opens a read-write form that enables you to modify order details. This page (`editorder.mcx`) displays as either the Edit Order page or the Add Order page depending on if the user is at the Browse Page (`home.mcx`) or the Order Details page (`details.mcx`).

Figure 3–13 The Edit Order Page



Where to Find Implementation Details

Following are sections of the *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* and *Oracle Fusion Middleware Mobile Client Developer's Guide for Oracle Application Development Framework* that describe how to create a task flow and how to create an input form with required values.

- Laying out the ADF Mobile page

  This page, called `editorder.mcx`, consists primarily of `inputText` components within `panelFormLayout` and `panelLabelAndMessage` components. For more information, see Section 6.4, "Designing the Layout of the Page."

- Grouping activities using a bounded task flow

  ADF Mobile client applications use bounded task flows. As described in "Task Flow Types" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*, a bounded task flow has one point of entry, zero or more exits, and represents the reusable portion of an application. The task flow for the demo application (`MobileClient-task-flow.xml`) is comprised of view activities (`home.mcx`, `editorder.mcx`, `details.mcx`, and `orderitems.mcx`), a router activity, a task flow return, and a wildcard control rule. For more information see Section 6.2, "Creating Task Flows."

- Using a router activity

  The conditional routing that enables `editorder.mcx` page to display as either the Edit Order page or the Add Order page is achieved through a router in the application's task flow that evaluates the `applicationScope` EL expression for the `panelGroupLayout` component. If the scope variable for `addMode` is true (that is, when the user is at the Browse Order page), then the application opens the Add Order page. For more information, see "Using Router Activities" in *Oracle*

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

- Creating a databound edit form

  When you want to create a basic form that collects values from the user, instead of having to drop individual attributes, JDeveloper allows you to drop all attributes for an object at once as an input form. You can create forms that display values, forms that allow users to edit values, and forms that collect values. For example, in the Fusion Order Demo application, the Order Details displays a form that displays user information and the Add Order and Edit Order forms used for collecting shipping information for the user's order. For more information, see "Creating an Input Form" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework* and Section 6.5, "Creating and Using Input Components."

- Creating databound UI controls

  The page's `selectOneChoice` components are used to filter view accessors and lists of values for status codes and for customer ID. Such `selectOneChoice` components are created by dragging and dropping the OrderStatus Code and CustomerID attributes of the OrderView data control into the Structure view and then by selecting **Select One Choice** from the context menu.

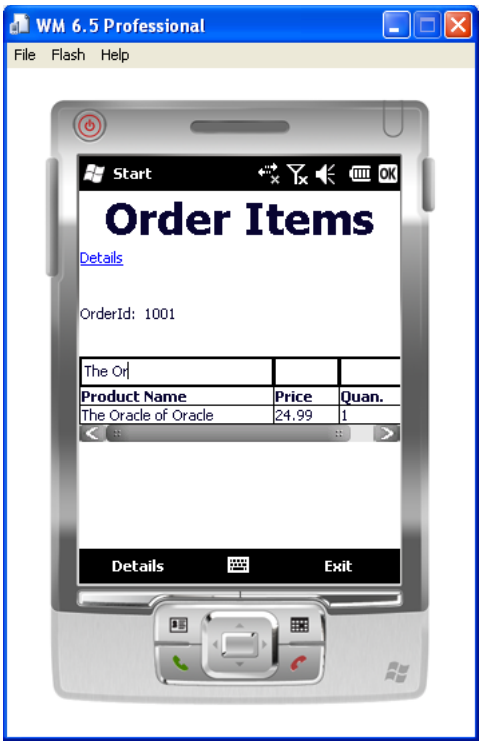
- Using commit and rollback functions

  The page's **Save** and **Undo** menu functions are standard ADF commit and rollback operations. For more information, see Section 6.11, "Creating and Using Menus."

3.5.6 Viewing Ordered Items

The Order Items page lists the items that the customer has ordered, providing information on the number of units and the price of each item. You can filter the results using the search form, as shown in Figure 3–14.

Figure 3–14 The Order Items Page



Where to Find Implementation Details

- Creating a search form

  You create a query search form by dropping a named view criteria item from the Data Controls panel onto a page. The Order Items page includes a search form, one that JDeveloper creates automatically when you drag a view object in to the Structure view and then select **Filtering** as the Enable ADF Behavior option in the Edit Table Columns dialog. For more information, see "Creating Query Search Forms" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

- Displaying the results of a query search

  Normally, you drop a query search panel with the results table. JDeveloper automatically wires the results table with the query panel. For more information, see "How to Create a Query Search Form and Add a Results Component Later" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

4

Getting Started with ADF Mobile Client

This chapter describes how to use JDeveloper to declaratively create ADF Mobile client applications.

This chapter includes the following sections:

4.1 About Declarative Development with JDeveloper

You can write, deploy, and test an ADF Mobile client application without writing a line of code because the JDeveloper design experience is enhanced to include support of mobile client application development. JDeveloper provides a series of wizards that step you through creating the mobile client application, creating its model and view projects, and creating business objects. In addition, it generates the required artifacts to define page flows. You can then define the views of the project using the drag and drop functionality of the Data Controls panel and the Component Palette.

4.2 About Developing an ADF Mobile Client Application

To develop an ADF Mobile client application:

1. Determine the mobile data requirements: While mobile client applications are extensions of ADF applications that run on a server, users will not use the handheld version of this application in the same way as they would use it on a desktop. Because mobile applications should not mimic their server-side counterparts in terms of presentation or user interaction, consider the tasks that users perform with the mobile application and the circumstances under which they use the mobile application. For example, would the user interact with the mobile application for minutes, as opposed to hours, as they would on a desktop application?

   Tip: To encourage usage, streamline the application as much as possible. For example, consider peripherals, such as barcode readers, cameras, or scanners.

2. Create a subset of the server data model for the ADF Mobile client application: Although the mobile applications themselves are not merely handheld copies of server-side applications, the underlying data that they manipulate is a subset of server data. Because the tasks enabled by a mobile client application have server-side data representation as well as server-side validation, begin creating a subset of the data by examining the server data model and identify a subset of those entity objects that should be available to the mobile client application. As described in Section 4.5, "Extending the Base Application for the Mobile Client Application," you use the Create Business Components for ADF Mobile Client wizard to identify the tables from the base application from which you want to create entity objects and the columns that you want to expose. This wizard also enables you to create view objects. For more information, see Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects." For more information on entity objects and view objects, see Section 5.1.1, "Support for the Core ADF Business Components."
3. Edit the entity objects as needed: If needed, you can expose additional attributes to the user interface or change certain properties of the attributes that already belong to entity objects. For more information, see Section 5.3, "Editing Mobile Entity Objects."
4. Test the mobile data model by running the mobile Application Module with the Business Component Browser. For more information, see "Using the Business Component Browser for Testing and Debugging" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
5. Create or configure custom ADF Mobile transaction replay items.
6. Define the mobile client task flow: You begin designing the UI of the mobile application by creating an overall page flow and then by populating the task flow with views (pages) and control rules.
7. Create the mobile client views: After you create the task flow, you can create the UI by populating the pages with ADF components and data controls. ADF Mobile client supports components for page layout, input and output mechanisms, image display, and page navigation. For more information, see Chapter 6, "Creating the ADF Mobile Client User Interface."
8. Define menus: ADF Mobile client enables you to create platform-specific menus for both BlackBerry smartphones and Windows Mobile devices. For more information, see Section 6.11, "Creating and Using Menus."

4.3 Deploying the Model Project of the Base Application as an ADF Library

Once you have created (or located) an ADF application on which to base the mobile client application, you then deploy its model project as an ADF library as described in "Packaging a Reusable ADF Component into an ADF Library" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. You draw from this library using the Business Components from ADF Library wizard to create the subsets of entity objects and view objects used by the ADF Mobile application. For more information, see Section 5.2, "Extending an ADF Application to Mobile Client."

4.3.1 How to Deploy the Model Project

You create the ADF Library for the mobile application by packaging the model project of the base application as a JAR file.

Before you begin:

Obtain a JAR file that contains the ADF library that will be used to create the business components for the ADF Mobile application or create an application that contains these business components using the Create Business Components from Tables Wizard as described in "Creating a Business Domain Layer Using Entity Objects" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To Deploy the model project as a JAR file:

1. In the Application Navigator, right click the model project.
2. In the Project Properties dialog's left pane, select **Deployment** and then click **New**.
3. In the Create Deployment Profile dialog, select **ADF Library JAR file** for archive type, enter a name for the deployment profile, and click **OK**.
4. Verify the default directory path or enter a new path to store your ADF Library JAR file. Click **OK**.
5. In the Application Navigator, right-click the model project and choose **Deployment**.
6. Select the deployment profile from the context menu.
7. Select **Deploy to ADF Library JAR File** in the Deployment Action page.
8. Click **Next**. Review the Summary page, which notes the output location for the JAR file. This location is set in the JAR Options page of the Edit JAR Deployment Profile Properties dialog.
9. Click **Finish**.

4.4 Creating an Application Workspace

The first steps in building an ADF Mobile client application are to assign it a name and to specify a directory where its source files will be saved. By creating an application using the application templates provided by JDeveloper, you automatically get the organization of your workspace into projects, along with many of the configuration files required by the type of application that you are creating.

4.4.1 How to Create an Application Workspace

You create an application workspace using the Create Mobile Client Application (ADF) Wizard.

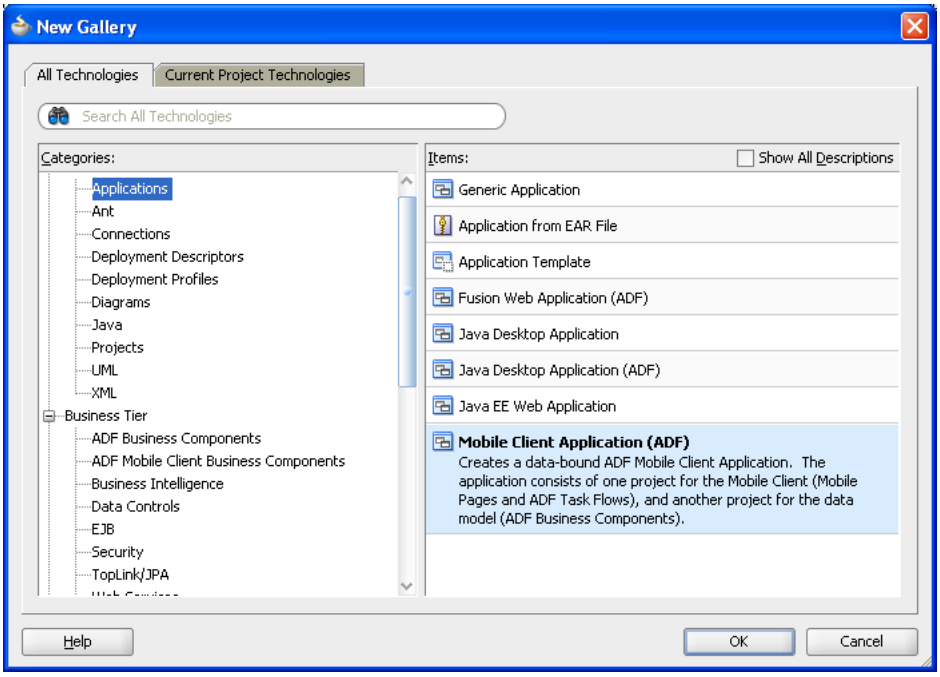
Before you begin:

The model project containing the entity objects and view objects on which you base the ADF Mobile client application must be deployed as a JAR file on the development computer, which can subsequently be imported in an ADF Mobile client application. For information on creating an ADF library JAR, see "Packaging a Reusable ADF Component into an ADF Library" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. See also Section 4.3, "Deploying the Model Project of the Base Application as an ADF Library."

To create a workspace:

1. In the main menu, choose **File > New**.
2. In the New Gallery, expand **General**, select **Applications** and then **Mobile Client Application (ADF)** and then click **OK**.

Figure 4–1 Selecting the Application Type

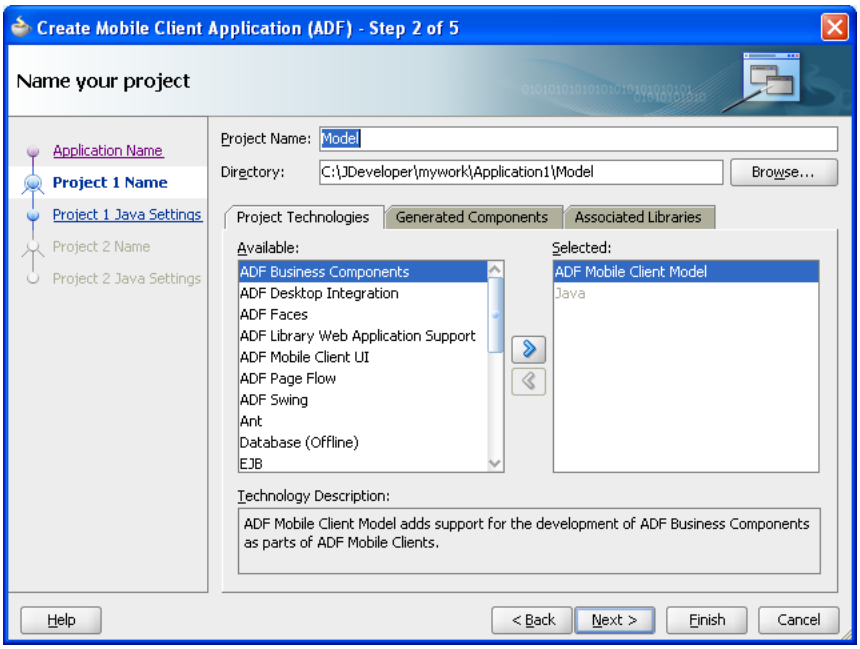


3. In the Name your application page, enter a name for the application and if needed, change the directory name and application prefix and then click **Next**.
4. In the Name your project page, change the name and location of the project, if needed. Click **Next**.

> **Note:** This step is optional.

Figure 4–2 shows the ADF Mobile Client Model and Java technologies, which are selected by default. You must select these technologies if they do not appear in the **Selected** list. You must add the **ADF Mobile Client Model** technology to a model project if it is not already present.

Figure 4–2 Selecting the Project Technologies



5. If needed, change the Java settings for the model project. Click **Next**.

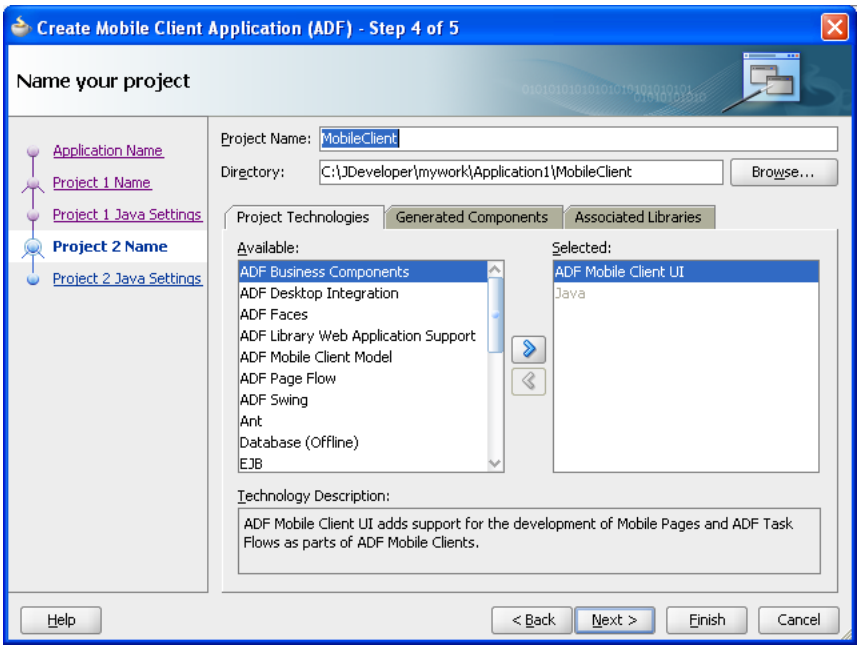
> **Note:** This step is optional.

6. In the Name your project page, enter a name, if needed, for the view controller project of the application. JDeveloper selects the **ADF Mobile Client UI** technology by default, which enables you to create task flows. Click **Next**.

> **Note:** This step is optional.

Figure 4–3 shows the **ADF Mobile Client UI** technology selected by default. To create a task flow and define application pages, you must select this technology if it is not already selected.

Figure 4–3 Creating the Mobile Client View Project



7. If needed, change the Java settings for the view project.

> **Note:** This step is optional.

8. Click **Finish**.

4.4.2 What Happens When You Create a Mobile Client Application Workspace

When you complete the entire Create Mobile Client Application wizard, JDeveloper creates two projects: a model project (which bears the default name, *Model*, as shown in Figure 4–4) and a view controller project. Table 4–1 lists the JDeveloper-generated files that are contained in this project, which has the default name, *MobileClient*.

Table 4–1 Artifacts of Mobile Client View Projects

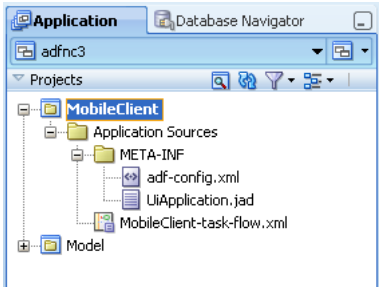
Artifact	Location	Description
`MobileClient-task-flow.xml`	Application Sources	The source file for the mobile client task flow. By default, this is a bounded task flow. This file is created when you select the Mobile Client UI technology for the project. This is the mobile client equivalent to the `adfc-config.xml` or `faces-config.xml` in an ADF Faces application.

Table 4–1 (Cont.) Artifacts of Mobile Client View Projects

Artifact	Location	Description
`UiApplication.jad`	META-INF	This JAD (Java Application Descriptor) file is required by BlackBerry to deploy applications. When you deploy a BlackBerry application, note that the BlackBerry Options page of the BlackBerry Deployment Profile Properties dialog contains entries written to this file before it is used to compile the application.
`adf-config.xml`	<appDir>/.adf/META-INF	A standard configuration file that specifies application-level settings that are usually determined at deployment and are read-only at runtime. ADF Mobile client uses specific key values in this file. This file has the same format as that used by ADF Faces. For information on `adf-config.xml`, see "adf-config.xml" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Figure 4–4 shows the MobileClient project in the Application Navigator, containing the files described in Table 4–1.

Figure 4–4 The MobileClient Project



4.5 Extending the Base Application for the Mobile Client Application

Extending an ADF application that resides on a server for ADF Mobile client is usually the starting point in developing an application. Conceptually, the mobile client application is always an extension of an existing ADF application (referred to as a base application). Because of this, the entity objects of the mobile client application are always copies of the base application's entity objects. Entity objects define the data that is available to the ADF Mobile client application. View objects are based on the entity objects.

Not only are the entity objects of a mobile client application copies, they are also a subset of the entity objects used by the base application. While a base application may have 20 entity objects, a mobile application may require only five of them. Further, the mobile client application may only require a subset of the attributes owned by the selected entity objects.

You create these data subsets using the ADF Mobile Client Business Components from Entity Objects wizard. This wizard, described in Section 5.2, "Extending an ADF Application to Mobile Client," steps you through creating the entity objects and view objects.

Because you derive a mobile client application from a base (server) ADF application, you should assess which entity objects and view objects in the base application should be used in its mobile counterpart before you develop the mobile client application.

Note: Read-only view objects that are based on SQL queries have limitations in Oracle Fusion Middleware 11*g* release 1 of ADF Mobile client. For example:

- A SQL query-based, read-only view object contains all of the columns of a table, while a mobile client application typically synchronizes only a subset of these columns.
- Although the mobile client ADF business components represent the client database schema, JDeveloper points to the database on the server rather than to the client database when determining which attributes to create for a SQL query-based, read-only view object. As a result, JDeveloper adds all of the columns on the server database to the view object, regardless of whether they exist on the client database.

If you create a read-only view object based on a SQL statement, you must:

1. Ensure that the SQL query used to create the view object can run on the device. For information on the SQL query syntax, refer to `http://www.sqlite.org`
2. Manually change the view object's attributes to match those of the database columns on the client database.

5

Developing the ADF Mobile Client Data Model

This chapter provides an overview of developing the ADF Mobile client model layer.

This chapter includes the following sections:

5.1 Building Business Services for ADF Mobile Client Applications

Mobile client utilizes ADF Business Components, which simplify building business services. By eliminating the substantial coding and testing related to common application development. ADF Business Components enable you to instead focus on implementing business solutions. ADF Business Components provide a foundation of Java classes that business-tier application components extend to leverage a robust implementation of the numerous design patterns you need in many areas. The benefits of ADF Business Components include:

- Simplified Data Access
- Enforcement of Business Domain Validation and Business Logic
- Support for Sophisticated User Interfaces with Multi-Page Units of Work
- Implementation of Best Practices, High-Performance, Service Oriented Architecture

Simplified Data Access

ADF Business Components enable you to do the following:

- Design a data model for client displays, including only necessary data

- Include master-detail hierarchies of any complexity as part of the data model
- Implement end-user Query-by-Example data filtering without code
- Automatically coordinate data model changes with business domain object layer
- Easily validates and saves any changes to the database

Enforcement of Business Domain Validation and Business Logic

- Declaratively enforce required fields, primary key uniqueness, data precision/scale, and foreign key references
- Easily capture and enforce both simple and complex business rules, programmatically or declaratively, with multilevel validation support
- Navigate relationships between business domain objects and enforce constraints related to compound components

Support for Sophisticated User Interfaces with Multi-Page Units of Work

- Reflect changes made by business service application logic in the user interface
- Retrieve reference information from related tables, and automatically maintain the information when user changes foreign-key values
- Simplify multistep, web-based business transactions with automatic web-tier state management
- Handle images with no code
- Synchronize pending data changes across multiple views of data
- Consistently apply prompts, format masks, and error messages in any application (these may be entity-based or a static list)
- Define custom metadata for any business components to support metadata-driven user interface or application functionality
- Add dynamic attributes at runtime to simplify per-row state management

Implementation of Best Practices, High-Performance, Service Oriented Architecture

- Enforce best-practice, interface-based programming style
- Reduce network traffic for remote clients through efficient batch operations

5.1.1 Support for the Core ADF Business Components

As the mobile implementation of the Oracle ADF architecture, ADF Mobile client utilizes the core ADF Business Components but with some modifications:

- entity object

  An entity object represents a row in a database table. It simplifies modifying the row's data by handling all data manipulation language (DML) operations for you. An entity object encapsulates business logic for the row to ensure your business rules are consistently enforced. You associate an entity object with others to reflect relationships in the underlying database schema to create a layer of business domain objects to reuse in multiple applications. Association objects define relationships between entity objects.

- view object

View objects provide the means to retrieve data from a data source. In most cases, the data source will be a database and the mechanism to retrieve data is the SQL query. The view object simplifies working with the results of the SQL query. A data model project can include the following types of view objects:

- Read-only view objects when updates to data are not necessary (may be entity-based)
- Entity-based view objects when data updates will be performed

ADF Mobile client applications generally use entity-based view objects. These rows are not populated until run time.

- application module

  An application module is the transactional component that UI clients use to work with application data. It defines an updatable data model and top-level procedures and functions (called service methods) related to a logical unit of work related to an end-user task. Unlike ADF Faces applications, ADF Mobile client supports only a single, globally accessible application module. For more information, see "Getting Started with ADF Business Components" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

  Because the application module component supports a "UI- aware" data model of row sets, you do not need to write additional code for typical Create, Update, and Delete operations. By declaratively binding such client UI components to active view object instances in the application module's data model, the components in an MCX page automatically update to reflect changes in the rows of the view object row sets of the data model. For more information, see "Overview of the UI-Aware Data Model" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

5.1.2 Support for Mobile Database Transactions

The ADF Mobile client framework does the following after the user saves changes made to the data using an ADF Mobile client application:

1. `DBTransaction.commit()` is invoked, which in turn generates appropriate DML statements for all changes to the data model.
2. As each statement is posted to the database, an event is fired, which in turn generates an XML record—an entity Replay Item—which encapsulates the details of the `INSERT`, `UPDATE`, or `DELETE` statement.
3. After the transaction is committed, another event is fired, which causes the previously generated XML records to be written to the database in a separate transaction.
4. At some later point, data synchronization is initiated, either on-demand or in the background, and Oracle Database Lite Mobile Server (Mobile Server) transfers the Replay Item(s) up to the Mobile Server instance.
5. If data satisfies all necessary criteria, then it is committed to the enterprise database. Otherwise, an error is recorded in Mobile Server and then delivered to the mobile client the next time synchronization occurs.

5.2 Extending an ADF Application to Mobile Client

As described in Section 4.5, "Extending the Base Application for the Mobile Client Application," the mobile client application is an extension of the server, or base application. The entity and view objects, as well as their attributes, are subsets of the objects owned by the base application. To enable you to create a subset of entity objects (as well as the attributes of these entity objects) and view objects that are required by the mobile client projects, mobile client provides the Create Business Components for ADF Mobile Client wizard, which derives objects from the ADF library JAR created from the base application. For more information about creating an ADF library JAR from the base application, see Section 4.3, "Deploying the Model Project of the Base Application as an ADF Library."

5.2.1 How to Create Subsets of Entity Objects and View Objects

The Create Business Components from ADF Library wizard provides a multi-step approach to adding business components to the ADF Mobile client application. First, you point to the ADF library JAR file that was previously exported from the base ADF application. The ADF library JAR enables you to reuse the ADF Business Components defined in the base application. Next, you select the entity objects and entity attributes that are required for the mobile client application. Lastly, you create updatable view objects based on the previously selected entity objects and add them to a new application module.

> **Tip:** You can add transient attributes to entity objects that do not persist to columns defined in the underlying table. Any manually added attributes on mobile client entity objects are considered transient attributes.

Before you begin:

As described in Section 4.4.1, "How to Create an Application Workspace," you must have an ADF library JAR of the model project containing the entity objects and view objects on which you base the ADF Mobile client application. For information on creating an ADF library JAR, see "Packaging a Reusable ADF Component into an ADF Library" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

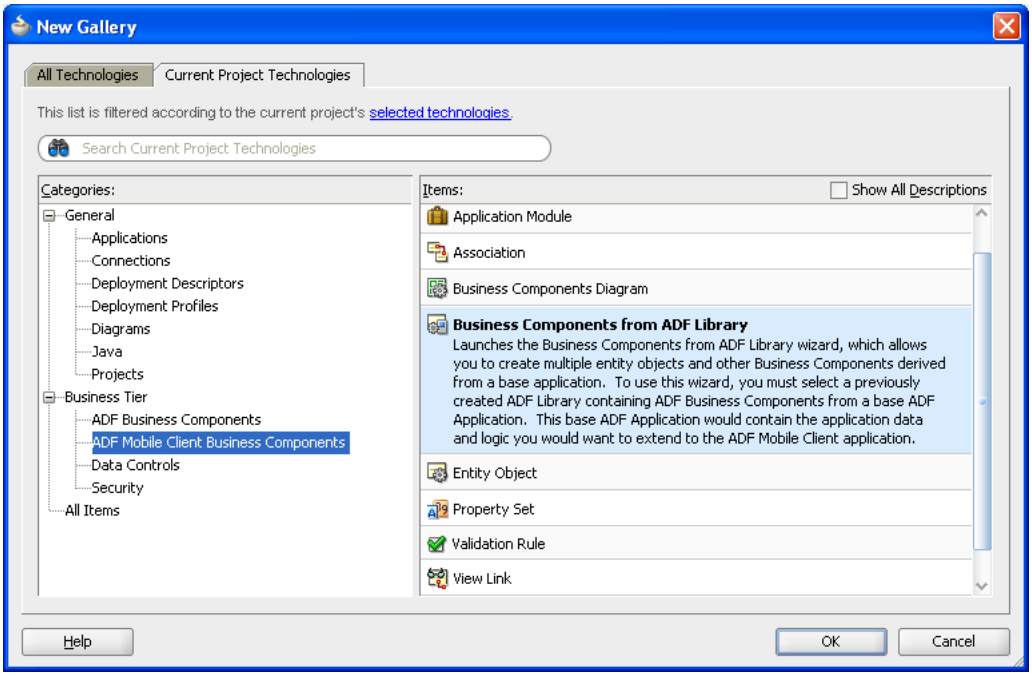
To create subsets of entity objects:

1. In the Application Navigator, right-click the model project you created in Section 4.4.1, "How to Create an Application Workspace," and choose **New**.
2. In the New Gallery, expand Business Tier, select **ADF Mobile Client Business Components** and then **Business Components from ADF Library**. Click **OK**.

   If this is the first component that you are creating in the project, the Initialize Business Components Project dialog appears to allow you to select a database connection to the server (base) application. You use this application to develop the business components for an ADF Mobile client application.

Note: ADF Mobile client only supports SQL92 as the SQL flavor for the database connection.

Figure 5–1 Selecting the Business Components from ADF Library Wizard



3. In the Initialize Business Components Project dialog, select the database connection or choose **New** to create a connection. Click **OK**.

> **Note:** You must use the same connection used by the base (server) application. Do not use the connection to the Oracle database schema where Mobile Server artifacts are stored.

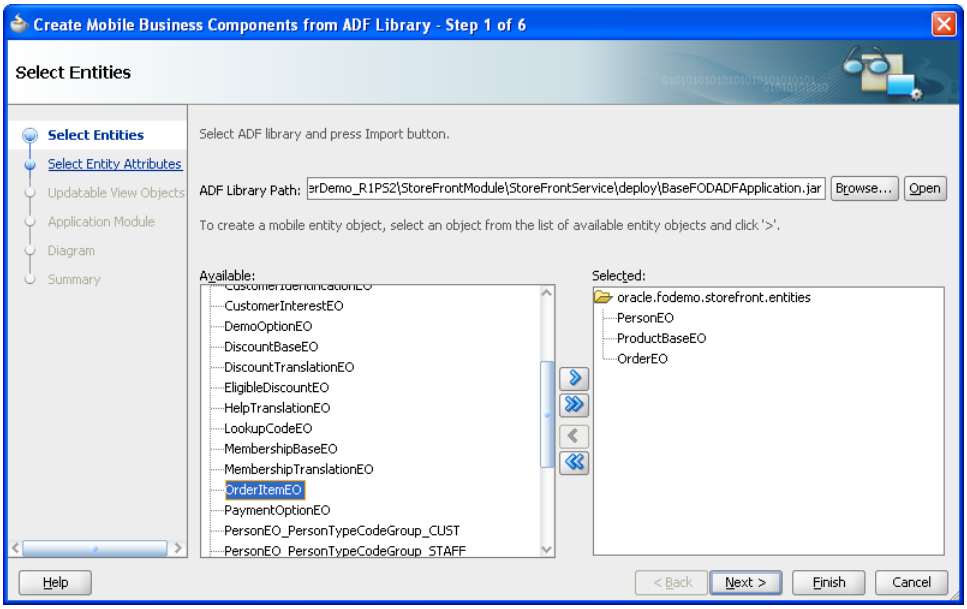
4. After JDeveloper establishes the connection, Business Components from ADF Library wizard appears. In the Select Entities page, enter the location (or browse for) the ADF library JAR created from the base application, and click **Open**.

> **Note:** Because the location of the ADF Library JAR used to create an ADF Mobile client application is hard-coded in the model project, you must update the JAR's location for the `NMCBaseLibraryFilePath` property in `Model.jpx` if you move the mobile application to another computer or change or delete the JAR. The application will not run unless you change this value.

For example, Figure 5–2 shows the result of shuttling the ProductBaseEO, OrderEO, and ProductBaseEO entity objects into the **Selected** list from the Available list, which includes all of the entity objects.

5. Shuttle the entity objects that you want to base the mobile entity objects on to the **Selected** list. Click **Next**.

Figure 5–2 Selecting Entity Objects for the Mobile Client Application

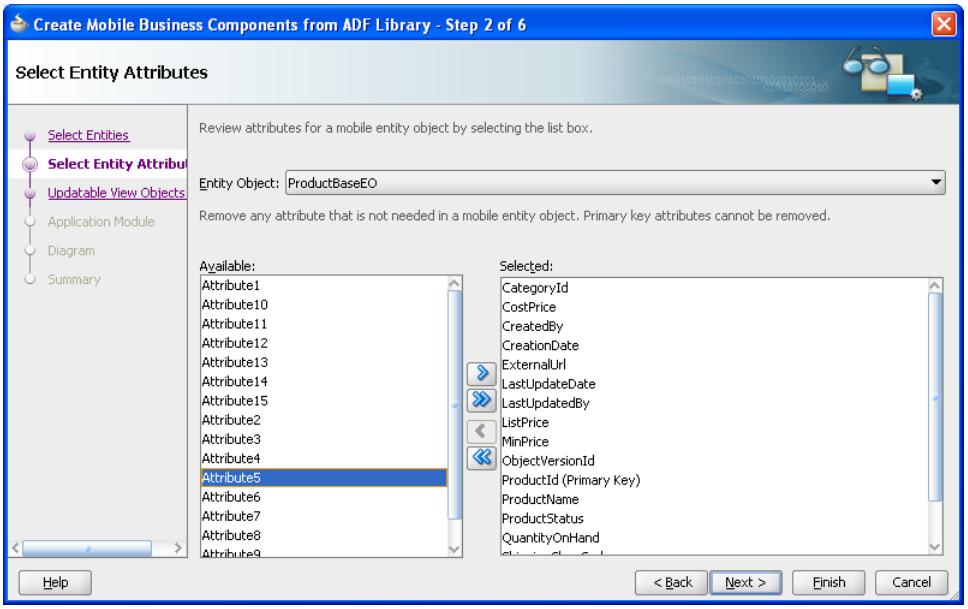


6. In the Select Entity Attributes page, select the attributes that you want to include from each entity usage in the **Selected** list. Shuttle the ones that you do not want to the **Available** list. Click **Next**.

> **Note:** The attributes that appear by default in the Available window are calculated fields that do not have a mapping in the database table definition. You cannot move any attribute that is a primary key from the **Selected** list.

For example, Figure 5–3 shows the attributes have been selected from the ProductBaseEO entity object.

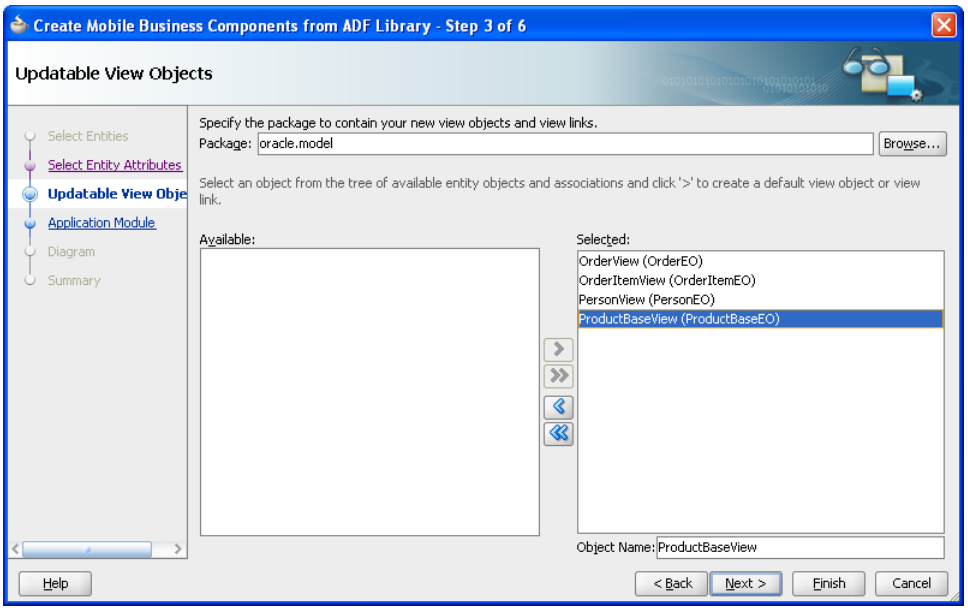
Figure 5–3 Selecting Attributes for the Entity Objects of the Mobile Client Application



7. In the Updatable View Objects page, select the entity objects to create a default view object. Click **Next**.

   For example, Figure 5–4 shows that OrderEO and OrderItemEO have been selected to create the view objects OrderEOView and OrderItemEOView.

Figure 5–4 Selecting View Objects



8. In the Application Module page, select **Application Module** and then enter a name for the application module. For more information on application modules, see "Getting Started with ADF Business Components" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. Click **Next**.

9. In the Summary page, click **Finish**.

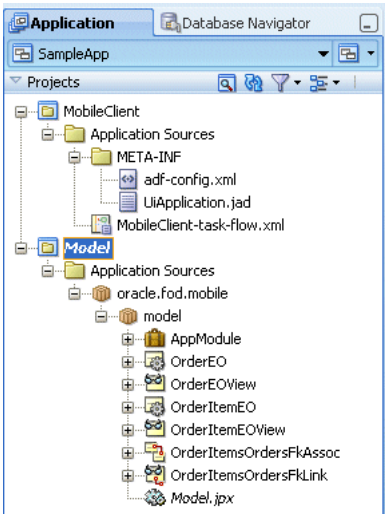
Tip: You can update the entity objects using the overview editor for entity objects, described in Section 5.3, "Editing Mobile Entity Objects."

5.2.2 What Happens When You Create a Subset of Entity Objects and View Objects

When you create a subset of the entity objects and view objects, JDeveloper creates the new application module, along with the entity objects, view objects, associations, and view links. It lists them in the Application Navigator.

For example, Figure 5–5 illustrates the model project and its contents in the Application Navigator.

Figure 5–5 Model Project in the Application Navigator



5.3 Editing Mobile Entity Objects

At any point during application development, you can expose additional attributes or change certain properties of the attributes that already belong to mobile entity objects using the overview editor for entity objects, shown in Figure 5–6. Likewise, you can use overview editors for view objects, view links, and application modules as you would for a standard ADF application. However, because you selected the ADF Mobile client technology when you created the project, these editors display only the properties that are pertinent to ADF Mobile client.

5.3.1 About Using the Overview Editors for Mobile Objects

The overview editors for entity objects, view objects, and application modules include only the features that pertain to ADF Mobile client. Table 5–1 lists the editor features that are not present for ADF Mobile client development.

Table 5–1 Non-ADF Mobile Client Features in Overview Editors

Editor	Page	Feature(s) Not Supported in ADF Mobile Client
Entity Object	General	Features not used for ADF Mobile client development include: ■ Custom Properties ■ Because ADF Mobile client does not support multiple application module, the **Extends** feature is not available; you cannot specify a parent application module.
	Attributes	Features not used for ADF Mobile client include: ■ Custom Properties ■ Security Features not supported in the Attributes Editor ■ Custom Properties page—not included in ADF Mobile client. ■ Dependencies page— not included in ADF Mobile client. Add Rule Validation dialog—Script Expression is not available as a rule type. For more information on the Add Rule dialog, see Section 5.3.5, "Adding Validation Rules."
View Object	General	Features not used for ADF Mobile client include: ■ Because ADF Mobile client does not support multiple application module, the **Extends** feature is not available; you cannot specify a parent application module. ■ Tuning section ■ Alternate keys section
	Entity	Features not used for ADF Mobile client include: ■ You cannot add child entity objects (the **Subtypes** button is not available). ■ Right outer join is not supported as a join type, meaning you cannot designate a view object to return all rows that exist in one entity object, even though corresponding rows do not exist in the joined entity object. For more information, see "How to Create Joins for Entity-Based View Objects" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. ■ **Participate in Row Delete** option is not available. As a result, for the updatable entity object, there is no action of removing rows in the UI that deletes the participating reference entity object.

Table 5–1 (Cont.) Non-ADF Mobile Client Features in Overview Editors

Editor	Page	Feature(s) Not Supported in ADF Mobile Client
	Attribute	Features not used for ADF Mobile include: ■ Custom Properties ■ In the List of Values editor (accessed through the List of Properties section), only **Choice List** is available as the type of component used by the user interface to display the attribute values list. **Most Recently Used Count** is not available; you cannot enter the number of items that display in the choice list. The following features are not supported in the View Attribute dialog (used for adding an attribute) and the View Attribute page accessed through the Edit Attribute dialog: ■ The Expression value type, meaning the default value for the binding variable cannot be based on an expression. ■ Discriminator ■ Effective Date In addition, the View Attribute dialog does not include: ■ Edit—The Edit Expression Editor is not available for the value type. This editor enables you to add an expression that defines the default value for the current attribute of an entity object or view object. ■ Passivate The Edit Attribute dialog does not include the Dependencies tab.

Table 5–1 (Cont.) Non-ADF Mobile Client Features in Overview Editors

Editor	Page	Feature(s) Not Supported in ADF Mobile Client
	Query	In the Bind Variable dialog, the Custom Properties tab and Control Hints tab are not available for ADF Mobile client. In the Variable tab: ■ The Expression option is not available, meaning the default value of a binding variable cannot be based on an expression. ■ ADF Mobile client supports the following Java types for bind variables: ■ BigDecimal ■ Boolean ■ Byte ■ Character ■ Date ■ Double ■ Float ■ Integer ■ Long ■ Number ■ Short ■ String ■ Timestamp For the Query Editor, the features that are not used include: ■ Declarative SQL mode—Only expert and normal modes are available. For more information on expert mode, see "Working with Objects in Expert Mode" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. ■ Alternate Mappings—used to map the columns in the SQL query to view attributes. ■ Alternate Keys—used for defining attributes for the alternate keys. In the View Criteria Editor, you can either select **Database** or **In Memory** as the query execution mode, but not both. For more information, see "How to Create Named View Criteria Declaratively" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. In the UI Hints tab, the following are not available in ADF Mobile client: ■ Search Region—For more information, see "Setting Up Search Form Properties" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. ■ Saved Search List—For more information, see "What Happens at Runtime: Search Forms" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Table 5–1 (Cont.) Non-ADF Mobile Client Features in Overview Editors

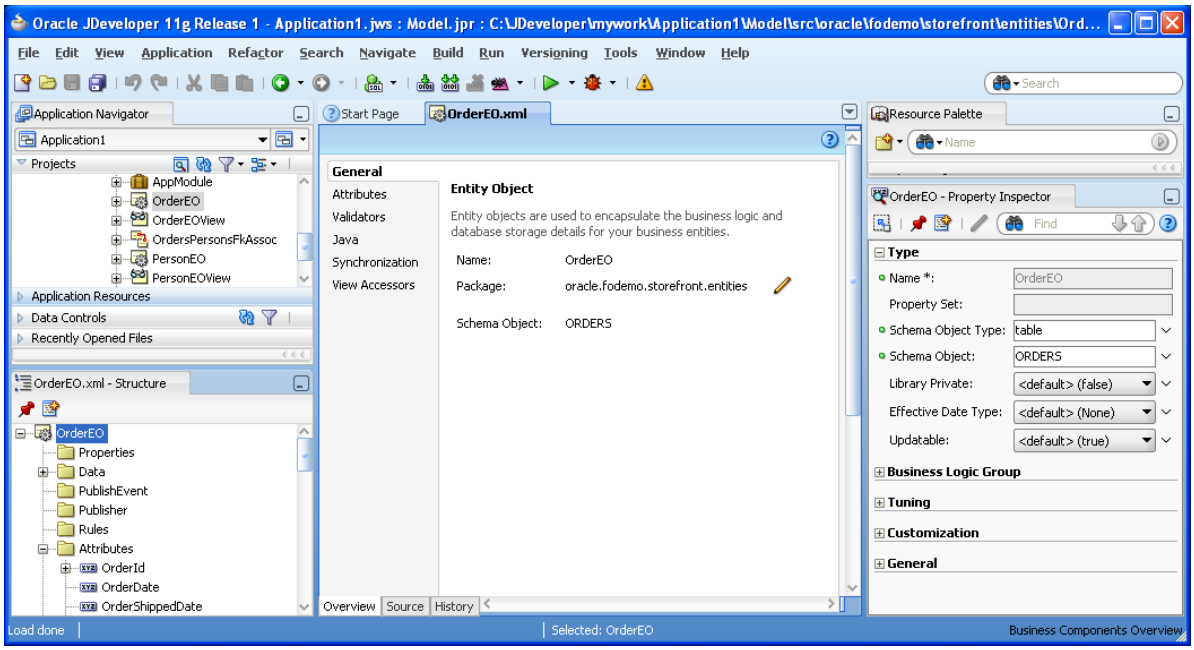
Editor	Page	Feature(s) Not Supported in ADF Mobile Client
	Java	For information on the types supported by ADF Mobile client, see Section 7.2, "Java Support for Business Components." ADF Mobile client supports the following: ■ View Object Class ■ View Row Class ■ View Object Client Interface
	List UI Hints	For ADF Mobile client: ■ **Only Choice List** is available as the type of component used by the user interface to display the attribute values list. ■ **Most Recently Used Count** is not available; you cannot enter the number of items that display in the choice list.
Application Module	General	Because ADF Mobile client does not support multiple application module, the **Extends** feature is not available; you cannot specify a parent application module. The Tuning and Custom Properties sections are not available for ADF Mobile client.
	Data Model	The Subtypes feature is not available, so you cannot add a child view instance to support of polymorphic view objects for a selected view instance.
	Java	ADF Mobile client supports the following: ■ Application Module Class ■ Application Module Client Interface ■ Application Module Client Class ADF Mobile client does not support Application Module Definition Class.
	EJB Session Bean	This page is not available for ADF Mobile client.
	Service Interface	This page is not available for ADF Mobile client.
	Configurations	This page is not available for ADF Mobile client.

5.3.2 About Editing Entity Objects

Because the mobile client application synchronizes with its server-side counterpart, this editor allows you to create transient attributes. You cannot create persistent attributes because a mobile client application synchronizes data as exposed through its entity objects against server-side data, which is exposed on an attribute-by-attribute basis, through server-side entity objects backed by a database table. If you create an entity object attribute that only exists in a mobile application, then this attribute cannot be synchronized against a server-side counterpart because one does not exist. You can create transient attributes using this editor, because by definition, a transient attribute does not persist and thus is not synchronized against server-side data.

Note: ADF Mobile client does not support groovy expressions. To set a value for calculated fields in an entity object, you must override the `set` method in the Java code for that attribute.

Figure 5–6 The Overview Editor for an Entity Object



Note: Because the location of the ADF Library JAR used to create an ADF Mobile client application is hard-coded in the resulting entity objects, you must update the JAR's location for the `NMCBaseLibraryFilePath` property if you move the mobile application to another computer or change or delete the JAR. If you do not change this value, then the application will not run.

5.3.3 How to Add Attributes to an Entity Object

You use the entity editor to modify the attributes belonging to a mobile entity object, or to add transient attributes to one. Because the mobile application must synchronize with the base application, you cannot change certain properties for an attribute. Otherwise, synchronization fails.

Use the Attributes page of the overview editor to create an attribute.

Before you begin:

Create a subset of entity objects for the ADF Mobile client application as described in Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects."

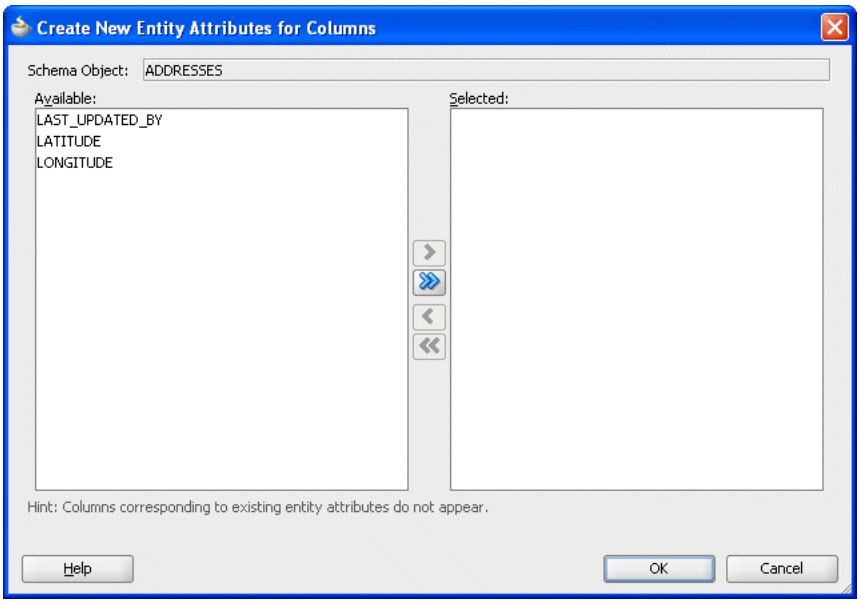
To add attributes:

1. Double-click an entity object in the Application Navigator.

   Note: The Name and Extends properties are read-only values.

2. Click the **Attributes** tab.
3. In the Attributes page, click **Add from Base Entity**. In the Create New Entity Attributes for Columns dialog, shown in Figure 5–7, the attributes that you did not select for the entity object using the Model Wizard appear in the **Available** list.

Figure 5–7 Selecting Attributes



4. Select the attributes by moving them from the **Available** window to the **Selected** window.
5. Click **OK**. The selected attributes display in the table.

> **Note:** You cannot use this editor to correct or update persistent attributes. If an attribute has been designed improperly, you can only correct it in the server-side application. You cannot edit the persistent attributes because they must match the attributes of the server object. For example, changing an attribute type from a date to a number causes synchronization to fail.

5.3.4 How to Add Transient Attributes

Use the Attributes page of the overview editor to create a transient attribute.

Before you begin:

Create a subset of entity objects for the ADF Mobile client application as described in Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects."

To Add transient attributes:

1. Click **Add** in the Attributes page.

> **Note:** ADF Mobile client supports only transient attributes based on literal values.

2. In the New Entity Attribute dialog, define the properties for the transient attribute (listed in Table 5–2).

Table 5–2 Attribute Properties

Property	Description
Name	The name of the attribute. This must be a valid Java identifier.
Type	Choose the Java type for the attribute
Property Set	Choose the type of element that this attribute represents. This field is not available for all attribute types.
Value Type	Select **Literal** to use a literal value for the default
Value	For discriminator columns in subtypes of polymorphic entity objects, enter the discriminator value for the subtype. ADF Mobile client does not support expressions.
Mandatory	Select if this attribute is mandatory. This option is selected by default if the corresponding database column has a NOT NULL constraint.
Derived from SQL Expression	ADF Mobile client does not support this option. Do not use it.
Discriminator	ADF Mobile client does not support this option. Do not use it.
Updatable	The view attribute setting is based on the entity attribute setting and can made more restrictive. ■ Always Select to make an attribute updatable. ■ While New Select to make an attribute updatable before an entity is first posted. ■ Never Select to make an attribute read-only.

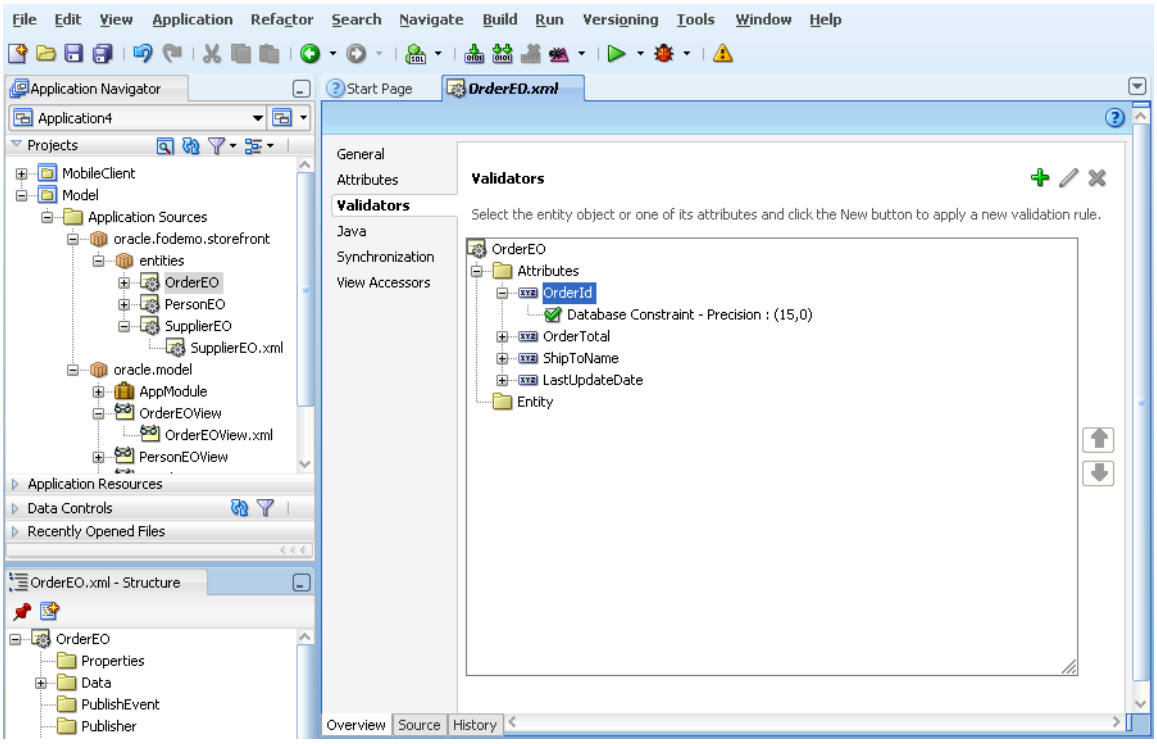
3. Click **OK**. The attribute appears in the table.

5.3.5 Adding Validation Rules

The Validators page of the overview editor enables you add the built-in Oracle ADF declarative validation rules described in "Defining Validation and Business Rules Declaratively" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. When you create a validation rule using the overview editor, they are stored in the entity object's XML file.

Figure 5–8 shows the Validators page for the entity object, OrderEO.

Figure 5–8 The Validators Page



ADF Mobile client supports the following validators, which are available from the Rule Type list of the Add Validation Rule dialog, which you invoke by first selecting an entity or attribute and then by clicking **Add**.

Figure 5–9 shows the Add Rule Validation dialog.

Figure 5–9 The Add Validation Rule Dialog

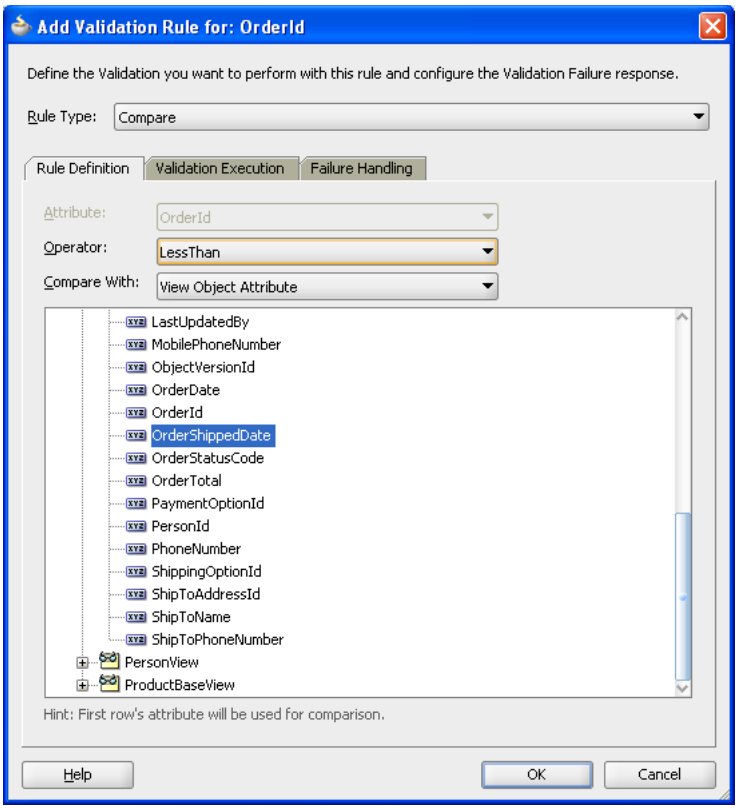


Table 5–3 lists the validators and the classes that they support.

Table 5–3 Validators and Their Supported Classes

Validator	Usage	Supported Classes
Compare	Performs a logical comparison between an entity attribute and a value. When you add a Compare validator, you specify an operator and a comparison. Entity Operators include: ■ Literal value ■ Query result ■ View object attribute ■ View accessor attribute ■ Expression ■ Entity attribute For more information, see "How to Validate Based on Comparison" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.	`oracle.jbo.rules.JboCompareValidator`
Key Exists	Determines whether a key value (primary, foreign, or alternate key) exists. For more information, see "How to Determine Whether a Key Exists" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.	`oracle.jbo.server.JboEOExistsValidator`, `oracle.jbo.server.JboVOExistsValidator`, `oracle.jbo.server.JboVOUsageExistsValidator`
Length	Validates whether the string length (in characters or bytes) of an attribute's value is less than, equal to, or greater than a specified number, or whether it lies between a pair of numbers. For more information, see "How to Validate Against a Number of Bytes or Characters" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*	`oracle.jbo.rules.JboLengthValidator`
List	Compares an attribute against a list of values (LOV). For more information, see "How to Validate Using a List of Values" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.	`oracle.jbo.rules.JboListValidator`

Table 5–3 (Cont.) Validators and Their Supported Classes

Validator	Usage	Supported Classes
Method	Supplements declarative validation rules and Groovy-scripted expressions using your own Java code. Method validators trigger Java code that you write in your own validation methods at the appropriate time during the entity object validation cycle. For more information, see "Using Method Validators" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.	`oracle.jbo.rules.JboMethodValidator`
Range	Performs a logical comparison between an entity attribute and a range of values. When you add a Range validator, you specify minimum and maximum literal values. The Range validator verifies that the value of the entity attribute falls within the range (or outside the range, if specified). For more information, see "How to Make Sure a Value Falls Within a Certain Range" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.	`oracle.jbo.rules.JboRangeValidator`
Regular Expression	ADF Mobile Client does not support Groovy Expressions. Do not create validation rules as Regular Expressions. Use **Method** validation rule for complex validation rules. for more information see Chapter 7, "Extending ADF Mobile Client Applications with Java.".	`oracle.jbo.rules.JboRegExpValidator`

For more information, see "Using the Built-In Declarative Validation Rules" and "Implementing Validation and Business Rules Programmatically" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

5.3.5.1 How to Add a Validation Rule to an Entity or Attribute

To add a declarative validation rule to an entity object, use the Validators page of the overview editor.

To add a validation rule:

1. In the Application Navigator, double-click the desired entity object.
2. Click the **Validators** navigation tab on the overview editor.
3. Select the object for which you want to add a validation rule, and then click the **Add** icon.
   - To add a validation rule at the entity object level, select **Entity**.

- To add a validation rule for an attribute, expand **Attributes** and select the desired attribute.

When you add a new validation rule, the Add Validation Rule dialog appears.

4. Select from the validation rules in **Rule Type** list.
5. Use the dialog settings to configure the new rule.

   The controls will change depending on the kind of validation rule you select. For more information about the different validation rules, see "Using the Built-in Declarative Validation Rules" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
6. You can optionally click the **Validation Execution** tab and enter criteria for the execution of the rule, such as dependent attributes and a precondition expression. For more information, see "Triggering Validation Execution" *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

   Note: For Key Exists and Method entity validators, you can also use the **Validation Execution** tab to specify the validation level.
7. Click the **Failure Handling** tab and enter or select the error message that will be shown to the user if the validation rule fails. For more information, see "Creating Validation Error Messages" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
8. Click **OK**.

5.3.6 Overriding Default Validation Error Handling

When a validation error occurs, a message box showing error messages appears by default. You can override the default behavior if you prefer the validation error message as an `outputText` component rather than as a message box.

Example 5–1 and Example 5–2 are two versions of the same application, `CustomErrorHandler.java`. Example 5–1, describes how to use a message box for the validation error message (similar to the default behavior) and how the main error handling method, `reportException`, handles the error messages in this context. Example 5–2 describes how to put the validating error messages into an `outputText` component by overriding the `reportException` method.

5.3.6.1 How to Show the Error Message as a Message Box

Example 5–1 illustrates default behavior of the error handler.

To show a message box with an error message:

1. Declare the error handler as an attribute of the `form` component. For example, in an MCX file, you would declare the error handler as an attribute of the `form` component as follows:

   ```
   <amc:form errorHandler="view.backing.CustomErrorHandler" ... >
   ```

   Declaring the error handler creates the class and casts it to an `oracle.adfnmc.component.ErrorHandler` interface.
2. Create the error handler class and implement the methods of the `oracle.adfnmc.component.ErrorHandler` interface.

The `reportException` method returns a boolean as to whether the error was handled. If this method returns false, the default error handling logic is invoked.

Example 5–1 view/backing/CustomErrorHandler.java

```
package view.backing;

import oracle.adfnmc.bindings.BindingContainer;
import oracle.adfnmc.bindings.dbf.BindingContext;
import oracle.adfnmc.component.ErrorHandler;
import oracle.adfnmc.component.MessageBox;
import oracle.adfnmc.component.ui.Form;
import oracle.adfnmc.java.util.List;


public class CustomErrorHandler
  implements ErrorHandler
{
  public CustomErrorHandler()
  {
  }

  public boolean reportException(Form form, BindingContainer formBnd, Exception
ex)
  {
    // do error handling here
    List errors = null;
    if (formBnd != null)
    {
      errors = formBnd.getErrors();
    }

    if ((errors != null) && (errors.size() > 0))
    {
      String[] messages = new String[errors.size()];
      for (int i = 0; i < errors.size(); ++i)
      {
        Exception exListElement = (Exception) errors.get(i);
        messages[i] = this.getDisplayMessage(BindingContext.getInstance(),
exListElement);
      }
      MessageBox.show(messages);
    }
    else
    {
      String message = this.getDisplayMessage(BindingContext.getInstance(), ex);
      MessageBox.show(message);
    }

    // return true so that the form knows the error has been handled
    return true;
  }

  public String getDisplayMessage(BindingContext ctx, Exception ex)
  {
    if (ex != null)
    {
      return ex.getMessage();
    }
    else
```

```
    {
      return "";
    }
  }
}
```

5.3.6.2 How to Show the Error Message as Output Text

Example 5–1 is similar to Example 5–2, but instead illustrates how to show an error message as an `outputText` component.

To show the error message as output text

1. Declare the error handler as an attribute of the `form` component.
2. Declare an `outputText` component in the MCX file and set its value to an EL expression. For example:

   ```
   <amc:outputText value="#{applicationScope.validationMessage}"
   foregroundColor="#FF0000" />
   ```

3. Override `reportException` in the Java class and set the EL expression whenever there is an error.

Example 5–2 view/backing/CustomErrorHandler.java Showing Output Text Error Message

```
package view.backing;

import oracle.adfnmc.bindings.BindingContainer;
import oracle.adfnmc.bindings.dbf.BindingContext;
import oracle.adfnmc.component.ErrorHandler;
import oracle.adfnmc.component.ui.Form;
import oracle.adfnmc.el.ValueExpression;
import oracle.adfnmc.el.impl.SimpleContext;
import oracle.adfnmc.java.util.List;


public class CustomErrorHandler

  implements ErrorHandler
{
  public CustomErrorHandler()
  {
  }

  public boolean reportException(Form form, BindingContainer formBnd, Exception
ex)
  {
    // do error handling here
    List errors = null;
    if (formBnd != null)
    {
      errors = formBnd.getErrors();
    }

    String message = "";
    if ((errors != null) && (errors.size() > 0))
    {
```

```
      for (int i = 0; i < errors.size(); ++i)
      {
        Exception exListElement = (Exception) errors.get(i);
        message += this.getDisplayMessage(BindingContext.getInstance(),
exListElement) + "\n";
      }
    }
    else
    {
      message = this.getDisplayMessage(BindingContext.getInstance(), ex);

    }

    ValueExpression valueExpression =
      SimpleContext.getValueExpression("#{applicationScope.validationMessage}",
String.class);
    valueExpression.setValue(message);

    // return true so that the form knows the error has been handled
    return true;
  }

  public String getDisplayMessage(BindingContext ctx, Exception ex)
  {
    if (ex != null)
    {
      return ex.getMessage();
    }
    else
    {
      return "";
    }
  }
}
```

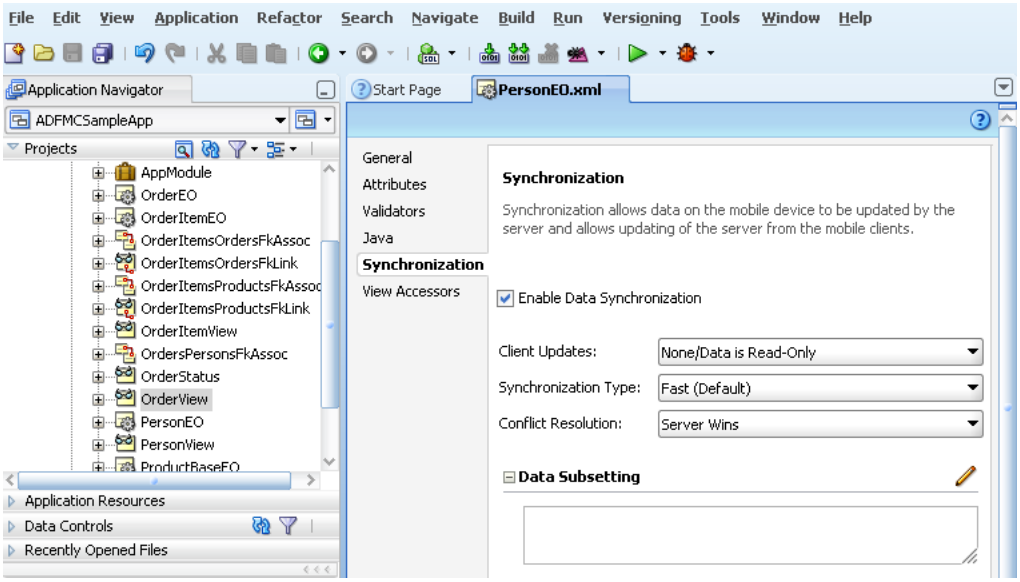
5.3.7 About Synchronization for Entity Objects

Data synchronization is not enabled be default for entity objects. Only the entity objects that are imported from a known schema context can be enabled for synchronization.

You can enable or disable synchronization for each entity object using the Synchronization page of the overview editor for entity objects, shown in Figure 5–10. This page enables you to do the following:

- Set data updates—You can set how the entity objects receive updates from the server and also how the entity objects can update the server. The downstream updates (from the server to the client) are accomplished through Oracle Database Lite Mobile Server (Mobile Server). To perform upstream updates, an entity object must be enabled for ADF Mobile transaction replay service.
- Set conflict resolution rules—You can also use this page to set the rules for resolving data conflicts between the client and server, as conflicts can arise if a mobile client updates a stale record while the original record may have been changed on the server.
- Create a data set appropriate for a small database—You can refine the entity object's SQL statement by adding `WHERE` and `ORDER BY` clauses to the JDeveloper-generated `SELECT` statement based on the selected entity attributes.

Figure 5–10 Setting the Synchronization for an Entity Object



5.3.7.1 How to Enable or Disable Synchronization for Entity Objects

The Synchronization page enables you to enable or disable synchronization for entity objects that have been enabled for ADF Mobile transaction replay service as well as for those that have not. Depending on whether transaction replay service has been enabled for the selected entity object, this page enables you to set both the upstream (client-to-server) and downstream (server-to-client) updates.

Before you begin:

The ADF Mobile client application must include entity objects imported from a base (server) application using the Create Business Components from ADF Library wizard as described in Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects." To use any of the ADF Mobile transaction replay service configuration options, the selected entity must be enabled for transaction replay service as described in Section 5.8, "Enabling ADF Mobile Transaction Replay Service for an ADF Application."

To configure synchronization for entity objects:

1. Select **Enable Data Synchronization**.

   Note: You can disable synchronization if you do not select **Enable Data Synchronization**.

2. Select how the ADF Mobile client application updates with the server. For downstream, Oracle Database Lite Mobile Server - enabled updates, select **Updates made via Tables**. For upstream updates, select **Updates via TRS**.

   Note: The **Updates via TRS** option is only available if the entity has been enabled for transaction replay service.

3. Select the type of table-level synchronization used by the entity object. The options include:

- Fast (the default setting)
- Complete Snapshot
- Queue-based
- None

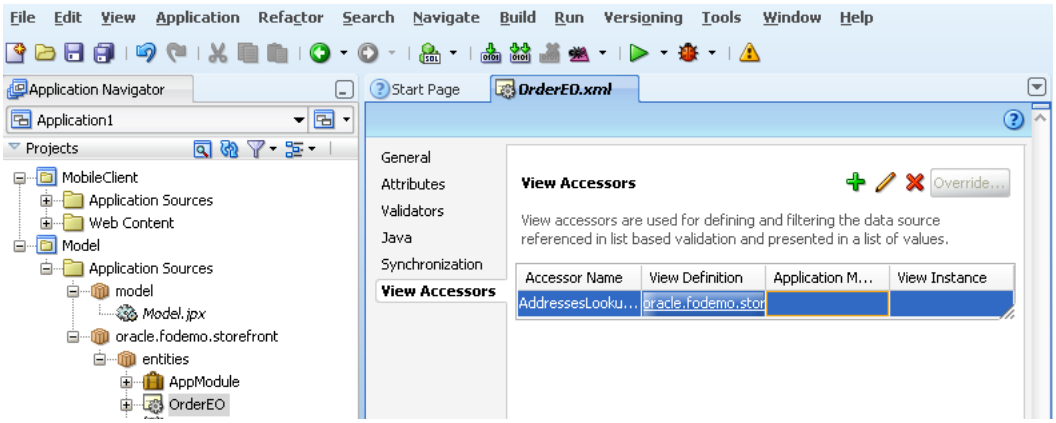
4. Select how the data conflicts are resolved between the client and server.
5. Click **Edit**.
6. Select which transaction replay service events are enabled for synchronization.
7. Add a `WHERE` and `ORDER BY` clause to the JDeveloper-generated SQL statement query to filter and order the data as required. You can add `ORDER BY` clauses using the Order By dialog (accessed by clicking **Edit**). If you prefer not to use the JDeveloper-generated `SELECT` clause and `FROM` list, then select **Expert Mode** from the **SQL Mode** drop down and enter the SQL statement in the editor as described in "How to Customize SQL Statements in Expert Mode" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

5.3.8 View Accessor Support for Entity Objects and View Objects

View accessors are value-accessor objects in ADF Business Components. You create a view accessor to point from a base entity object attribute or view object attribute to a source view row set. The view accessor returns a list of all possible values to the attribute of the base object.

View Accessors are set in the View Accessors page of the Overview Editor for entity objects and view objects, shown in Figure 5–11. For more information, refer to "How to Create a View Accessor for an Entity Object or View Object" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Figure 5–11 The Overview Editor for View Accessor Page



5.3.9 Using List UI Hints for View Objects

You can associate a control hint with the current entity attribute. View objects will inherit the hint values at runtime. For more information, see "Defining Attribute Control Hints for Entity Objects" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Note: ADF Mobile client does not support the following in the Control Hints dialog. Do not use them.

- Display Hint
- Tooltip Text
- Control Type
- Display Width
- Display Height
- Form Type
- Auto Submit

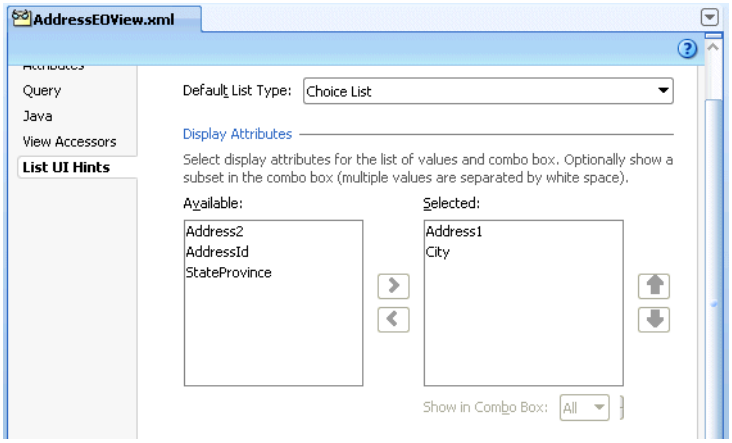
Because any attribute-specific information that is added to the base application is available to the ADF Mobile application after you complete the Create Mobile Business Components from ADF Library wizard, you must re-import objects from the base application if attribute-specific changes have been made to it. Otherwise, control hints are not available to the ADF Mobile client application.

5.3.10 Using Display Hints for Entity Objects

You can specify default LOV (list of values) hints when you want the LOV-enabled attributes of other view objects to inherit the list UI hints from the current view object. To define LOV-enabled attributes, you must create a view accessor that points to the view object that supplies the values. At design time, the LOV-enabled attributes whose view accessor points to this source view object will inherit any list UI hints you have defined. At runtime, the UI will display the same LOV usage for these LOV-enabled attributes. For more information, see "Working with List of Values (LOV) in View Object Attributes" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

For example, Figure 5–12 shows the List UI Hints page of the Overview Editor for AddressEOView, a view object.

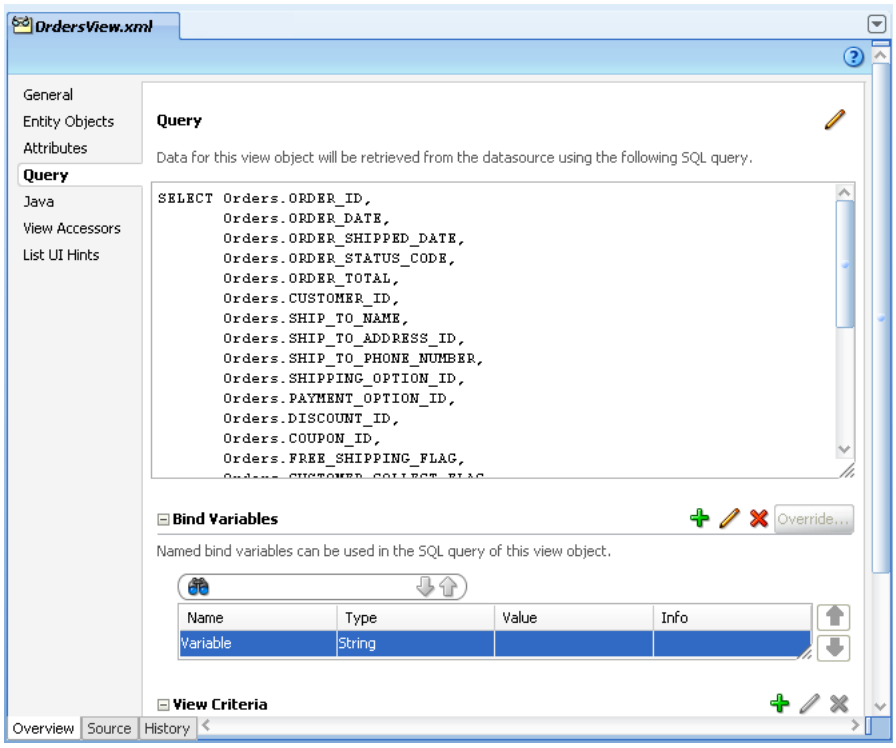
Figure 5–12 The Overview Editor for List UI Hints



5.3.11 Adding Bind Variables to View Objects

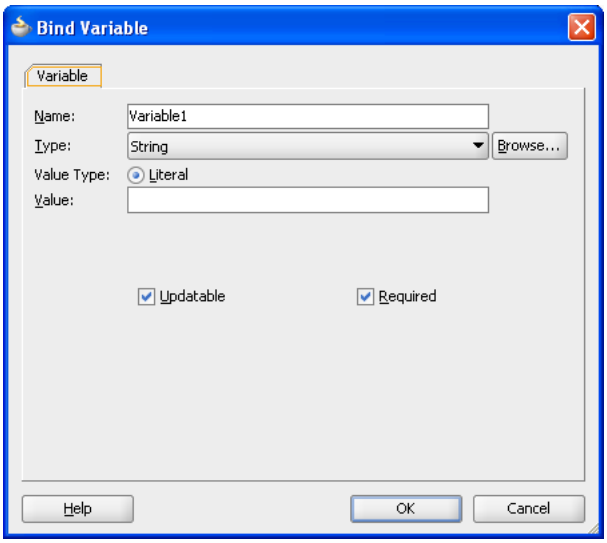
The Query page of the overview editor for a view object (illustrated in Figure 5–13 and described in "Working with Bind Variables" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*) enables you to add bind variables to a query. Although ADF Mobile client supports JDBC Positional, Oracle Named, and Oracle Positional binding styles, Oracle Named is the preferred binding style and is the only one supported in the overview editor for mobile client view objects.

Figure 5–13 The Query Page of the Overview Editor for View Objects



You edit the binding using the Bind Variable dialog, shown in Figure 5–14. For more information on Oracle Named binding style, see the online help available from JDeveloper.

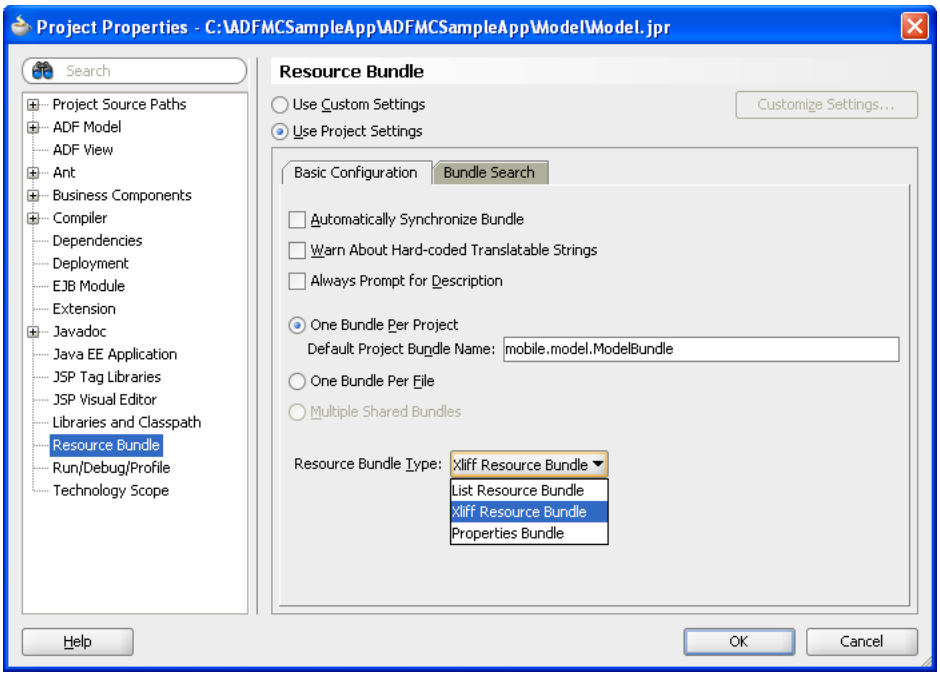
Figure 5–14 The Bind Variable Dialog



5.3.12 Working with Resource Bundles

ADF Mobile client provides design-time localization support by enabling the use of the standard localization structures of ADF to specify resource strings in multiple languages. As with standard ADF applications, you can configure an ADF Mobile client application to store translatable strings such as control hints in a resource bundle as described in "Working with Resource Bundles" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. As shown in Figure 5–15, in addition to the default `.properties` bundle file, ADF Mobile also supports the XLFF (XML Localization Interchange File Format) properties bundles used for exchanging localization data. ADF Mobile client supports multiple resource bundles. For more information, see Section 7.4.2, "Supporting Localization through XLFF Resource Bundles."

Figure 5–15 Selecting Resource Bundle Type for an ADF Mobile Client Project



5.4 The Entity Object and View Object Extension

You can extend the mobile application using the same techniques for extending an Oracle ADF application, described in "Advanced Business Components Techniques" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. You extend mobile client applications using the Java page of the Overview Editor for the entity object or view object. For more information on using the overview editor, see "Creating and Modifying an Application Module" and "Generating Custom Java Classes for a View Object" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*

Note: ADF Mobile client applications, which are based on the J2ME platform, differ from J2EE-based Oracle ADF applications. Because J2ME lacks the VM support to implement Java reflection, ADF Mobile client applications do not bind methods dynamically; they instead link methods statically at compile time. To simulate dynamic binding, you must add delegation code to invoke application modules or view object from the EL expressions and the entity object validation methods as described in Chapter 7, "Extending ADF Mobile Client Applications with Java."

The ADF Mobile client entity objects and view objects support only a subset of the functionality available to the server entity objects and view objects. Java Assist in JDeveloper filters out the methods that are not supported.

Note: When you generate entity object and view object Java implementations for ADF Mobile client applications, the generated Java source must be compatible to Java 1.3. The source uses integers for field indexes instead of enumerations.

5.4.1 Supported Constructs

The following JDeveloper-generated Java classes are supported by the runtime:

- entity object
- view object
- view row
- application module

The following are not used by the runtime, but can safely be generated and used at design time.

- view row client Interface
- view row client

The following are explicitly not supported:

- entity collection
- entity definition
- view object definition
- application module definition

5.4.2 Unsupported Methods

Even though the classes listed in Section 5.4.1, "Supported Constructs" are supported by the runtime, ADF Mobile client does not support every method in those classes. Do not override or otherwise call the following methods:

entity object `(oracle.jbo.server.EntityImpl)`

- `doDMLWithLOBs(int operation, TransactionEvent e)`
- `handleEffectiveDateOperations()`
- `validateDateEffectivity()`
- `removeAndRetain()`

view object `(oracle.jbo.server.ViewObjectImpl)`

- `addEffectiveDateDstAttributes()`
- `buildEffectiveDateFromClauseFragment(StringBuffer fromClause, int noUserParams)`
- `sortRows(Row[] rows)`
- `activateNewRowTracker(ViewRowSetImpl vrs, KXmlParser parent)`
- `validateRangeSizeForRangePaging(ViewRowSetImpl vrs, int rangeSize)`
- `activateState(ViewRowSetImpl vrs, ViewRowImpl currentRow, KXmlParser parent)`

view row `(oracle.jbo.server.ViewRowImpl)`

application module `(oracle.jbo.server.ApplicationModuleImpl)`

- `activateState(int id, SessionData info, int flags)`
- `passivateStateForUndo(String id, byte[] clientData, int flags)`

- `getAMSerializer()`

5.5 Testing Application Modules

You can test the application logic of the application module (created using the Create Business Components for ADF Mobile Client wizard) using the Business Component Browser. To launch the Business Component Browser, select the application module in the Application Navigator and then choose **Run**.

Note: If you receive a method not found exception when you test the application module, rearrange the model project's classpath entries as follows:

1. In the Application Navigator, right-click the model project and choose **Project Properties**.
2. Select **Libraries and Classpath**.
3. Select the ADF Mobile Client Runtime classpath entry.
4. Using Move Up or Move Down, move the ADF Mobile Client Runtime entry directly beneath the ADF Model Runtime entry.
5. Click **OK**.

For more information on the Business Component Browser, see "Testing Application Modules Using the Business Component Browser" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

5.6 Interacting Directly with SQLite

The ADF Mobile client framework provides a large degree of compatibility with BC4J, which is designed to relieve developers from manually performing create, update, delete (CRUD) operations. Nonetheless, there are use cases that may require bypassing the framework and interacting directly with the underlying database. As illustrated in Example 5–3, ADF Mobile client supports these use cases in a manner that is similar to BC4J's.

Example 5–3 Obtaining a Database Connection

```
import oracle.adfnmc.bindings.DataControl;
import oracle.adfnmc.bindings.bc4j.BC4JDataControl;
import oracle.adfnmc.bindings.dbf.BindingContext;
import oracle.jbo.ApplicationModule;

DataControl dc =
(DataControl)BindingContext.getInstance().get("AppModuleDataControl");
if (dc instanceof BC4JDataControl)
{
  ApplicationModule am = (ApplicationModule)dc.getDataProvider();
  DBTransaction dbTrans = am.getDBTransaction();
  PreparedStatement prepStmt = null;
  try
  {
    prepStmt = dbTrans.createPreparedStatement("arbitrary SQL query");
    // work with PreparedStatement as usual

  }
  finally
```

```
  {
    if (prepStmt != null)
      prepStmt.close();
  }
}
```

5.6.1 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 are a number of limitations that may be foreign to developers accustomed to working with Oracle databases. The most important differences are:

- Concurrency
- SQL Support
- Data Types
- Foreign Keys
- Database Transactions

> **Note:** Consult the Documentation section of the SQLite site (http://www.sqlite.org/docs.html) for complete details.

5.6.1.1 Concurrency

Do not open your own connection to the database at any time during application execution. Always reuse the framework's connection by following Example 5–3. This limitation arises directly from SQLite's coarse-grained locking. For more information, see the following documents available from the Documentation section at the SQLite site (http://www.sqlite.org/docs.html):

- File Locking And Concurrency In SQLite Version 3
- BEGIN TRANSACTION

5.6.1.2 SQL Support

Although SQLite complies with the SQL92 standard, there are a few unsupported constructs. For more information, see *SQL Features That SQLite Does Not Implement*, available from the Documentation section at the SQLite site (http://www.sqlite.org/docs.html).

5.6.1.3 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. The ADF Mobile client framework takes care of converting values to and from the database in accordance with the relevant entity metadata, but if you are working directly against the database, be sure to maintain data integrity. SQLite will 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 at the SQLite site (http://www.sqlite.org/docs.html).

5.6.1.4 Foreign Keys

Although SQLite parses foreign key constraints, it does not enforce them. For more information, see the SQLite FAQ available from the Documentation section at the SQLite site (http://www.sqlite.org/docs.html).

5.6.1.5 Database Transactions

Although SQLite is ACID-compliant and hence supports transactions, there are some fundamental differences between its transaction support and Oracle's.

5.6.1.5.1 Nested Transactions SQLite does not support nested transactions. Only a single transaction may be active at any given time.

5.6.1.5.2 Savepoints Although SQLite itself supports transaction savepoints, support for this feature is not exposed in the ADF Mobile client framework.

5.6.1.5.3 Commit SQLite permits either multiple read-only connections **or** a single read-write connection to any given database. The ADF Mobile client framework manages a single read-write connection to your application's database and shares it with OracleLite Mobile Server Sync (mSync) as needed. As long as you push changes to the database through standard ADF business components, you can simply invoke `DBTransaction.commit()` just as you would for an ADF Faces application.

5.6.1.5.4 Rollback ADF Mobile client fetches rows from the database into memory as needed, like BC4J. This generally occurs as you iterate over the rows of the application's viewobjects. As a consequence, there are usually multiple `ResultSets` open at any given time while the application executes. This does not present a problem for traditional server-grade databases such as Oracle, but SQLite does not permit a transaction to be rolled back until all open `ResultSets` have been closed first. ADF Mobile client has been designed to work around this limitation in the following manner:

- When `DBTransaction.rollback()` is invoked, the application is first disconnected from its database. This will implicitly close all open JDBC ResultSets and cancel any changes that have been posted to the database but not yet committed.
- Next, the application is reconnected to the database and rollback proceeds as usual.
- The state of all business components is preserved across the database disconnect, including information about which row was the current row in a given view object. Therefore, when the next row is requested by the application, the framework can return the row that would have come next had the application not been disconnected from its database.

This sequence of steps effectively produces the same behavior as a simple transaction rollback in an ADF Faces application backed by an Oracle database.

5.7 Configuring JDeveloper to Connect to and Test Against a SQLite Database

During the development cycle, you may need to test against the SQLite databases that are generated from OracleLite Mobile Server Sync (mSync). Doing this enables you to test against real data using direct SQL statements and verify if the SQL statement is valid against the SQLite database. Because you must always test manually created SQL statements against SQLite databases, this testing method is useful when you

define view objects and create custom SQL statements in expert mode as described in"Working with View Objects in Expert Mode" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*. Also, when a query returns unexpected results, you can test the generated SQL in question by copying and pasting the SQL statement into the view object editor.

5.7.1 How to Test Against a SQLite Database

Use JDeveloper's Database Navigator to create a database connection to the SQLite database. Use the SQL Worksheet to test SQL statements and the update SQLite database schema.

Before you begin:

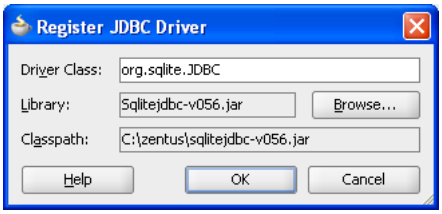
Download Oracle JDeveloper and the ADF Mobile client extension as described in Section 2.3, "Setting Up JDeveloper."

Download the SQLiteJDBC driver from `http://www.zentus.com/sqlitejdbc/`. This driver is contained in a JAR file called `sqlitejdbc-v0xx.jar`. Save this file to an easily accessed location.

To register the Zentus SQLite JDBC driver and test against the SQLite database:

1. In JDeveloper, create a new database connection. You can create this connection in the application or in the IDE shows how a finished connection may look. The name and password fields are dummy values.
2. In the Database Navigator, select **File** > **New** > **Connections** and then **Database Connections**.
3. Select **Generic JDBC** from the Connection drop-down menu.
4. Click **New**.
5. Enter **org.sqlite.JDBC** as the Driver Class in the Register JDBC Driver dialog, as shown in Figure 5–16.

Figure 5–16 Registering the Zentus SQLite JDBC Driver

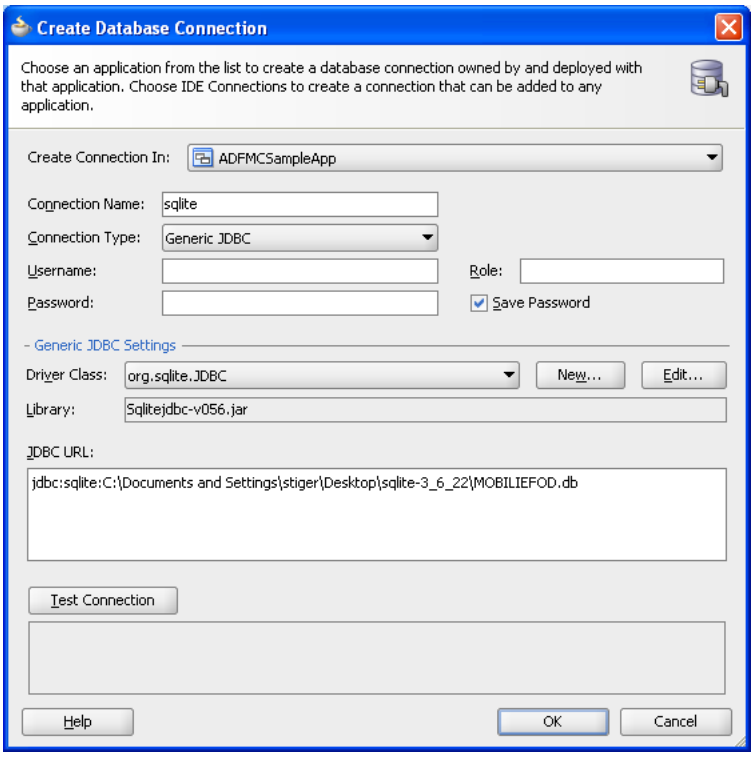


6. Click **Browse**, click **New**, and then click **Add Entry**.
7. Select the **sqlitejdbc-v0xx.jar** on your file system. Click **OK.**
8. Click **OK** until you return to the Create Database Connection page.
9. In the JDBC URL window, enter

   `jdbc:sqlite:<Path to your SQLite data file>` as shown in Figure 5–17.

   For example, enter `jdbc:sqlite:C:\Documents and Settings\<your name>\Desktop\sqlite-3_6_22\MOBILEFOD.db`, where `MOBILEFOD.db` is the SQLite file containing the data.

Figure 5–17 The Create Database Connection Dialog



10. Enter dummy values in the name and password fields.
11. Click **Test Connection**. A *Success!* message appears if the connection is configured correctly.
12. In the Database Navigator, right-click the connection and select **SQL Worksheet**, from the context menu. The SQL Worksheet enables you to test SQL statements and also review, modify, and export the SQLite database.

5.8 Enabling ADF Mobile Transaction Replay Service for an ADF Application

Enabling transaction replay service for an ADF application involves the following steps:

1. Enabling the server (base) application for transaction replay service by adding the ADF Mobile transaction replay service technology to the project.
2. Adding a Replay Type to a mobile client application using the TRS Enablement Wizard.

5.8.1 How to Add the ADF Mobile Transaction Replay Service Technology to an ADF Application

You can add the ADF Mobile transaction replay service technology to a new or existing ADF project. You can enable transaction replay service for any application that has the `web.xml` deployment descriptor file.

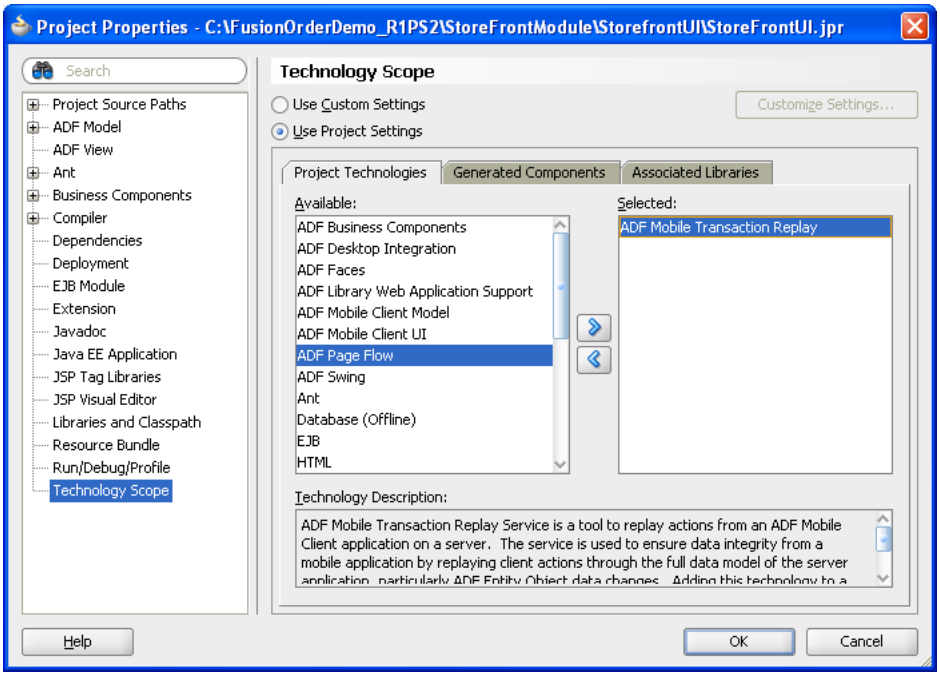
Before you begin:

You must have an ADF application model project containing the entity objects and view objects on which you base the ADF Mobile client application.

To add the ADF Mobile transaction replay service technology:

1. Select the a project in the Application Navigator, then select **Edit** and then **Properties**.
2. In the Technology Scope page, select **ADF Mobile Transaction Replay** from the Available window and move it to the Selected window as shown in Figure 5–18. Click **OK**.

Figure 5–18 Adding the ADF Mobile Transaction Replay Service Technology



3. Deploy the application as an ADF JAR as described in "Packaging a Reusable ADF Component into an ADF Library" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

5.8.2 What Happens When You Add the ADF Mobile Transaction Replay Service Technology to an Application

When you add the ADF Mobile transaction replay service technology to an application, JDeveloper adds the transaction replay service servlet JAR, `ReplayServlet`, and updates `web.xml` with transaction replay service configuration parameters as shown in Example 5–4.

Example 5–4 ADF Mobile Transaction Replay Service Configuration Parameters in web.xml

```
<servlet-name>ReplayServlet</servlet-name>
    <servlet-class>oracle.txnreplay.adfadapter.ReplayServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
```

```
    <url-pattern>/faces/*</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>ReplayServlet</servlet-name>
    <url-pattern>/replayservlet</url-pattern>
  </servlet-mapping>
```

> **Note:** Because transaction replay service requires a `web.xml` project in the base (server) application, you usually add the ADF Mobile technology scope to the view-controller project.

In addition, JDeveloper updates the `web.xml` file's `<filter-mapping>` element for `ReplayServlet`, as shown in Example 5–5.

Example 5–5 Filter Mapping in web.xml

```
<filter-mapping>
    <filter-name>adfBindings</filter-name>
    <servlet-name>ReplayServlet</servlet-name>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
  </filter-mapping>
  <filter-mapping>
    <filter-name>JpsFilter</filter-name>
    <servlet-name>ReplayServlet</servlet-name>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
  </filter-mapping>
```

5.8.3 What Happens When JDeveloper Creates an ADF Mobile Transaction Replay Service-Enabled Application

If a project already is already configured with the ADF Faces technology, then you can enable the transaction replay service Object Replay mechanism in the application.

5.8.4 How to Create a Transaction Replay Type

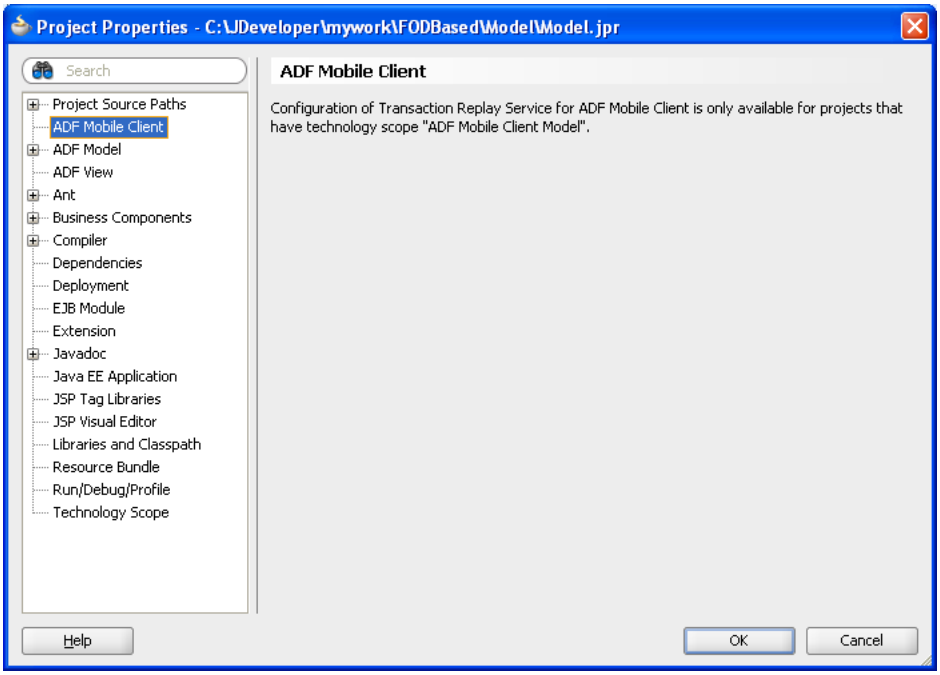
The TRS Enablement Wizard configures the ADF Mobile client application, project, and entity objects to enable the use of transaction replay service. The wizard verifies that the ADF Mobile application has entity object definitions from an ADF library file. When you create the entities for an ADF Mobile client application using the Create Business Components from ADF Library wizard as described in Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects," the entity objects imported by this wizard are enabled for synchronization because they have been given create, update, and delete actions. For applications that have not been created using this wizard, you can enable transaction replay service for the entity objects using the TRS Enablement Wizard.

Note: The transaction replay service-enabled mobile application must use business components that are created from the base application's JAR file because both the mobile client application and the base application deployed to the server must share the same entity objects in the application module so that the transactions generated from the mobile applications can be replayed successfully against the base application.

Before you begin:

Only applications whose Model projects contain the **ADF Mobile Client Model** technology can be enabled for transaction replay service; this technology enables the transaction replay service logging mechanism for the application. You can verify if an application includes this technology by opening the Project Properties dialog for a Model project and then selecting **ADF Mobile Client**. As shown in Figure 5–19, JDeveloper notes that the project lacks the **ADF Mobile Client Model** technology required for transaction replay service.

Figure 5–19 ADF Mobile Client Project Properties



When you create the entities for an ADF Mobile client application using the Create Business Components from ADF Library wizard as described in Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects," the entity objects imported by this wizard are enabled for synchronization because they have been given `CREATE`, `UPDATE`, and `DELETE` actions.

You should have a connection that has a transaction replay service schema deployed to it. If such a connection does not exist, then the wizard prompts you to create one.

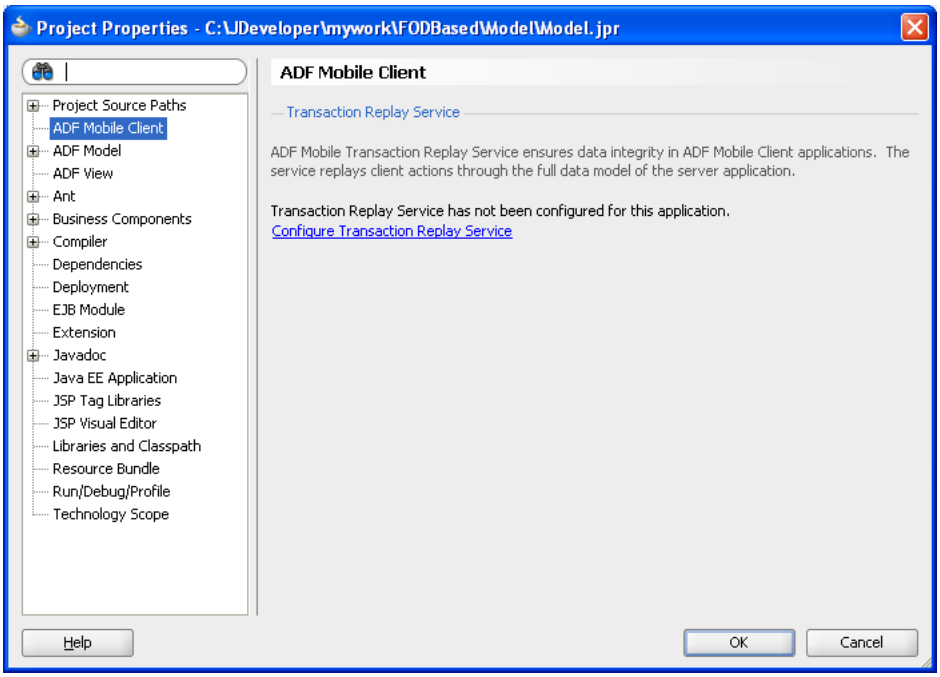
To add the transaction replay service to an ADF Mobile client application:

1. In the Application Navigator, right-click the Model project.
2. In Project Properties, choose **ADF Mobile Client**.

3. Click **Configure Transaction Replay Service** and then click **OK**.

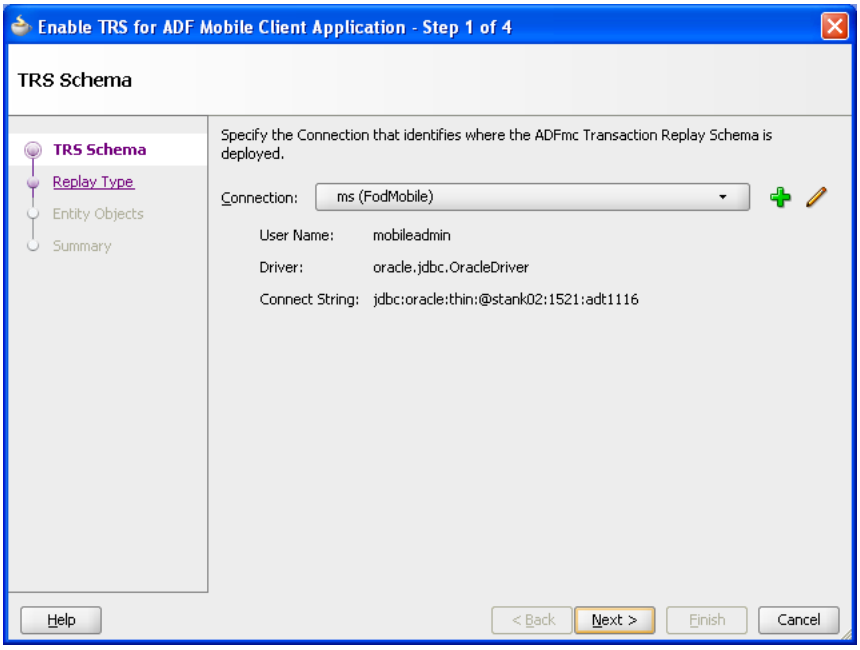
   As shown in Figure 5–20, if **Configure Transaction Replay Service** displays, then the entity objects are not transaction replay service-enabled.

Figure 5–20 ADF Mobile Client Project Properties



4. In the TRS Schema page, shown in Figure 5–21, select an IDE connection or an application connection. The transaction replay service schema must already be deployed to this connection.

Figure 5–21 Selecting the ADF Mobile Transaction Replay Service Schema Connection



If the schema is not in the selected connection, you must create a new connection. Click **Next**. The wizard proceeds after it verifies that the selected connection contains the transaction replay service schema.

5. In the Replay Type page, shown in Figure 5–22, enter the replay type information. The wizard defines the replay type by reviewing the imported entity object data from the base (server) application and then populates the **Type Name** field with a suggested name. You can override this value with any name as long as it is unique in the `REPLAY_TYPE` table in the transaction replay service schema. The page also enables you to designate how transaction replay service manages passwords.

Figure 5–22 The Replay Type Page

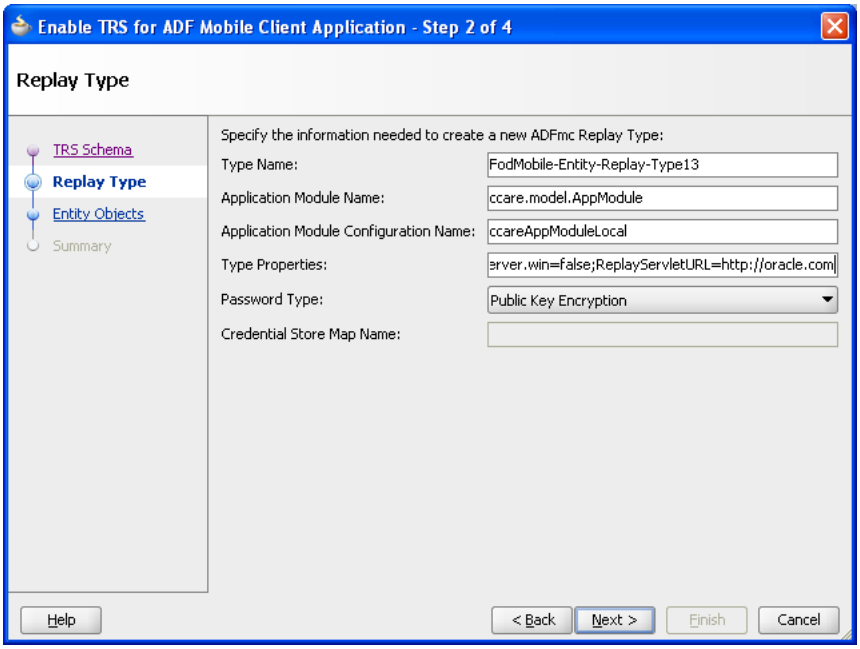


Table 5–4 describes the properties of the Replay Type page.

Table 5–4 Defining the Reply Type

Property	Description
Type Name	If needed, enter a name for the replay type or accept the name populated by the wizard.
Application Module Name	Enter the application module of the base (server) application. Open the base server application's model project and locate the application module. Include the application module package and application module name. For example, if the package is *model* and the name is *AppModule*, enter *model.AppModule*. You can find these properties on the General page of the overview editor for the application module.

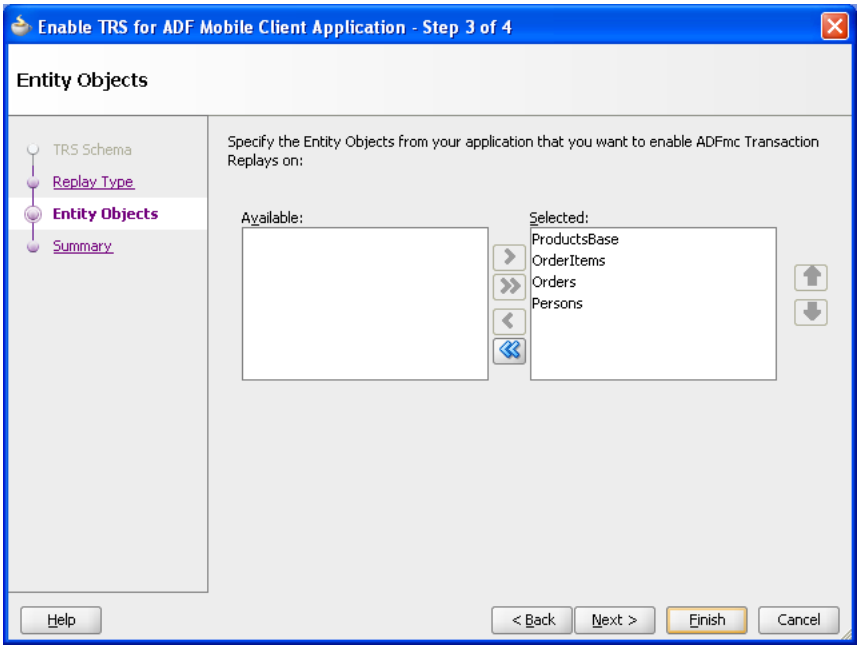
Table 5–4 (Cont.) Defining the Reply Type

Property	Description
Application Module Configuration Name	Enter the configuration name of the base (server) application module. You can select the application module configuration name using the Manage Configurations dialog. To access this dialog: 1. Open the base (server) application in JDeveloper. 2. In the Application Navigator, right-click the application module. 3. Choose **Configurations** from the context menu. 4. Choose the configuration from the Names column. 5. Enter this configuration name in the **Application Module Configuration Name** field.
Type Properties	This field contains a group of name-value pairs that you can configure for a replay type. This field has the format of `name1=value1;name2=value2;name3=value3`. The ADF Entity Event type has the following two properties: 1. Name: `server.win` Values: `true` (the default) or `false` (optional) 2. Name: `ReplayServletURL` Value: The URL to the replay servlet set up by adding the **ADF Mobile Transaction Replay** technology as described in Section 5.8.1, "How to Add the ADF Mobile Transaction Replay Service Technology to an ADF Application." This is a required value. Example: `server.win=false;ReplayServletURL=http://test.example.com:7001/CCare-ViewController-context-root/replayservlet`
Password Type	Select from among the following password types: ■ Public Key Encryption (PKE) ■ Credential Store Framework (CSF). A credential store is a repository for user name-password or generic credentials. When a credential store is used, an application does not store its passwords in clear text or invent solutions for protecting passwords. The Credential Store Framework (CSF) used by Oracle Platform Security Services (OPSS) is a set of APIs that enable secure `Create`, `Read`, `Write`, and `Update` operations. For more information, see "Configuring the Credential Store" in *Oracle Fusion Middleware Security Guide*. If you select this option, then you must also enter the name of the CSF map for the credentials. ■ No Password Type—Select this option if neither PKE nor CSF apply. Use this option for testing only. Do not select this option for a live deployment.
Credential Store Map Name	If you select **Credential Store Framework** as the means by which transaction replay service retrieves the password, then you must enter the name of the map for the credentials.

Note: Although you can change the name of the replay type, you cannot rename the underlying application references that are needed to successfully replay events from the client to the server.

6. Click **Next**.

Figure 5–23 Selecting ADF Mobile Transaction Replay Service Entity Objects



The Entity objects page displays all of the entity objects defined in the ADF Mobile client Model project. As illustrated in Figure 5–23, all entity objects are selected by default. Transaction replay service events (`INSERT`, `UPDATE`, and `DELETE`) are enabled for each selected entity object. Select the entity objects designated for transaction replay service by moving the entity objects from the **Selected** window to the **Available** window. Click **Next**.

Note: You can enable or disable transaction replay service for entity objects using the overview editor for entity objects. For more information see Section 5.3.7, "About Synchronization for Entity Objects."

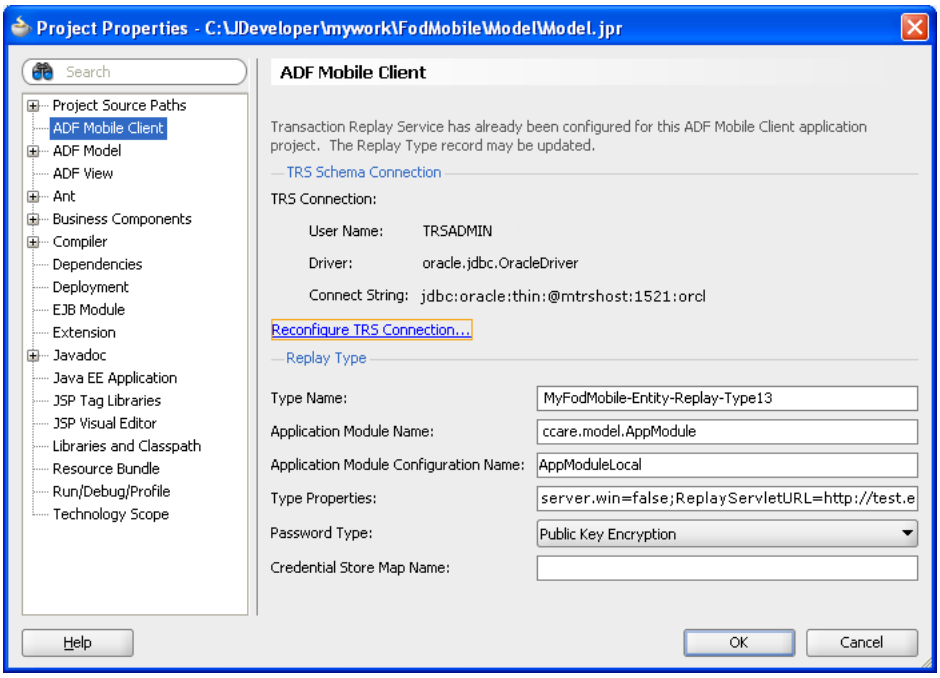
7. Review the information displayed on the Summary page. To make changes to the replay type, click **Back**. Otherwise, click **Finish**.

5.8.5 What Happens When JDeveloper Creates a Transaction Replay Type

When you complete the wizard, the ADF Mobile Client page is updated with the information configured for the transaction replay type. Figure 5–24 shows this page. By clicking **Reconfigure TRS Connection**, you can access the TRS Enablement Wizard starting at the TRS schema page. You can update the transaction replay type's current connection to the ADF Mobile transaction replay service schema, or select a new one.

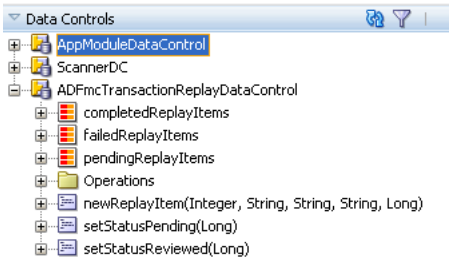
In addition, you can edit the replay type properties and entity objects using the wizard's pages.

Figure 5–24 ADF Mobile Client Page (After Creation of a Replay Type)



In addition, JDeveloper updates `adf-config.xml`, writes the TRS replay type for the identified server application to the REPLAY_TYPE table in the transaction replay service Schema, and sets appropriate application, project, and entity object properties to reflect that transaction replay service has been enabled. As shown in Figure 5–25, JDeveloper also creates a transaction replay service data control named `ADFmcTransactionReplayDataControl`. JDeveloper overwrites this data control each time you complete the wizard.

Figure 5–25 The Transaction Replay Service Data Control



You can use this data control to create a view for reviewing Replay Items. Creating such a view enables you to review any conflicts that occurred during a prior synchronization with the server.

You do not need to use the transaction replay service data control to log the standard Replay Items that are generated from the ADF business components used by the mobile client entity objects if you enabled transaction replay service for the entity objects as described in Section 5.3.7.1, "How to Enable or Disable Synchronization for Entity Objects." The ADF Mobile client framework handles this logging automatically for the transaction replay service-enabled entity objects. If you create a mobile

application that uses transaction replay service to invoke some business logic outside of ADF Business Components, then you can use the transaction replay service data control to manually log those Replay Items.

5.9 Authentication

When an application contacts the ADF Mobile transaction replay service server through the `AuthenticationManager` for the first time, it downloads an RSA public key and stores it in the application database. When a user successfully authenticates through `AuthenticationManager` for the first time, the user name and password are stored in the application database in two forms: first, an SHA1 hash of the password is stored so that the application can authenticate this user in the future if the device does not have network connectivity; second, `AuthenticationManager` uses the RSA public key to encrypt the password and stores this encrypted password in the database.

As the user runs the application, it generates transaction replay service records for replay on the server. When the application synchronizes these records to the server, the user name and encrypted password are sent to the server so that the transactions can be executed in the user's server security context. The server has the private key to decrypt the user's password and the private key is never downloaded to the mobile client.

ADF Mobile client provides authentication through the interaction of the `AuthenticationManager` class and the following EL expressions.

- `"#{securityContext.userName}"`
- `"#{securityContext.password}"`
- `"#{securityContext.loggingIn}"`
- `"#{securityContext.statusMessage}"`
- `"#{securityContext.hasError}"`
- `"#{securityContext.errorMessage}"`

The `userName` and `password` expressions must be set to enable a user's application to use the `AuthenticationManager` class; otherwise backing beans cannot use them. Other EL expressions are optional. For more information, see Section 5.9.2, "What You May Need to Know About SecurityContext EL Expressions."

5.9.1 What You May Need to Know About the AuthenticationManager Class

Example 5–6 illustrates a mobile client login page containing components that use EL expressions. For more information, see Section 5.9.2.1, "Using EL Expressions for Authentication."

Example 5–6 A Mobile Client Login Page

```
<?xml version='1.0' encoding='windows-1252'?>
<amc:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:amc="http://xmlns.oracle.com/jdev/amc">
    <amc:form id="form1">
    <amc:menuControl refId="altMain" />
            <amc:panelGroupLayout id="home" layout="vertical">
            <amc:panelGroupLayout layout="vertical">
                <amc:inputText id="txtUserName" label="User Name: "
value="#{securityContext.userName}" />
```

```
                <amc:inputText id="txtPassword" label="Password: "
value="#{securityContext.password}" secret="true" />
            </amc:panelGroupLayout>
            <amc:panelGroupLayout layout="horizontal">
            <amc:commandButton disabled="#{securityContext.loggingIn}"
actionListener="#{LoginBean.onLogin}" text="Login" />
            <amc:commandButton disabled="#{!securityContext.loggingIn}"
actionListener="#{LoginBean.onCancel}" text="Cancel" />
             </amc:panelGroupLayout>
                <amc:outputText id="lblStatus" value="Status:
#{securityContext.statusMessage}" rendered="#{securityContext.loggingIn}" />
              <amc:outputText id="lblError" value="#{securityContext.errorMessage}"
rendered="#{securityContext.hasError}" foregroundColor="#FF0000" />
            </amc:panelGroupLayout>
         </amc:form>
         <amc:menu id="altMain" type="alt" platform="wm">
        <amc:menuGroup id="menuGroup1" index="200">
                  <amc:commandMenuItem label="Exit" index="0" action="appExit"/>
        </amc:menuGroup>
         </amc:menu>
</amc:view>
```

Example 5–7 illustrates a backing bean called LoginBean that uses the `AuthenticationManager` class.

Example 5–7 Sample LoginBean

```
package view.backing;

import oracle.adfnmc.el.util.BeanResolver;
import oracle.adfnmc.model.security.AuthenticationManager;
import oracle.adfnmc.model.security.CredentialsCache;
import oracle.adfnmc.util.Utility;


public class LoginBean
  extends BeanResolver
{
  private static final String HOME_ACTION = "toHome"; // defined in the task flow
definition files, typically task-flow-definition.xml

  private AuthenticationManager m_mgr;

  public LoginBean()
  {
  }

  private synchronized AuthenticationManager getAuthenticationManager()
  {
    if (m_mgr == null)
    {
      m_mgr = new AuthenticationManager();
      m_mgr.setNavigationAction(HOME_ACTION);
      m_mgr.setWebServiceUrl("<enter value here>");
    }
    return m_mgr;
  }

  private void onLogin()
  {
```

```
    AuthenticationManager mgr = getAuthenticationManager();

    String replayTypeName = "<enter value here>";
    mgr.authenticate(replayTypeName);
  }

  private void onCancel()
  {
    AuthenticationManager mgr = getAuthenticationManager();
    mgr.cancel();
  }

  public Object invokeMethod(String methodName, Object[] params)
  {
    if (methodName.equals("onLogin"))
    {
      this.onLogin();
    }
    else if (methodName.equals("onCancel"))
    {
      this.onCancel();
    }

    return null;
  }
}
```

As illustrated in Example 5–7, the `authenticate` method of the `AuthenticationManger` class is called and returns immediately after a user clicks a login button. The bean's `onLogin` and `onCancel` methods are hooked to the action listeners (such as `actionListener="#{LoginBean.onLogin}" text="Login"` and `actionListener="#{LoginBean.onCancel}" text="Cancel"` in Example 5–8). The `AuthenticationManager` class spawns a background authentication thread that checks if the user has the permission to update a transaction replay service replay type. If authentication succeeds, it calls a `NavigationAction`, which is set in the mobile client task flow. In this example, the `NavigationAction` leads to the home page.

> **Note:** You must set the `String ReplayTypeName` variable to the same value entered in the Type Name field of the transaction replay service wizard's Replay Type Page. For example, you would enter *MyFodMobile-Entity-Replay_Type13*, the value shown in Figure 5–22.

5.9.1.1 Public Accessors

Table 5–5 lists the public accessors of the `AuthenticationManager` class.

> **Note:** Typically, you change only `NavigationAction`.

Table 5–5 Public Accessors

Name	Type	Default Value	Description
`NavigationAction`	String	None. The value must be set before the `authenticate` method is called	Denotes the navigation action that runs after a successful login
`WebServiceAuthenticationMethodName`	String	`validateCredential`	Denotes a web service method that is used for credential validation
`WebServiceNameSpace`	String	`http://webservice.tnxreplay.oracle/`	Denotes the web service namespace that is used for web service calls
`WebServicePublicKeyMethodName`	String	`getPublicKey`	Denotes a web service method that is used for public key retrieval
`WebServiceUrl`	String	None; The value must be set before the `authenticate` method is called.	Denotes a web service URL that is used
`Callback`	AuthenticationCallback	null	Denotes a callback used for success or failure of a login

5.9.1.2 Public Methods

Table 5–6 lists the public methods of the `AuthenticationManager` class.

Table 5–6 Public Methods

Method	Usage
`clearPublicKey`	Call this method when the server's public key changes. Although servers seldom change their public keys, this method clears the public key to enable the `AuthenticationManager` class to request the public key when the user next tries to authenticate.
`authenticate`	The `AuthenticationManager` class calls the `authenticate` method when a user wants to authenticate credentials. Before this method is called, the values must be set to `#{securityContext.userName}` and `#{securityContext.password}`. The authentication process is as follows: ■ For a user who has never logged in before: If the device is offline, then an error is raised that indicates that the device is offline. If the device is online, then the public key is requested and the credentials are validated with the web service. ■ For a user who has logged in previously: If the device is offline, then the credentials are validated against the cached credentials. If the device is online, then the credentials are validated against the web service.
`cancel`	Call this method when a user wants to terminate the current authentication process.

5.9.1.3 The AuthenticationCallback Class

The callback method is:

```
public void response(int value, Exception e);
```

The current `int` values are passed in:

```
            /**
             * This denotes a successful login
             */
            public static final int AUTH_SUCCESS = 0;

            /**
             * This denotes a general failure
             */
            public static final int AUTH_FAILURE = 1;

            /**
             * This denotes a failure due to invalid credentials
             */
            public static final int AUTH_FAILURE_CREDENTIALS = 2;

            /**
             * This denotes a failure due to the device being offline with no
stored credentials
             */
            public static final int AUTH_FAILURE_OFFLINE = 3;

            /**
             * This denotes a failure resolving the URL
             */
            public static final int AUTH_FAILURE_INVALID_URL = 4;
```

5.9.2 What You May Need to Know About SecurityContext EL Expressions

Write to the expressions listed in Table 5–7 and read from the expressions listed in Table 5–8.

Table 5–7 EL Expressions (Written To)

Name	Type	Expression	Description
UserName	String	`#{securityContext.userName}`	This must be set before `AuthenticationManager.authenticate()` is called. This value is passed to the web service as the user name to be authenticated.
Password	String	`#{securityContext.password}`	This must be set before `AuthenticationManager.authenticate()` is called. This value is passed to the web service as the password to be authenticated. This value is cleared automatically on a successful login to prevent it from getting read later in the application.

Table 5–8 lists the EL expressions that are read from.

Table 5–8 EL Expressions (Read From)

Name	Type	Expression	Description
LoggingIn	Boolean	`#{securityContext.loggingIn}`	This expression is set to true when `AuthenticationManager.authenticate()` is called. It remains set to true until there is a successful login or a failure. You can use this expression to swap the disabled nature of a Login/Cancel CommandButton pair based on the current action. For example: `<amc:commandButton disabled="#{securityContext.loggingIn}" actionListener="#{LoginBean.onLogin.execute} " text="Login" /> <amc:commandButton disabled="#{!securityContext.loggingIn}" actionListener="#{LoginBean.onCancel.execute }" text="Cancel" />`
Status Message	String	`#{securityContext.statusMessage}`	This expression is updated when `AuthenticationManager.authenticate()` is called for the different steps that are taken to authenticate the user. Use this expression to populate an `OutputText` component that updates users on the current status of the authentication. For example: `<amc:outputText id="lblStatus" value="Status: #{securityContext.statusMessage}" rendered="#{securityContext.loggingIn}" />`
HasError	Boolean	`#{securityContext.hasError}`	This expression is set to false when `AuthenticationManager.authenticate()` is called and remains set to false unless a login failure occurs. Use this expression for `OutputText` components that render only when there is an error. For example: `<amc:outputText id="lblError" value="#{securityContext.errorMessage}" rendered="#{securityContext.hasError}" foregroundColor="#FF0000" />`
Error Message	String	`#{securityContext.errorMessage}`	The expression is updated when `AuthenticationManager.authenticate()` results in an error. Use this expression when creating an `OutputText` component that notifies the user of an error. For example: `<amc:outputText id="lblError" value="#{securityContext.errorMessage}" rendered="#{securityContext.hasError}" foregroundColor="#FF0000" />`

5.9.2.1 Using EL Expressions for Authentication

Example 5–8 illustrates an mobile client login page containing components with some of the EL expressions listed in Table 5–7 and Table 5–8. In lines 7 and 8, the `UserName`

and `Password` EL expressions are set. Setting `secret=true` in line 8, hides the password entered in the user interface by making it appear as a series of asterisks (*). In lines 11 and 12, the `LoggingIn` EL expression is used to alternately disable the login button or enable the cancel button. Lines 14 and 15 illustrate the Status Message and Error Message expressions.

Example 5–8 Sample Mobile Client Login Page Using EL Expressions

```
1: <?xml version='1.0' encoding='windows-1252'?>
2:   <amc:view>
3:      <amc:form xmlns:amc="http://xmlns.oracle.com/jdev/amc">
4:      <amc:menuControl refId="altMain" />
5:      <amc:panelGroupLayout id="home" layout="vertical">
6:      <amc:panelGroupLayout layout="vertical">
7:         <amc:inputText id="txtUserName" label="User Name: " value="#{securityContext.userName}"
/>
8:         <amc:inputText id="txtPassword" label="Password: " value="#{securityContext.password}"
secret="true" />
9:      </amc:panelGroupLayout>
10:      <amc:panelGroupLayout layout="horizontal">
11:      <amc:commandButton disabled="#{securityContext.loggingIn}"
          actionListener="#{LoginBean.onLogin}" text="Login" />
12:      <amc:commandButton disabled="#{!securityContext.loggingIn}"
          actionListener="#{LoginBean.onCancel}" text="Cancel" />
13:      </amc:panelGroupLayout>
14:      <amc:outputText id="lblStatus" value="Status: #{securityContext.statusMessage}"
         rendered="#{securityContext.loggingIn}" />
15:      <amc:outputText id="lblError" value="#{securityContext.errorMessage}"
      rendered="#{securityContext.hasError}" foregroundColor="#FF0000" />
16:      </amc:panelGroupLayout>
17:      </amc:form>
18:      <amc:menu id="altMain" type="alt" platform="wm">
19:      <amc:menuGroup index="200">
20:      <amc:commandMenuItem label="Exit" index="0" action="appExit"/>
21:      </amc:menuGroup>
22:      </amc:menu>
23:   </amc:view>
```

6

Creating the ADF Mobile Client User Interface

This chapter describes how to build ADF Mobile client user interfaces for Windows Mobile devices and BlackBerry smartphones using ADF Mobile client components.

This chapter includes the following sections:

This chapter describes the following:

- Design and development of the flow of the user interface.
- Creation of MCX pages.
- Features of ADF Mobile client components that are unique to the mobile client.
- Usage of the Expression Language.
- Usage of binding layer components.

6.1 Introduction to Creating the ADF Mobile Client User Interface

ADF Mobile client provides a set of layout and field components that enable you to create applications that behave appropriately for both the BlackBerry and Windows Mobile user experience. While the mobile client 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: the mobile client components do not support every property and behavior of ADF Faces components. In essence, the mobile client components represent wrappers around native components in BlackBerry and Windows Mobile, with their appearance and behavior being very similar to the ADF Faces components.

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

For more information, see the following:

- Chapter 4, "Getting Started with ADF Mobile Client"
- Chapter 5, "Developing the ADF Mobile Client Data Model"

6.2 Creating Task Flows

Using your application workspace (see Section 4.4, "Creating an Application Workspace"), you start creating the user interface for your application by designing task flows. As with any standard JSF application, ADF Mobile client applications use navigation cases and rules to define the task flow. These definitions are stored in the `MobileClient-task-flow.xml` file (see Section 6.2.6, "What You May Need to Know About the MobileClient-task-flow.xml File").

For Oracle Fusion Middleware 11*g* release 1 of ADF Mobile client, you can create mobile applications that have only bounded task flows. As described in "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. Bounded task flows have a single entry point and zero or more 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. Other features of bounded task flows include accepting input parameters and generating return values.

You use the Mobile Client Task Flow Designer to create bounded task flows for mobile client applications (mobile client task flows). When designing a mobile client task flow, JDeveloper maintains the same experience as designing an ADF task flow, as described in "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 6.2.7, "What You May Need to Know About the Mobile Client Task Flow Diagrammer") in which you build the task flow by dragging and dropping activities and control flows from the Component Palette. You then define these activities and the transitions between them using the Property Inspector.

6.2.1 How to Create a Task Flow

You use the navigation diagrammer to declaratively create a task flow. When you use the diagrammer, JDeveloper creates the XML metadata needed for navigation to work in your application in the `MobileClient-task-flow.xml` file (default). To use the task flow other than the default on the application startup, you need to manually modify the value of the following setting in the `adf-config.xml` file:

```
<amc:setting name="root-task-flow" value="MobileClient-task-flow.xml"/>
```

> **Note:** Each project must have one root task flow. For more information, see Table 10–1, " ADF Mobile Client Framework Settings".

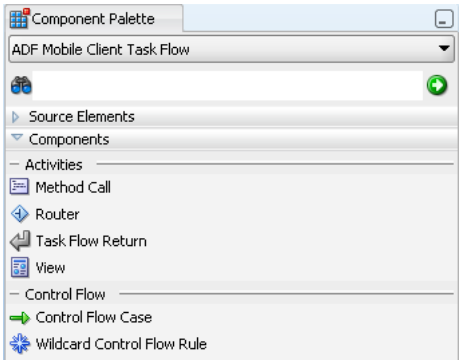
Before you begin:

To design a task flow, the ADF Mobile application must include a view controller project file (that is, a project that includes the ADF Mobile UI technology), which includes the `MobileClient-task-flow.xml` file. If the project does not include this file, or if the project requires an additional flow, you can create one using the Mobile Client Task Flow Creation Wizard as described in Section 6.2.3, "How to Use the Mobile Client Task Flow Creation Wizard."

To create a task flow:

1. Open the `MobileClient-task-flow.xml` file for your application. By default, this is in the **Application Sources** node.
2. In the editor window, click the **Diagram** tab to open the navigation diagrammer.
3. If the Component Palette is not displayed, from the main menu choose **View > Component Palette**. By default, the Component Palette is displayed in the upper right-hand corner of JDeveloper.
4. In the Component Palette dropdown list, choose **ADF Mobile Client Task Flow**.

Figure 6–1 ADF Mobile Client Task Flow Palette in JDeveloper



5. Select the component you wish to use and drag it onto the diagram.

   JDeveloper redraws the diagram with the newly added component.

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

Additionally, you can manually add elements to the `MobileClient-task-flow.xml` file by directly editing the page in the source editor. To view the file in the source editor, click the **Source** tab.

Once the navigation for your application 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 6.2.11, "How to Enable Page Navigation Using Control Flow Case."

After you define the task flow for the application, you can double-click a view file to access the MCX view. For more information, see Section 6.3, "Creating Mobile Views."

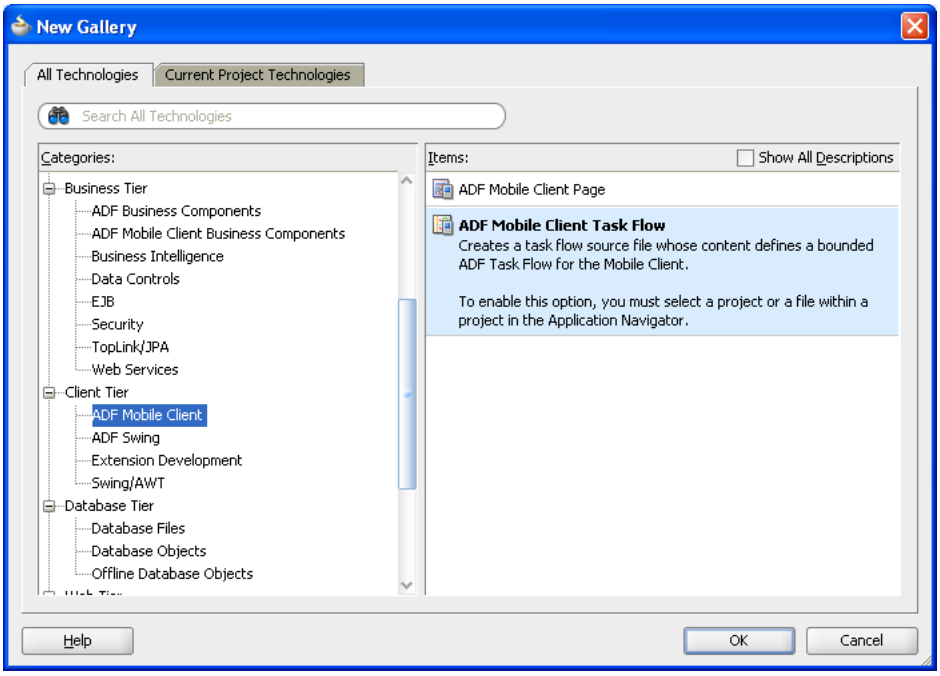
6.2.2 How to Create an Additional Task Flow

JDeveloper automatically creates a bounded mobile client task flow (named `MobileClient-task-flow.xml` by default) when you select the Mobile Client UI technology for the project. You can create additional bounded task flows using the task flow dialog.

To create a bounded task flow:

1. In the main menu, select **File**, and then **New**.
2. In New Gallery, expand the **Client Tier** node, and then select **ADF Mobile Client**.
3. From the Items list, select **ADF Mobile Client Task Flow**, as Figure 6–2 shows, and then click **OK**.

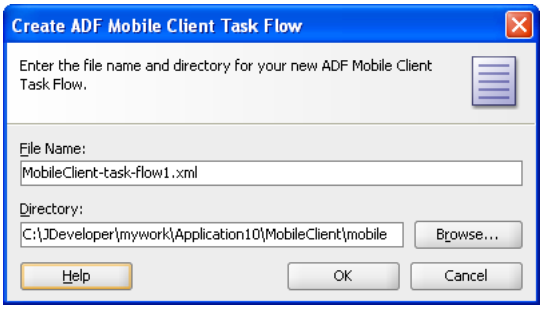
Figure 6–2 Creating a Mobile Client Task Flow



4. Complete the Create ADF Mobile Client Task Flow dialog by adding the following:
   - Enter a name for the task flow. By default, JDeveloper names this file `MobileClient-task-flow.xml`.
   - Enter the directory path.
5. Click **OK**.

   Figure 6–3 shows the Create ADF Mobile Client Task Flow dialog. JDeveloper increments the number of the task flow according to the number of task flows that already exist in the same pattern. For example, Figure 6–3 shows a task flow named `MobileClient-task-flow1.xml`.

Figure 6–3 Create ADF Mobile Client Task Flow Dialog



6.2.3 How to Use the Mobile Client Task Flow Creation Wizard

Although JDeveloper creates the mobile client task flow automatically when you assign the Mobile Client technology to a project, you can also manually create a mobile client task flow using the Mobile Client Task Flow Creation Wizard.

Before you begin:

Because the task flow files are included in the view controller project, the ADF Mobile client application must include the view controller project file that results from the selection of the ADF Mobile UI technology (see Section 4.4.1, "How to Create an Application Workspace.")

To create a mobile client task flow:

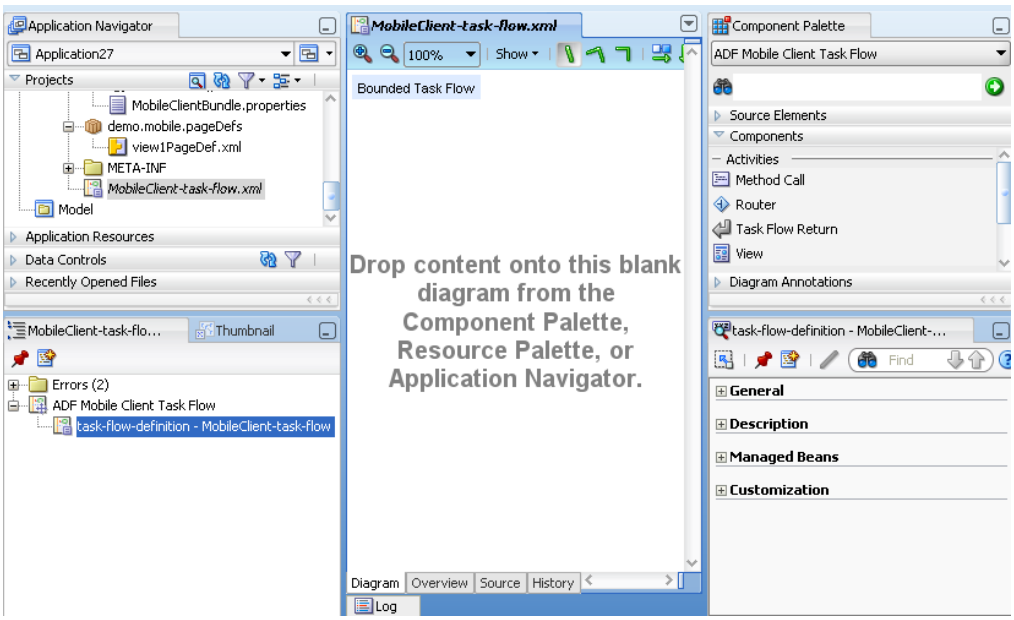
1. In the Application Navigator, select the project in which you want to create the mobile client task flow, and then choose **New** to open the New Gallery dialog.
2. Select the **Current Project Technologies** tab, and then expand the **Client Tier** node in the **Categories** list.
3. Select **ADF Mobile Client**, and then select **ADF Mobile Client Task Flow** from the Items list.
4. Enter the name of the task flow (the default value is `MobileClient-task-flow.xml`) and, if needed, enter the directory for the task flow. If an application already includes a mobile client task flow, the wizard will automatically give the task flow a unique name within the application. For example, the task flow that Figure 6–5 shows is called `MobileClient1-task-flow.xml`.
5. Click **OK**.

6.2.4 What Happens When You Create a Mobile Client Task Flow

JDeveloper creates a bounded task flow, which by default is called `MobileClient-task-flow.xml`. You can populate this task flow by dragging ADF Mobile Client Task Flow components into the diagrammer.

Figure 6–4 shows the Diagram tab selected, revealing the diagrammer. ADF Mobile Client Task Flow is selected in the Component Palette.

Figure 6–4 The Bounded Task Flow in the Diagrammer



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

The mobile client task flow designer supports a subset of ADF activities and control flows.

Table 6–1 lists the activities supported for mobile client task flows.

Table 6–1 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 application to invoke application logic based on control flow rules. For additional information, see "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 in a task flow at design time. These include standard ADF parameters for a method call action in a mobile client 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 mobile client 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 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 "Using Router Activities" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
View	Displays an MCX page. You can create an MCX page by double-clicking the view activity. For more information, see "Using View Activities" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
Task Flow Return	In mobile client applications, this activity is used as the exit Figure 6–5 shows a task flow return named *Exit*.

Table 6–2 lists the control flows supported by mobile client task flows.

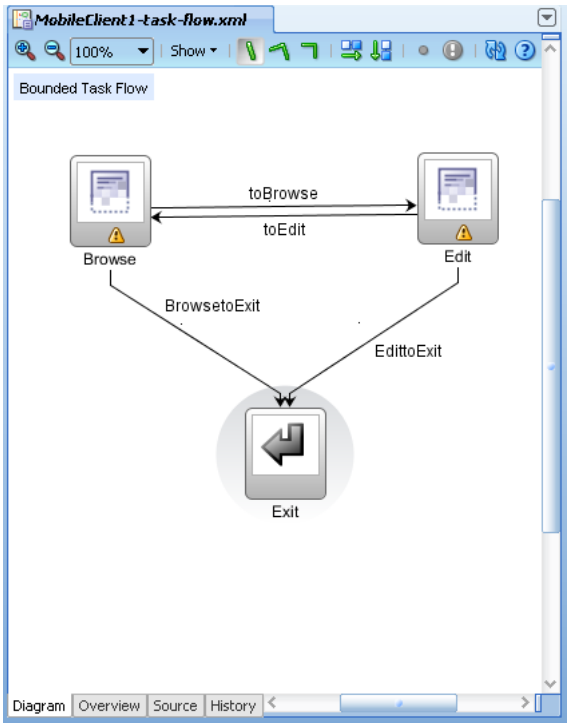
Table 6–2 Supported Control Flows

Control Flows	Description
Control Flow Case	Identifies how control passes from one activity to the next in an application. For more information, see "Control Flows" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
Wildcard Control Flow Rule	Represents a control flow case that can originate from any activity with an ID matching a wildcard expression. For more information, see "How to Add a Wildcard Control Flow Rule" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

6.2.6 What You May Need to Know About the MobileClient-task-flow.xml File

As described in Section 4.4.2, "What Happens When You Create a Mobile Client Application Workspace," adding the Mobile Client technology to the project results in the addition of the `MobileClient-task-flow.xml` file. This file is the mobile client counterpart to `task-flow-definition.xml`, and lets you design the interactions between views (MCX pages) by dragging and dropping the mobile client task flow components from the Component Palette into the diagrammer. The `MobileClient-task-flow.xml` file (shown in Figure 6–5), which JDeveloper generates as a result of selecting the ADF Mobile UI technology for the project, is the source file for creating task flows for mobile client applications.

Figure 6–5 The MobileClient-task-flow.xml File



You use the Property Inspector to define the components in the mobile client task flow in the same manner as you would using `task-flow-definition.xml` or `faces-config.xml`.

6.2.7 What You May Need to Know About the Mobile Client Task Flow Diagrammer

As illustrated in Figure 6–5, the task flow diagram and Component Palette display automatically after you create a task flow using the Mobile Client Task Flow Creation Wizard. The task flow diagram is a visual editor into which you can drag and drop activities and task flows from the Component Palette or from the Application Navigator.

For more information, see the following:

- Section 6.2.3, "How to Use the Mobile Client Task Flow Creation Wizard"
- Section 6.2.9, "How to Add View Activities"

6.2.8 How to Add Mobile Client Activities

After you create a mobile client task flow, the task flow diagrammer and Component Palette automatically display. As in ADF application development, this task flow diagrammer is the visual editor onto which you drag and drop activities, views, and control flows.

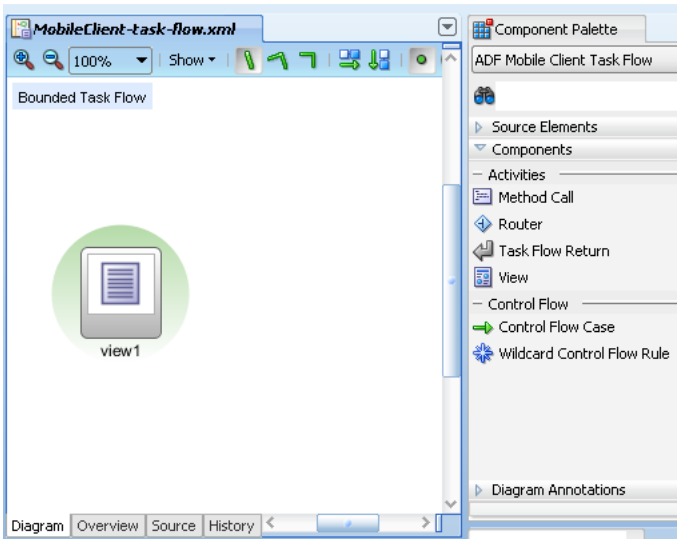
Before you begin:

You must select **ADF Mobile Client Task Flow** from the Component Palette as Figure 6–6 shows.

To add an activity to a mobile client task flow:

1. In the Application Navigator, double-click a task flow source file (`MobileClient-task-flow.xml`) to display the task flow diagram and the Component Palette, as Figure 6–6 shows. The diagrammer displays the task flow editor. The Component Palette automatically displays the components available for a mobile client task flow.

Figure 6–6 The Diagrammer for the Task Flow Editor



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 MCX File wizard.

6.2.9 How to Add View Activities

The view activity is associated in metadata with an actual MCX page. You add a view activity by dragging and dropping a view activity from the Component Palette. A view activity displays an MCX page. You can create an actual MCX page by double-clicking the view activity in the Diagram window. You can also create a view activity by dragging and dropping an MCX file in the Application Navigator into the overview editor's Diagram tab.

6.2.10 How to Add a Wildcard Control Flow Rule

Mobile client 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.

6.2.11 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 Mobile client 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 "What Happens When You Create a Control Flow Rule" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

6.3 Creating Mobile Views

You start creating mobile views by doing the following:

- Getting familiar with the MCX page structure
- Using the visual editor
- Dragging and dropping components into the MCX page
- Adding data controls to a view

6.3.1 How to Work With MCX Pages

The MCX page is represented by an XML file similar to a JSPX file in ADF Faces. The view layer of a mobile client application is persisted in this file.

6.3.1.1 Interpreting the MCX Page Structure

The following is a basic structure of the MCX file:

```
<amc:view>
   <amc:form/>
   <amc:menu/>
    …
```

```
   <amc:menu/>
</amc:view>
```

For more information, see Section 6.3.1.3, "What Happens When You Create an MCX Page."

6.3.1.2 Creating MCX Pages

MCX files are contained in the MobileClient project. You create these files using the Create ADF Mobile Client Page dialog.

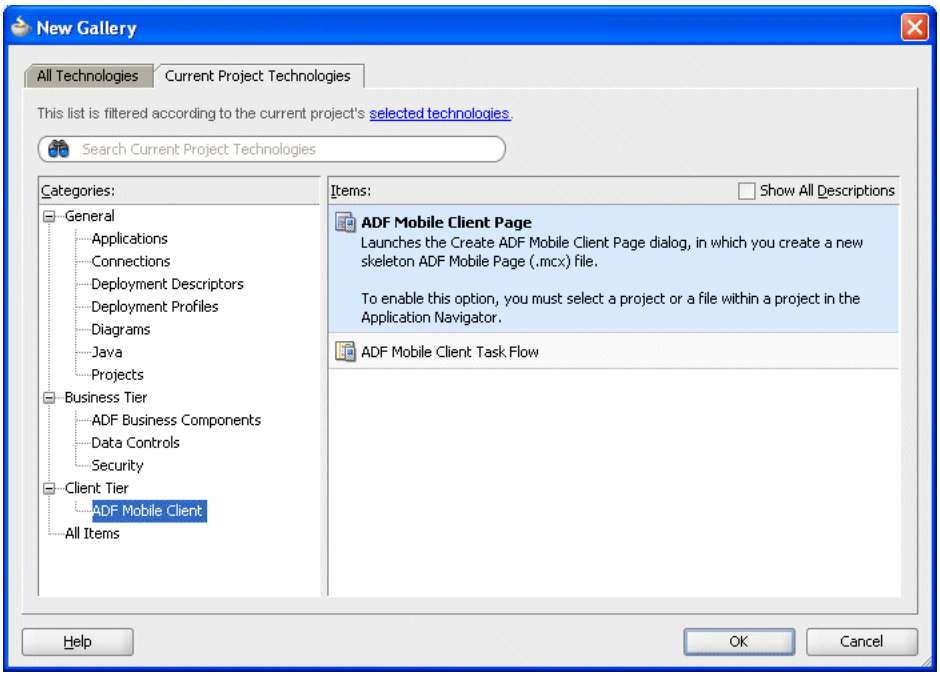
Before you begin:

The ADF Mobile client application must include a MobileClient project (that is, a project that includes the ADF Mobile UI technology).

To create an MCX page:

1. In the Application Navigator, right-click the directory where you would like the page to be saved, and choose **New**.
2. In the New Gallery, expand the **Client Tier** node, select **ADF Mobile Client** and then **ADF Mobile Client Page** and click **OK**.

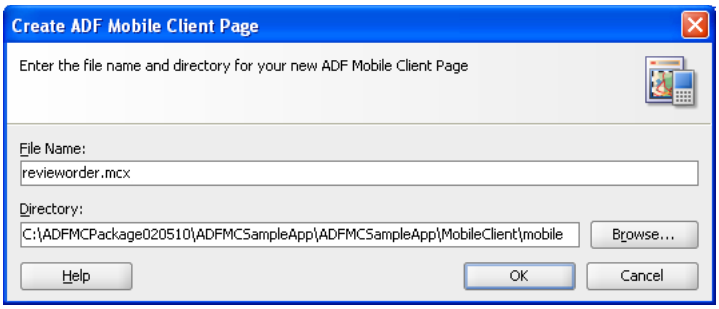
Figure 6–7 Creating a Mobile Client Page



Tip: Alternatively, you can also create an MCX page by double-clicking a view icon in the task flow editor for a page that has not yet been created.

3. In the Create ADF Mobile Client Page dialog, enter a name and, if needed, a location. For help, click **Help** in the dialog. Click **OK**.

Figure 6–8 Create ADF Mobile Client Page Dialog

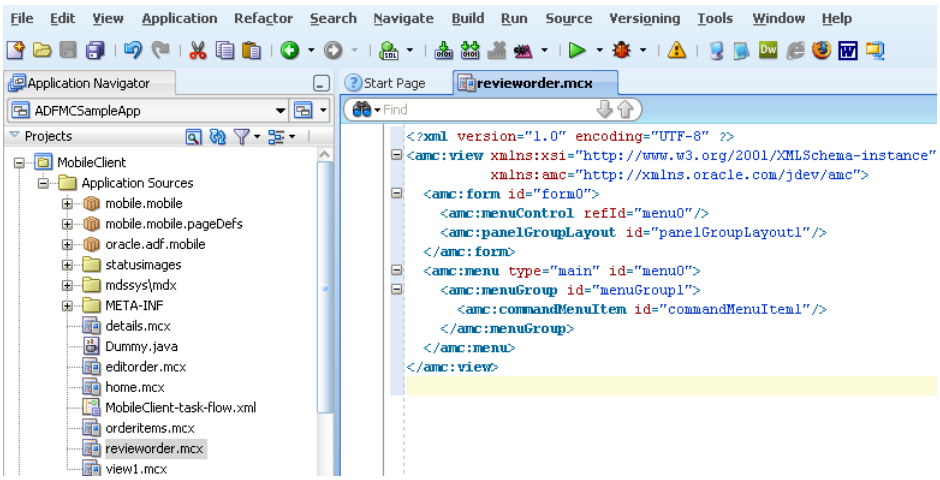


6.3.1.3 What Happens When You Create an MCX Page

When you use the Create ADF Mobile Client Page dialog to create an MCX page, JDeveloper creates the physical file and adds it to the `mobile` directory of the MobileClient project.

In the Application Navigator that Figure 6–9 shows, the `mobile` node contains a newly created MCX file called `revieworder.mcx`.

Figure 6–9 The MCX File in the Application Navigator



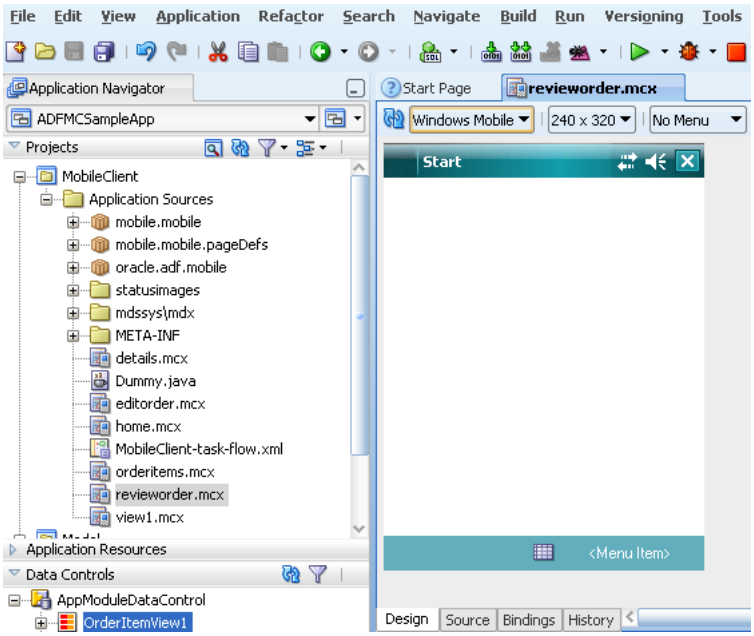
JDeveloper also adds the code necessary to import the component libraries and display a page. This code is illustrated in the source editor shown in Figure 6–9.

Example 6–1 Declarative Page Source Created by JDeveloper

```
<?xml version="1.0" encoding="UTF-8" ?>
<amc:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amc="http://xmlns.oracle.com/jdev/amc">
  <amc:form id="form0">
    <amc:menuControl refId="menu0"/>
    <amc:panelGroupLayout id="panelGroupLayout1"/>
  </amc:form>
  <amc:menu type="main" id="menu0">
    <amc:menuGroup id="group1">
      <amc:commandMenuItem id="commandMenuItem1"/>
    </amc:menuGroup>
  </amc:menu>
</amc:view>
```

When the page is first displayed in JDeveloper, it is displayed in the visual editor (accessed by clicking the **Design** tab), which allows you to view the page in a WYSIWYG environment. Figure 6–10 shows the Preview tab selected for a newly created MCX page called `revieworder.mcx`. This page is blank because it has not yet been populated with ADF Mobile client components or data controls.

Figure 6–10 The Visual Editor for a Newly Created MCX Page



You can also view the source for the page in the source editor by clicking the **Source** tab, as shown in Figure 6–9. The Structure window located in the lower left-hand corner of JDeveloper (also shown in Figure 6–10), provides a hierarchical view of the page. For more information, see Section 6.3.2.2, "Using the Visual Editor."

6.3.2 How to Add Mobile Client Components and Data Controls to an MCX Page

After you create an MCX page, you can start adding the mobile client UI components and data controls to your page.

6.3.2.1 Adding UI Components

You can use the Component Palette to drag and drop components onto the page. JDeveloper then adds the necessary declarative page code and sets certain values for component attributes.

For information on adding and using specific mobile client components, see the following:

- Section 6.4, "Designing the Layout of the Page"
- Section 6.5, "Creating and Using Input Components"
- Section 6.6, "Creating and Using Output Components"
- Section 6.7, "Displaying Images"
- Section 6.8, "Creating and Using Tables"

- Section 6.9, "Using Buttons and Links"
- Section 6.10, "Creating and Using Scanners"

Before you begin:

The ADF Mobile application must include a MobileClient project (that is, a project that includes the ADF Mobile UI technology). This project must also contain an MCX page, or an ADF Mobile task flow from which to create a page.

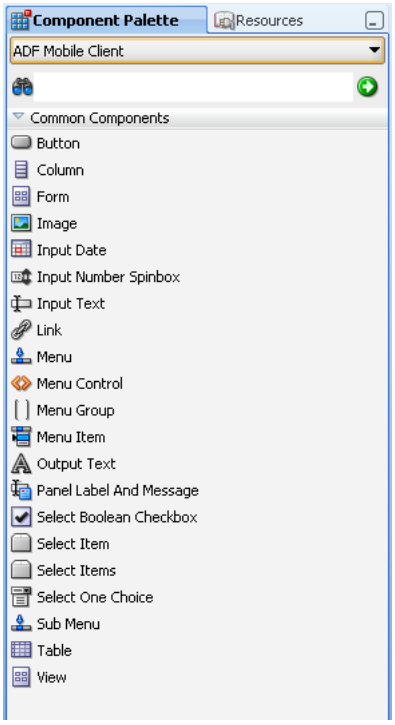
As described in Section 6.3.1.2, "Creating MCX Pages," you can invoke the Create ADF Mobile Client page dialog by double-clicking a view icon in a navigation diagram or by selecting the **MobileClient** project, then selecting **ADF Mobile Client** in the New Gallery and then selecting **ADF Mobile Client Page**.

To add mobile client components to a page:

1. Open an MCX page in the visual editor.
2. In the Component Palette, use the menu to choose **ADF Mobile Client** (see Figure 6–11).

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

Figure 6–11 ADF Mobile Client Component Palette



3. Select the component you wish to use, and then drag and drop it onto source editor, page designer, or structure window.

Note: When building an MCX page, you can only drop UI components into UI containers such as a Panel Group Layout, Panel Form Layout, and Panel Label and Message.

JDeveloper redraws the page in the visual editor with the newly added component. In the visual editor, you can directly select components on the page and use the resulting context menu to add more components.

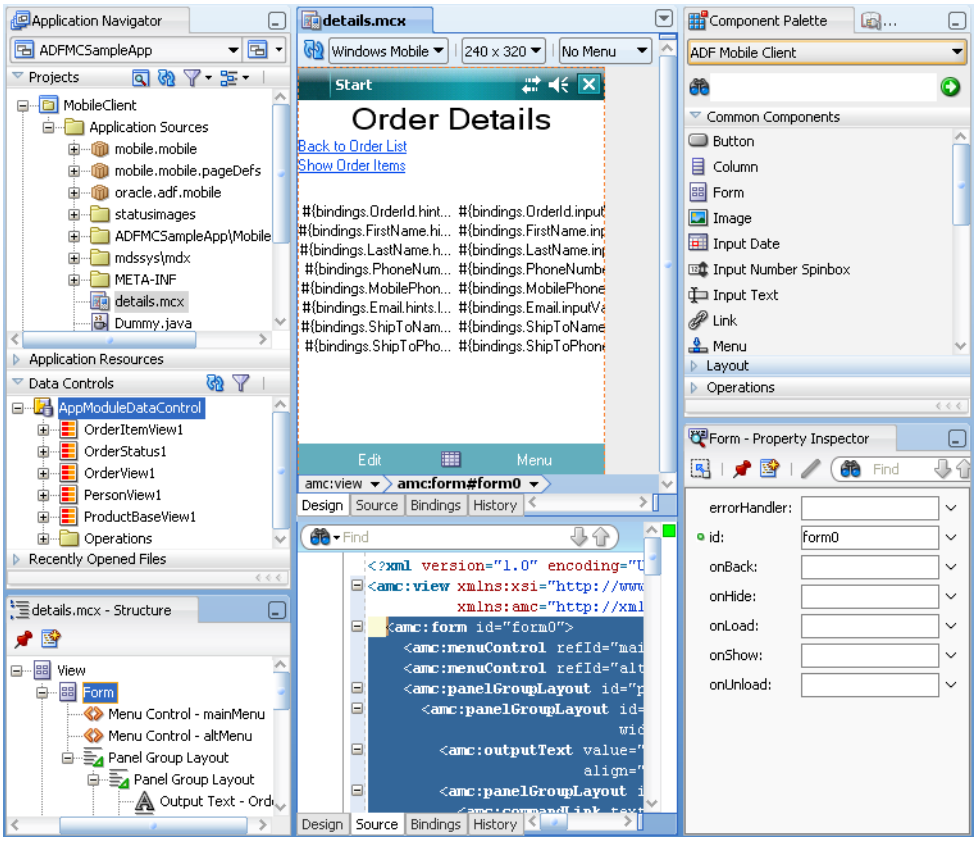
Tip: You can also drag and drop components from the palette into the Structure window or directly into the code in the source editor.

You can always add components 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.

6.3.2.2 Using the Visual Editor

JDeveloper's editor provides WYSIWYG support for both the Windows Mobile and BlackBerry platforms when you build views using MCX files. As illustrated in Figure 6–12, splitting a view while adding the mobile client components to the MCX file enables you to see both the code view through the source editor and a mobile client UI view through the visual interactive designer represented by the Design tab. As a result, you can modify the source view and get instant feedback in terms of the look and feel of that application on both the BlackBerry and Windows Mobile platforms.

Figure 6–12 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.

Besides providing the means to drag and drop components onto the MCX page, the designer lets you interact with the components directly, therefore eliminating the need to use the Structure pane:

- When you select a component in the Designer, this component is outlined in all visible renderers.
- When you move the mouse over a component, you can see the highlight in the current renderer.
- You can select a component using a breadcrumb bar that shows the hierarchy of components on the current page. Note that the breadcrumb bar appears below the Design tab, and selecting a component in this bar has the same effect as selecting the component in the Structure pane.
- When you right-click a component in the designer, the standard context menu is displayed. This menu is identical to the context menu of the Structure window.

> **Note:** The MCX page is rendered even for an invalid MCX 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.

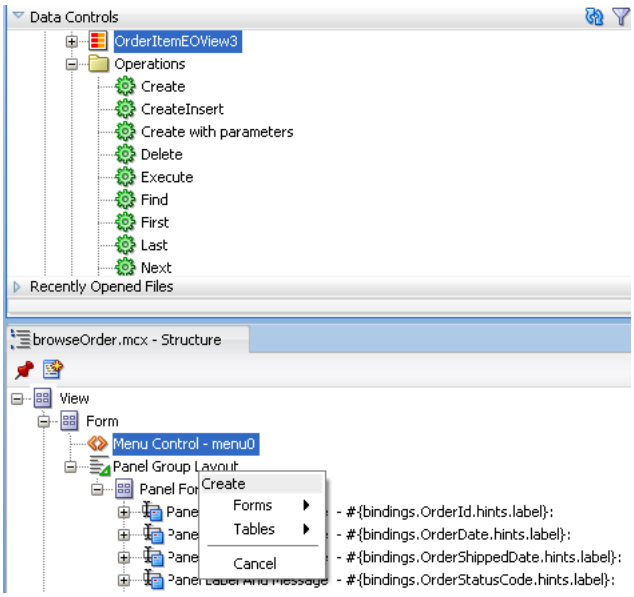
6.3.2.3 Adding Data Controls to the View

You can create databound UI components in an MCX 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 into an MCX page. In addition, JDeveloper creates the binding information in the associated page definition file. If no such file exists, then JDeveloper creates one.

Depending on the approach you take, you can insert different types of data controls into the Structure window of an MCX page. For example, dropping a collection such as a View object enables you to create a Form or a Table.

Figure 6–13 shows the context menu for creating forms or tables that appears when you drag a View object (such as `OrderItemEOView3` in Figure 6–13) in to the Structure window of an MCX page.

Figure 6–13 Creating a Form or Table from a View Object



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 mobile client 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 ADF Faces application.

The following scenarios describe the specific controls that result from dropping an entire collection or from dropping selected attributes into the Structure window or the MCX page's source file.

Scenario 1

In this scenario, you populate a Panel Form Layout based on the View objects in the model layer by performing the following:

1. Create a new MCX page. Drag a Panel Form Layout component from the Component Palette into the page's Panel Group Layout.
2. From the Data Controls panel, select a collection such as a View object, and then drop it into the Structure window. A context menu appears that presents options to create a form or a table.
3. Select either form or table. A wizard appears that describes all of the View object attributes that can be added to the MCX page as well as the type of mobile client components that can be created for each of the fields.
4. Select the appropriate attributes and change the component as needed.
5. Click **OK**. JDeveloper adds all of the controls to the MCX page and adds the required EL expressions.

Scenario 2

In this scenario, you add only specific attributes of a View object by performing the following:

1. Create an MCX page.

2. From the Data Controls panel, expand a collection such as a View object.
3. Select an attribute, and then drop it into the Structure window of the MCX source file. Depending on the type of attribute that you select, a context menu appears that presents the appropriate components.

   Table 6–3 lists the selection components, which are available when you select **Single Selections**.

Table 6–3 Selection-Related Components

Component	Description	EL Expressions

Select One Choice	Creates a mobile client combo box with a label and with value properties	`label="#{bindings.FieldName.label}"`
		`value="#{bindings.FieldName.inputValue}"`
		`(selectOneChoice)`
		
		`value="#{bindings.FieldName.items}"`
		`(nested selectItems)`

Select Boolean Checkbox	Creates a mobile client check box with a label and with a value properties.	`value="#{bindings.FieldName.inputValue}"` `text="#{bindings.FieldName.label}"`

Selecting **Texts** in the context menu enables you to create text controls by populating the page with the components listed in Table 6–4.

Table 6–4 Text-Related Components

Component	Description	EL Expressions

Output Text with Label	Creates a mobile client Panel Label and Message component that contains an Output Text component.	`label="#{bindings.FieldName.hints.label}"`
		`(on panelLabelAndMessage)`
		
		`value="#{bindings.FieldName.inputValue}"`
		`(on outputText)`

Output Text	Creates a mobile client output text value property.	`value="#{bindings.FieldName.inputValue}"`
Input Text with Label	Creates a mobile client input text with label and value properties.	`label="#{bindings.FieldName.hints.label}"` `value="#{bindings.FieldName.inputValue}"`
Input Text	Creates a mobile client input text with value property.	`value="#{bindings.FieldName.inputValue}"`

If you select a `date` attribute, then the context menu presents the components listed in Table 6–5 when you select **Date**.

Table 6–5 Date Attribute-Related Components

Component	Description	EL Expression

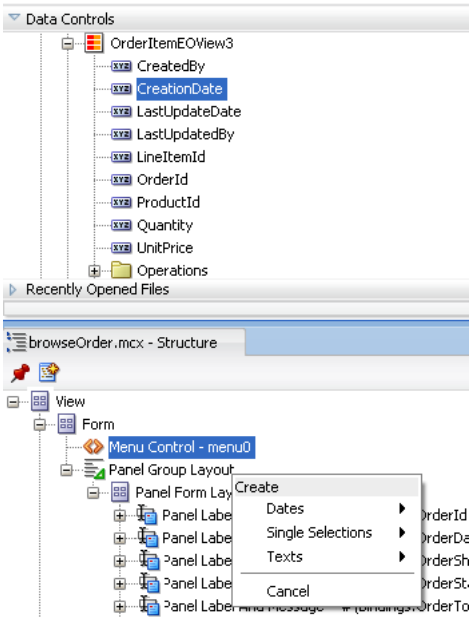
Input Date with Label	Creates a mobile client input date with a label and value property.	`value="#{bindings.FieldName.inputValue}" label="#{bindings.FieldName.hints.label}" required="#{bindings.FieldName.hints.mandatory}" (inputDate)`
		
		`pattern="#{bindings.FieldName.format}" (nested convertDateTime)`

Input Date	Creates a mobile client input date with a value property.	`value="#{bindings.FieldName.inputValue}" required="#{bindings.FieldName.hints.mandatory}" (inputDate)`
		
		`pattern="#{bindings.FieldName.format}" (nested convertDateTime)`

4. Select the component. JDeveloper adds the controls to the MCX page and adds the required EL expressions as well.

   Figure 6–14 shows the context menu for adding date, single selection, and text controls that appears when you drag an attribute from the Data Controls panel into the Structure window of an MCX page. As shown in Figure 6–14, dragging a date-related attribute (*CreationDate*) results in a context menu that includes the Dates selection, one that enables you to create date input components that Table 6–5 describes.

Figure 6–14 Adding Controls from Attributes



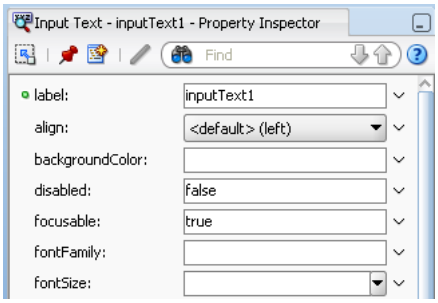
6.3.2.4 Configuring UI Components

Once you drop 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 **View > Property Inspector** from the main menu.

Figure 6–15 shows the Property Inspector displaying the attributes for an Input Text component.

Figure 6–15 The Property Inspector



To set component attributes:

1. Select the component for which you want to set attributes. You can select the component either in the visual editor or the Structure window, or you can select its tag directly in the source editor.
2. 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 "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.

6.4 Designing the Layout of the Page

ADF Mobile client provides layout components (listed in Table 6–6) that let you arrange 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 6–6 ADF Mobile Client Page Management, Layout, and Spacing Components

Component	Type	Description
Form	Page Management Component	Creates a `form` element in an MCX file. For more information, see Section 6.4.1, "How to Use a Form Component." For more information about MCX files, see Section 6.3.1.2, "Creating MCX Pages."
Panel Form Layout	Page Layout Container	Creates a `panelFormLayout` element in an MCX file. Positions components, such as Input Text components, so that their labels and fields line up horizontally. For more information, see Section 6.4.3, "How to Use a Panel Form Layout Component."
Panel Label And Message	Page Layout Container	Creates a `panelLabelAndMessage` element in an MCX file. Lays out a label and its children. For more information see Section 6.4.4, "How to Use a Panel Label And Message Component."
Panel Group Layout	Page Layout Container	Creates a `panelGroupLayout` element in an MCX file. Groups child components either vertically or horizontally. For more information, see Section 6.4.2, "How to Use a Panel Group Layout Component."
Spacer	Spacing Component	Creates an area of blank space represented by a `spacer` element in an MCX file. For more information, see "Separating Content Using Blank Space or Lines" section in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

With the exception of the Form component, you add a layout component by dragging and dropping it onto the MCX page from the Component Palette (see Section 6.3.2.1, "Adding UI Components"). Then you use the Property Inspector to set the component's attributes (see Section 6.3.2.4, "Configuring UI Components"). For information on attributes of each particular component, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

6.4.1 How to Use a Form Component

A Form is a component that serves as a container for other components.

By default, when you create an MCX page, JDeveloper automatically creates and inserts a Form component with its containing Panel Group Layout component (see Section 6.4.2, "How to Use a Panel Group Layout Component") into the page. When you add components to the page, they will be inserted inside the Form component (`form` element in an MCX file).

6.4.1.1 How to Add a Form to a Page

You do not have to explicitly add the Form component to an MCX page, as JDeveloper will add it for you. Note that each page must contain one Form component.

6.4.2 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. The Form component must have one Panel Group

Layout child component. 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 MCX 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 arrange the child components in the desired layout by setting the **layout** property. For more information, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

   You can also change horizontal and vertical alignments. For more information, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.
5. In the **Property Inspector**, set the remaining attributes of this component. For more information, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

6.4.2.1 What You May Need to Know About Geometry Management and the Panel Group Layout Component

The geometry behavior of ADF Mobile client user interface components depends on whether they are layered on a Panel Group Layout component with its `layout` attribute set to `vertical` (default) or `horizontal`.

6.4.2.1.1 Geometry Management and Vertical Panels A vertical panel places components in vertical direction in one column.

Typically, the height of the panel grows as components are added to the panel, whereas the width of the panel is defined by the widest component in the panel. However, if the `height` attribute is set, the panel will take on the specified height. If the `width` attribute is set, the component will take on the lesser of the specified width and the width made available by its parent container. This behavior is consistent when scrolling is enabled. If horizontal scrolling is enabled and the `width` attribute is not set, then the panel is as wide as the lesser of the width of its widest component and the width made available by the parent container.

As a general rule, if no size is specified in a vertical Panel Group Layout, it will be as high as the sum of the heights of its child components, and as wide as the widest component.

The default width for such components as an Input Date and Input Text is the width of the visible part of their parent container and not the maximum available width for the entire screen.

A vertical Panel Group Layout ignores the `verticalAlign` attribute of its child components.

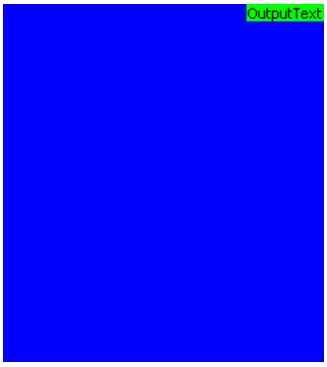
The following examples show how the combination of various settings of the Panel Group Layout and its child components affect the run-time display. Note that in all of the examples, the Panel Group Layout has a blue `backgroundColor`, while the Output Text child component has a green `backgroundColor`.

Example 6–2 Defining Full-Width Vertical Panel Group Layout with Right-Aligned Child Component

```
<amc:panelGroupLayout id="home" layout="vertical"
                      backgroundColor="#0000FF" width="100%"
                      height="100%" scrollable="true">
   <amc:outputText id="ot1" value="OutputText"
                   align="right" backgroundColor="#00FF00"/>
</amc:panelGroupLayout>
```

Figure 6–16 shows a Panel Group Layout component that fills up the whole screen area, since its `width` and `height` are set to 100%. At run time, the Output Text is right-aligned.

Figure 6–16 Full-Width Vertical Panel Group Layout with Right-Aligned Child Component at Run Time

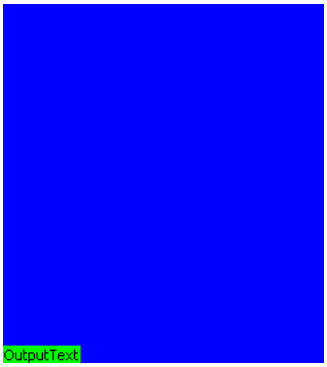


Example 6–3 Defining Full-Width Horizontal Panel Group Layout with Bottom-Aligned Child Component

```
<amc:panelGroupLayout id="home" layout="horizontal"
                      backgroundColor="#0000FF" width="100%"
                      height="100%" scrollable="true">
   <amc:outputText id="ot1" value="OutputText"
                   verticalAlign="bottom" backgroundColor="#00FF00"/>
</amc:panelGroupLayout>
```

Figure 6–17 shows a Panel Group Layout component that fills up the whole screen area, since its `width` and `height` are set to 100%. At run time, the Output Text is bottom-aligned.

Figure 6–17 Full-Width Horizontal Panel Group Layout with Bottom-Aligned Child Component at Run Time



Example 6–4 Defining Vertical Panel Group Layout of the Size of Its Child Component

```
<amc:panelGroupLayout id="home" layout="vertical" backgroundColor="#0000FF">
   <amc:outputText id="ot1" value="OutputText"
                   align="right" backgroundColor="#00FF00"/>
</amc:panelGroupLayout>
```

Figure 6–18 shows a Panel Group Layout component that is not visible, since it expands only enough to accommodate its child component. At run time, only the Output Text is visible for the Panel Group Layout is the same size as its child component.

Figure 6–18 Vertical Panel Group Layout of the Size of Its Child Component at Run Time

Example 6–5 Defining Vertical Panel Group Layout That Is Wider than Its Child Component

```
<amc:panelGroupLayout id="home" layout="vertical"
                      width="200" backgroundColor="#0000FF">
   <amc:outputText id="ot2" value="OutputText"
                   align="right" backgroundColor="#00FF00"/>
</amc:panelGroupLayout>
```

Figure 6–19 shows a Panel Group Layout component that is set to be wider than its child component. At run time, the Output Text is right-aligned with extra space to the left.

Figure 6–19 Vertical Panel Group Layout That Is Wider than Its Child Component at Run Time

Example 6–6 Defining Horizontal Panel Group Layout of the Size of Its Child Component

```
<amc:panelGroupLayout id="home" layout="horizontal" backgroundColor="#0000FF">
   <amc:outputText id="ot3" value="OutputText"
                   verticalAlign="bottom" backgroundColor="#00FF00"/>
</amc:panelGroupLayout>
```

Figure 6–20 demonstrates the same layout behavior as Figure 6–18 the Panel Group Layout component is not visible, since it expands only enough to accommodate its child component. At run time, only the Output Text is visible for the Panel Group Layout, as this component is the same size as its child component.

Figure 6–20 Horizontal Panel Group Layout of the Size of Its Child Component at Run Time

Example 6–7 Defining Horizontal Panel Group Layout That Is Higher than Its Child Component

```
<amc:panelGroupLayout id="home" layout="horizontal"
                      height="50" backgroundColor="#0000FF">
   <amc:outputText id="ot4" value="OutputText"
                   verticalAlign="bottom" backgroundColor="#00FF00"/>
</amc:panelGroupLayout>
```

Figure 6–21 shows a Panel Group Layout component that is set to be higher than its child component. At run time, the Output Text is right-aligned with extra space above it.

Figure 6–21 Horizontal Panel Group Layout That Is Higher than Its Child Component at Run Time

Horizontal Scrolling in Vertical Panels

By default, horizontal scrolling is disabled, but you can enable it. When horizontal scrolling is disabled, the vertical panel attempts to lay out all components within the visible width of the panel. When it is not possible to lay out a component within the visible width, it extends of the screen to the right. The behavior of components on a vertical panel varies depending on the component type.

On BlackBerry smartphones, when horizontal scrolling is enabled in a vertical panel, the vertical panel scrolls in a horizontal direction when the focusable component that was initially obscured becomes activated.

Vertical Scrolling in Vertical Panels

By default, vertical scrolling is automatically enabled on BlackBerry devices on a vertical panel when the bottom edge of the component at the bottom exceeds the maximum visible height of the panel. Note that the vertical scrolling takes place in the panel that is the panel closest to the root panel in the hierarchical order and that is enabled for vertical scrolling. Therefore, no local scrolling is performed unless there is no containing panel enabled for vertical scrolling.

Note: Scrolling (both vertical and horizontal) is not enabled by default on Windows Mobile devices. When enabled, vertical scrolling takes place on the Panel Group Layout on which the `verticalScroll` attribute is set to `true`.

You can explicitly disable vertical scrolling of parts of the view while enabling localized vertical scrolling by setting appropriate attributes. When vertical scrolling of the root panel, as well as that of all the ancestors of a vertical panel, is disabled, the panel scrolls in vertical directions within the panel.

You can also completely disable vertical scrolling on a view by explicitly disabling vertical scrolling on all panels including the main view.

Alignment in Vertical Panels

A vertical panel can be aligned to left (default), center, or right with respect to the current applicable width of the current column.

For example, suppose there is a vertical panel whose `width` attribute is set to 200. The panel contains two components: Button `commandButton1` that is 100 pixels wide, and Button `commandButton2` that is 150 pixels wide. Since the right horizontal alignment is specified on both contained components, both are right-aligned within the panel and there is an empty space to the left of both of them.

If in the preceding example the panel's width is not set, the panel would be 150 pixels wide, because it assumes the width of the widest contained component

(`commandButton2`). When the contained components are laid out on the panel, there would not be any empty space to the left of the `commandButton2`.

Note that the horizontal alignment of the panel does not affect the horizontal alignment of the contained UI components.

Vertical alignment is ignored on vertical panels.

Nesting Vertical Panels

A vertical panel can be nested within any panel, whether it is vertical or horizontal. A vertical panel can contain any panel.

The width of a vertical panel is equal to the widest component in the panel when the `width` attribute is not specified. When the `width` attribute is set, then the panel takes on the smaller of the specified width or the available width provided by its parent container. Vertical panels behave in the same way with respect to height.

At the same time, a vertical panel lays out its contained panels indifferent from the way it would lay out other components.

Overlapping Vertical Panels

It is not possible for two panels or two components within a panel to overlap.

Spacing Between Panels

To create gaps between panels or components, use the Spacer component

6.4.2.1.2 Geometry Management and Horizontal Panels A horizontal panel places components in a horizontal direction in one row. Components contained in a horizontal panel may be multiline, but no component is placed on top of another.

The width of the panel when scrolling is enabled is the lesser of the total width occupied by its children or 1073741823 (0x3FFFFFFF) pixels. The height of the panel is defined by the highest component in the panel. If a horizontal panel contains a component whose width is equal to the available width, the component does not occupy the entire available width of a horizontally scrollable container by default. This behavior is consistent regardless of the enablement of horizontal scrolling.

Horizontal Scrolling in Horizontal Panels

By default, horizontal scrolling is disabled, but you can enable it. When horizontal scrolling is disabled, the horizontal panel attempts to lay out all components within the visible width of the panel. The available width is the lesser of the remaining (or provided) width of the parent container and the `width` specified on the panel.

This means that when a nonscrollable panel is embedded in a horizontally scrollable panel, the embedded panel's available width is 1073741823 (0x3FFFFFFF) pixels. This is because the very wide panel can be made visible through the scrolling on the underlying panel.

When it is not possible to lay out a component within the panel, the component extends to the right and it may become obscured and inaccessible. The behavior of components varies depending on the component type.

Some components are laid out differently when horizontal scrolling is enabled. However, if the component has a fixed width, such as a Button or an Output Text, the component is not stretched but maintains the same size regardless of the enablement of horizontal scrolling.

Some components display in different ways depending on the enablement of the horizontal scrolling in a horizontal panel. When horizontal scrolling is enabled, an input component can grow in width beyond the visible width. When horizontal scrolling is disabled, the text in an input component wraps and all text is made visible in most cases, possibly through vertical scrolling.

Vertical Scrolling in Horizontal Panels

By default, vertical scrolling is disabled, but you can enable it by setting the `verticalScroll` property to `true`.

Alignment in Horizontal Panels

Changing horizontal alignment on components in a horizontal panel has no effect. Changing vertical alignment on components in a horizontal panel shows an effect if the horizontal panel is taller than the components being aligned."

Nesting Horizontal Panels

A horizontal panel can be nested within any panel, whether it is a vertical or a horizontal panel. A vertical panel can contain any panel.

The geometry of a horizontal panel is calculated the containing panel as follows: when the `height` attribute is not specified, the height of a horizontal panel is the lesser of the highest component in the panel and the available height provided by the parent. When the `height` attribute is set, then the panel takes on the lesser of the specified height and the available height provided by its parent container. The width is the sum of the widths of the panel's children when the `width` attribute is not set. When the `width` attribute is set, the panel width is the lesser of the specified width and the sum of the widths of its children.

At the same time, a horizontal panel lays out its contained panels indifferent from the way it would lay out other components.

Overlapping Panels

It is not possible for two panels or two components within a panel to overlap.

Spacing Between Panels

To create gaps between panels or components, use the Spacer component

6.4.3 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 Input Date) and selection components (such as Select One Choice).

For more information, see Section 6.4.3.1, "What You May Need to Know About Geometry Management and the Panel Form Layout Component."

To add the Panel Form Layout component:

1. In the **Component Palette**, drag and drop a **Panel Form Layout** component to the MCX page.
2. In the **Property Inspector**, set the component's attributes. For more information, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

6.4.3.1 What You May Need to Know About Geometry Management and the Panel Form Layout Component

You can specify the `labelWidth` and the `fieldWidth` attributes as either pixels or percentages.

> **Note:** While you can specify the `labelWidth` and the `fieldWidth` attributes as either pixels or percentages, you cannot set one attribute as a percentage and the other in pixels. The `labelWidth` and `fieldWidth` attributes are disregarded at run time if you mix attribute definitions.

To specify the width as a percentage, set these `labelWidth` and `fieldWidth` attributes so that the sum of their combined percentages is 100. If only one of these width attributes is set as a percent, then ADF Mobile client calculates the width for the attribute that has not been set by subtracting the percentage set for the `width` attribute from 100. For example, if you set `"labelWidth=40%"`, but do not specify a value for `fieldWidth`, then the value for `fieldWidth` will be calculated as 60% at run time. If the combined values set for both `fieldWidth` and `labelWidth` do not total 100%, then these values are disregarded at run time.

If you specify the `labelWidth` and `fieldWidth` attributes in absolute terms and their length exceeds either the available width or the width of the Panel Form Layout component itself, then the value set for the `labelWidth` attribute takes precedence and the length of the `fieldWidth` attribute is truncated to the available width.

The display position of the Panel Form Layout component depends on the parent Panel Group Layout on which it is layered. The following geometry behavior of the Panel Form Layout is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Panel Form Layout is always positioned below the previous component and horizontally aligned according to the `align` attribute value of the Panel Form Layout.
- On horizontal panels: Panel Form Layout is always positioned to the right of the previous component and is vertically aligned according to the `align` attribute value of the Panel Form Layout component.

6.4.4 How to Use a Panel Label And Message Component

Use the Panel Label And Message (`panelLabelAndMessage`) component to lay out a label and its child, which is usually an output component (see Section 6.6, "Creating and Using Output Components").

To add the Panel Label And Message component:

1. In the **Component Palette**, drag and drop a **Panel Label And Message** component into a **Panel Group Layout** component.
2. In the **Property Inspector**, set the component's attributes. For more information, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

6.4.4.1 What You May Need to Know About Arranging Labels

You can use the Panel Form Layout to lay out multiple Panel Label And Message components. When you place Panel Label And Message components within a Panel Form Layout component, the labels align. If you place a component that already has a label within a Panel Label And Message component, then the component's label will

be displayed in the fields column at run time and the Panel Label And Message component's label is used in the label's column. To avoid this behavior, set the `simple` attribute to `true` on the Panel Label And Message's child component.

6.5 Creating and Using Input Components

You can use the following input components when developing your ADF Mobile client application:

- Input Text (see Section 6.5.1, "How to Use the Input Text Component")
- Input Date (see Section 6.5.2, "How to Use the Input Date Component")
- Input Number Spinbox (see Section 6.5.3, "How to Use the Input Number Spinbox Component")
- Select Boolean Checkbox (see Section 6.5.4, "How to Use the Select Boolean Checkbox Component")
- Select One Choice (see Section 6.5.5, "How to Use the Select One Choice Component")

You add an input component by dragging and dropping it onto the MCX page from the Component Palette (see Section 6.3.2.1, "Adding UI Components"). Then you use the Property Inspector to set the component's attributes (see Section 6.3.2.4, "Configuring UI Components"). For information on attributes of each particular component, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

You can add event listeners to ADF Mobile client's input components (see Section 6.5.6, "What You May Need to Know About Event Listeners and Input Components").

When creating these components, always consider their geometry behavior (see Section 6.5.1.1, "What You May Need to Know About Geometry Management and the Input Text Component").

6.5.1 How to Use the Input Text Component

The Input Text (`inputText`) component represents an editable field with an optional text label in front of it.

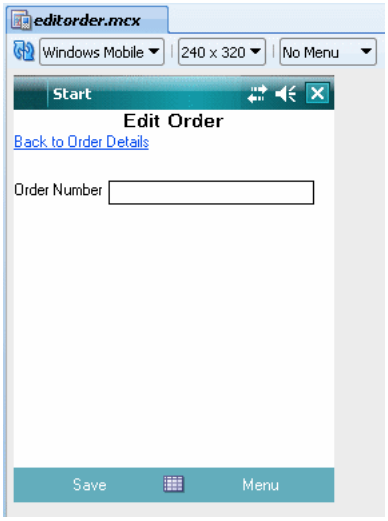
Example 6–8 demonstrates how to declare the `inputText` element in an MCX file.

Example 6–8 Creating Input Text

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
      <amc:panelGroupLayout id="panelGroupLayout3"
                     layout="vertical"
                     rendered="#{!applicationScope.addMode}"
                     width="100%">
      ...
         <amc:inputText label="Order Number"
                        id="inputText2"/>
      ...
```

Figure 6–22 shows the Input Text component in the Design page.

Figure 6–22 Input Text in Design Page



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 6.6.1.2, "Converting Numerical Values") and Convert Date Time (see Section 6.6.1.3, "Converting Date and Time Values") components.

When creating the Input Text component for the use on BlackBerry smartphones, note that the track wheel moves the focus caret in the main direction (usually vertical), whereas pressing ALT+trackwheel moves the focus caret in the less-used direction (usually horizontal).

You can use an Input Text component within the Panel Group Layout, Panel Form Layout, and Panel Label And Message components.

6.5.1.1 What You May Need to Know About Geometry Management and the Input Text Component

The default width of an Input Text component is 100%, which implies that it attempts to occupy the visible width of the parent container. Note that the visible width is not necessarily the same as the available width.

The display position of the Input Text component depends on the panel on which it is layered. The following geometry behavior of the Input Text is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Input Text is positioned according to the settings of the `width` and `align` attributes.
- On horizontal panels: Input Text is positioned according to the settings of the `width` attribute.

6.5.2 How to Use the Input Date Component

The Input Date (`inputDate`) component presents an 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`. However, you can override the format using a date-time converter (for more information about using converters, see Section 6.6.1.3, "Converting Date and Time Values").

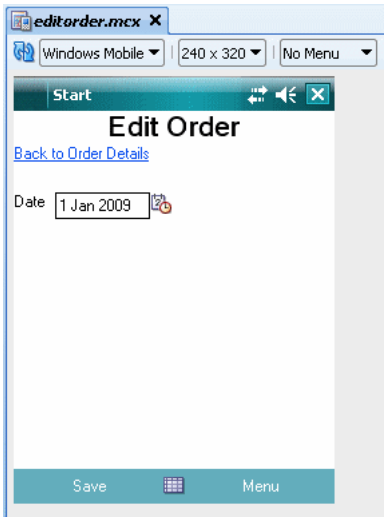
Example 6–9 demonstrates how to declare the `inputDate` element in an MCX file.

Example 6–9 Creating Input Date

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
      <amc:panelGroupLayout id="panelGroupLayout3"
                            layout="vertical"
                            rendered="#{!applicationScope.addMode}"
                            width="50%">
      ...
         <amc:inputDate value="1 Jan 2009"
                        label="Date "
                        required="#{bindings.OrderDate.hints.mandatory}"
                        id="inputDate2"/>
      ...
```

Figure 6–23 shows the **Date** Input Date component in the Design page of the visual editor.

Figure 6–23 Input Date in Design Page



The text label part of the Input Date component is always aligned to the left, while the edit field is aligned to right.

To enable conversion of date and time values that are entered in the Input Date component, you use the Convert Date Time component (see Section 6.6.1.3, "Converting Date and Time Values").

6.5.2.1 What You May Need to Know About Geometry Management and the Input Date Component

The display position of the Input Date component depends on the panel on which it is layered. The following geometry behavior of the Input Date is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Input Date is positioned according to the `width` attribute with the default value of 100%.
- On horizontal panels: Input Date is positioned according to the `width` attribute.

For more information, see Section 6.4.2.1, "What You May Need to Know About Geometry Management and the Panel Group Layout Component."

6.5.3 How to Use the Input Number Spinbox Component

The Input Number Spinbox (`inputNumberSpinbox`) is used for entering numbers and quickly stepping through the numbers. The input number must be within the range defined by its `minimum` and `maximum` attributes.

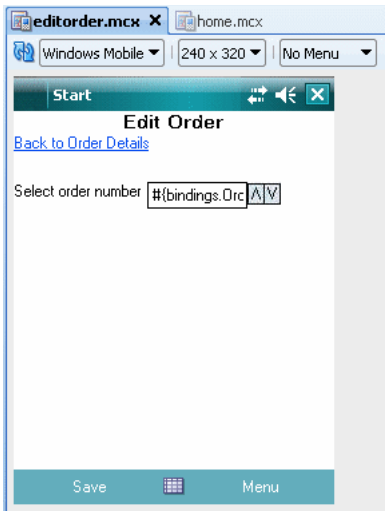
Example 6–10 demonstrates how to declare the `inputNumberSpinbox` element in an MCX file.

Example 6–10 Creating Input Number Spinbox

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
      <amc:panelGroupLayout id="panelGroupLayout3"
                            layout="vertical"
                            rendered="#{!applicationScope.addMode}"
                            width="100%">
      ...
         <amc:inputNumberSpinbox label="Select order number"
                        value="#{bindings.OrderId.inputValue}"
                        id="inputNumberSpinbox1" width="200"/>
      ...
```

Figure 6–24 shows the **Select order number** Input Number Spinbox component in the Design page of the visual editor.

Figure 6–24 Input Number Spinbox in Design Page



6.5.3.1 What You May Need to Know About Geometry Management and the Input Number Spinbox Component

The display position of the Input Number Spinbox component depends on the panel on which it is layered. The following geometry behavior of the Input Number Spinbox is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Input Number Spinbox is positioned according to the `width` attribute.

- On horizontal panels: Input Number Spinbox is positioned according to the `width` attribute.

> **Note:** On BlackBerry platform, if the Input Number Spinbox component is set to a width that is too small to display all the numeric text, an `IndexOutOfBoundsException` or `IllegalArgumentException` may be thrown. To avoid this, take one of the following approaches:
>
> - do not set the `width` and `height` attributes, so that the component determines the appropriate size;
> - give the `width` and `height` attributes values that are large enough to accommodate the text.

For more information, see Section 6.4.2.1, "What You May Need to Know About Geometry Management and the Panel Group Layout Component."

6.5.4 How to Use the Select Boolean Checkbox Component

The Select Boolean Checkbox (`selectBooleanCheckbox`) component represents a check box that you use to enable single selection of `true` or `false` values, which allows toggling between selected and deselected states.

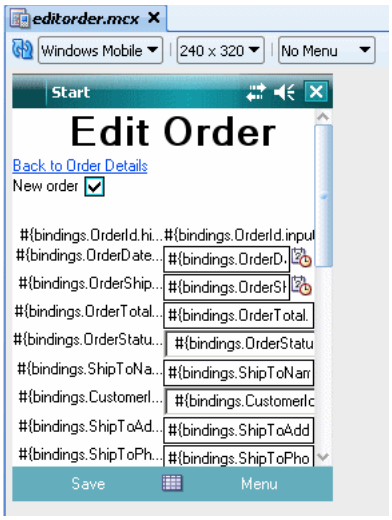
Example 6–11 demonstrates how to declare the `selectBooleanCheckbox` element in an MCX file.

Example 6–11 Creating Select Boolean Checkbox

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
      <amc:panelGroupLayout id="panelGroupLayout3"
                            layout="vertical"
                            rendered="#{!applicationScope.addMode}"
                            width="100%">
      ...
         <amc:selectBooleanCheckbox label="New order"
                                    id="selectBooleanCheckbox1"/>
      ...
```

Figure 6–25 shows the **New Order** Select Boolean Checkbox component in Design page of the visual editor.

Figure 6–25 Select Boolean Checkbox in Design Page



If you use the default font, this box will appear either empty or containing a check mark, depending upon the state of the field. Pressing the spacebar when the Select Boolean Checkbox component has the focus toggles its state.

6.5.4.1 What You May Need to Know About Geometry Management and the Select Boolean Checkbox Component

The display position of the Select Boolean Checkbox component depends on the panel on which it is layered. The following geometry behavior of the Select Boolean Checkbox is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Select Boolean Checkbox is always positioned below the previous component and horizontally aligned according to the `alignment` attribute value.
- On horizontal panels: Select Boolean Checkbox is always positioned to the right of the previous component and is vertically aligned according to the `verticalAlignment` attribute value.

For more information, see Section 6.4.2.1, "What You May Need to Know About Geometry Management and the Panel Group Layout Component.".

6.5.5 How to Use the Select One Choice Component

The Select One 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 component (see Section 6.5.5.2, "What You May Need to Know About Differences Between Select Items and Select Item Components") contained by the Select One Choice component.

Example 6–12 demonstrates how to declare the `selectOneChoice` element with the `selectItems` subelement in an MCX file.

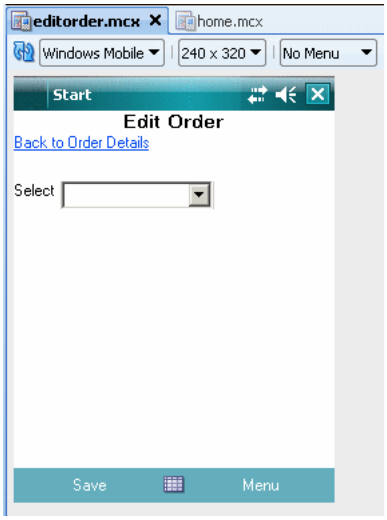
Example 6–12 Creating Select One Choice

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
```

```
<amc:panelGroupLayout id="panelGroupLayout3"
                      layout="vertical"
                      rendered="#{!applicationScope.addMode}"
                      width="100%">
...
   <amc:selectOneChoice label="Select">
      <amc:selectItems value="#{bindings.OptionBean.optionList}"/>
   </amc:selectOneChoice>
...
```

Figure 6–26 shows the **Select** Select One Choice with the Select Items subcomponent in Design page of the visual editor.

Figure 6–26 Select One Choice in Design Page



The combo box part of the Select One Choice component is positioned after the text label part.

6.5.5.1 What You May Need to Know About Geometry Management and the Select One Choice Component

The display position of the Select One Choice component depends on the panel on which it is layered. The following geometry behavior of the Select One Choice is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Select One Choice is always positioned below the previous component and horizontally aligned according to the `alignment` attribute value.
- On horizontal panels: Select One Choice is positioned according to the setting of the `width` attribute.

For more information, see Section 6.4.2.1, "What You May Need to Know About Geometry Management and the Panel Group Layout Component.".

6.5.5.2 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` tag 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 `selectItems` tag 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*.

6.5.6 What You May Need to Know About Event Listeners and Input Components

You can add the `valueChangeListener` event listener to input components.

This event listener is applicable to input components for ADF Mobile client run-time description on both BlackBerry smartphones and Windows Mobile devices, but it does not have any effect at design time.

For more information, see Section 6.12, "Using Event Listeners."

6.6 Creating and Using Output Components

ADF Mobile client provides the Output Text component for you to use as a label to display text.

Use the Convert Number (see Section 6.6.1.2, "Converting Numerical Values") and Convert Date Time (see Section 6.6.1.3, "Converting Date and Time Values") converters to facilitate the conversion of numerical and date-and-time-related data for the Output Text components.

You add an output component by dragging and dropping it onto the MCX page from the Component Palette (see Section 6.3.2.1, "Adding UI Components"). Then you use the Property Inspector to set the component's attributes (see Section 6.3.2.4, "Configuring UI Components"). For information on attributes of each particular component, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

When creating an Output Text component, always consider its geometry behavior (see Section 6.6.1.1, "What You May Need to Know About Geometry Management and the Output Text Component").

6.6.1 How to Use the Output Text Component

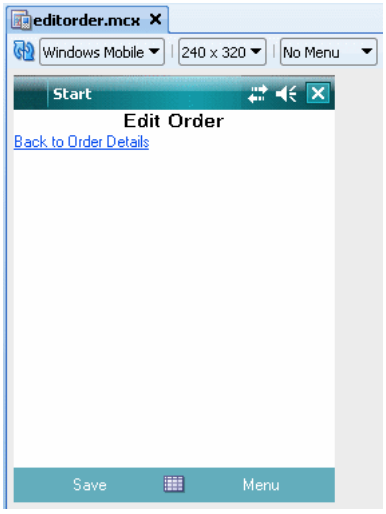
The Output Text component is represented by the `outputText` element in an MCX file. Example 6–13 demonstrates how to declare this element in an MCX file.

Example 6–13 Creating Output Text

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
      <amc:panelGroupLayout id="panelGroupLayout1"
                            layout="vertical"
                            rendered="#{!applicationScope.addMode}"
                            width="100%">
      ...
         <amc:outputText value="Edit Order"
                         id="outputText2"
                         align="center"
                         fontSize="14"
                         fontStyle="bold"
                         justification="center"/>
      ...
```

Figure 6–27 shows the **Edit Order** Output Text component in the Design page of the visual editor.

Figure 6–27 Output Text in Design Page



6.6.1.1 What You May Need to Know About Geometry Management and the Output Text Component

The display position of the Output Text component depends on the panel on which it is layered. The following geometry behavior of the Output Text is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Output Text always positions itself below the previous component, and is horizontally aligned according to the `alignment` attribute value.
- On horizontal panels: Output Text always positions itself to the right of the previous component, and is vertically aligned according to the `verticalAlignment` attribute value.

6.6.1.2 Converting Numerical Values

The Convert Number (`convertNumber`) is not an independent UI component: it is a converter that you use in conjunction with an Output Text or Input Text component or to display converted number or currency figures in a variety of formats following the specified pattern.

Example 6–14 demonstrates how to use the `convertNumber` element in the MCX file.

Example 6–14 Using Convert Number

```
<amc:panelLabelAndMessage label="ID:" truncateAt="7">
   <amc:outputText value="#{bindings.OrderId.inputValue}">
      <amc:convertNumber groupingUsed="false"
                         pattern="#{bindings.OrderId.format}"/>
   </amc:outputText>
</amc:panelLabelAndMessage>
```

To convert numerical values:

1. From the **Component Palette**, drag a **Convert Number** component and insert it within an **Output Text** or Input Text component, making it a child element of those components.
2. Open the **Property Inspector** for the Convert Number component and define its attributes. For more information, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

6.6.1.3 Converting Date and Time Values

The Convert Date Time (`convertDateTime`) is not an independent UI component: it is a converter that you use in conjunction with an Output Text or Input Date component to display converted date, time, or a combination of date and time in a variety of formats following the specified pattern.

Example 6–15 demonstrates how to use the `convertDateTime` element in an MCX file.

Example 6–15 Using Convert DateTime

```
<amc:panelLabelAndMessage label="ID:" truncateAt="7">
   <amc:outputText value="#{bindings.OrderId.inputValue}">
      <amc:convertDateTime type="date"
                           dateStyle="short"
                           pattern="#{bindings.OrderId.format}"/>
   </amc:outputText>
</amc:panelLabelAndMessage>
```

To convert date and time values:

1. From the **Component Palette**, drag a **Convert Date Time** component and insert it within an **Output Text** or Input Date component, making it a child element of that component.
2. Open the **Property Inspector** for the Convert Date Time component and define its attributes. For more information, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

6.7 Displaying Images

ADF Mobile client enables the display of images on BlackBerry smartphones and Windows Mobile devices using the Image component represented by a bitmap.

You add an Image (`image`) component by dragging and dropping it onto the MCX page from the Component Palette (see Section 6.3.2.1, "Adding UI Components"). Then you use the Property Inspector to set the component's attributes (see Section 6.3.2.4, "Configuring UI Components" and *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*).

6.7.1 How to Display an Image

You specify the image source in the MCX file (either manually, or using the Property Inspector), as well as other applicable attributes. Example 6–16 demonstrates how to declare this element in an MCX file.

Example 6–16 Creating Image

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
```

```
   <amc:form id="form0">
   ...
<amc:image source="/images/globalhelp.gif"
           shortDesc="global help"
           longDescURL="help.html"
           id="i2"/>
   ...
```

6.7.2 What You May Need to Know About Supported Image File Formats

The following are supported formats for BlackBerry smartphones:

- GIF
- JPEG
- PNG
- BMP
- TIFF
- WBMP

The following are supported formats for Windows Mobile devices:

- GIF
- JPG
- PNG

6.7.3 What You May Need to Know About Geometry Management and the Image Component

By default, the Image component uses only enough area to fit its contained bitmap.

If the `width` attribute is specified, the Image component will take on the lesser of the specified width and available width. If the `height` attribute is specified, the Image component will take on the lesser of the specified height and available height. If the `align` attribute is specified, the Image component will not necessarily use the full width of the parent container.

The display position of the Image component depends on the panel on which it is layered. The following geometry behavior of the Image is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Image is always positioned below the previous component and horizontally aligned according to the `align` attribute value.
- On horizontal panels: Image is always positioned to the right of the previous component and is vertically aligned according to the `verticalAlign` attribute value.

6.8 Creating and Using Tables

The Table (`table`) component displays data as rows and columns. This component allows a single row selection, sorting, and record navigation. The mobile client Table can include the following subcomponents in cells:

- Input Text (see Section 6.5.1, "How to Use the Input Text Component")
- Input Date (see Section 6.5.2, "How to Use the Input Date Component")

- Input Number Spinbox (see Section 6.5.3, "How to Use the Input Number Spinbox Component")
- Select Boolean Checkbox (see Section 6.5.4, "How to Use the Select Boolean Checkbox Component")
- Select One Choice[1] (see Section 6.5.5, "How to Use the Select One Choice Component")
- Output Text (see Section 6.6.1, "How to Use the Output Text Component")
- Button (see Section 6.9.1, "How to Use the Button Component")
- Link (see Section 6.9.2, "How to Use the Link Component")
- Image (see Section 6.7.1, "How to Display an Image")
- Spacer
- Panel Group Layout (see Section 6.4.2, "How to Use a Panel Group Layout Component")
- Panel Form Layout (see Section 6.4.3, "How to Use a Panel Form Layout Component")
- Panel Label and Message (see Section 6.4.4, "How to Use a Panel Label And Message Component")

Generally, when creating a Table, you add Column components, and then add other components to every Column component.

You can also add multiple components in a column by adding a Panel Group Layout or Panel Form Layout component as a child of the Column, and then in turn adding components to the Panel Group Layout (or Panel Form Layout).

You can add event listeners to Table components (see Section 6.8.3, "What You May Need to Know About Event Listeners and Table Components").

6.8.1 How to Use the Table Component

Example 6–17 demonstrates the declaration of a `table` element in an MCX file, with three `column` subelements, each containing `outputText` and `convertNumber` subelements.

Example 6–17 Creating Noneditable Table with Bindings

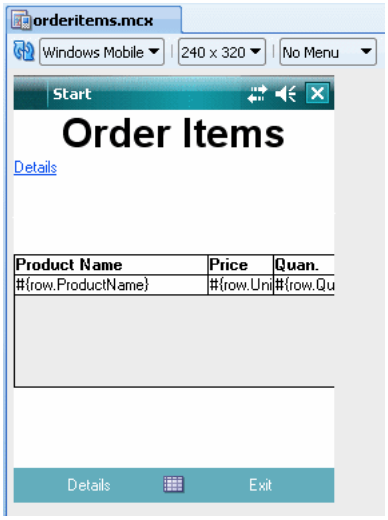
```
...
<amc:table id="tblOrders"
           value="#{bindings.OrderItems1View2.collectionModel}" var="row">
   <amc:column sortProperty="ProductName" sortable="true" headerText="Product Name">
      <amc:outputText value="#{row.bindings.ProductName.inputValue}"/>
   </amc:column>
   <amc:column sortProperty="UnitPrice" sortable="true" headerText="Price">
      <amc:outputText value="#{row.bindings.UnitPrice.inputValue}">
         <amc:convertNumber groupingUsed="false"
                            pattern="#{bindings.OrderItems1View2.hints.UnitPrice.format}"/>
      </amc:outputText>
   </amc:column>
   <amc:column sortProperty="Quantity" sortable="true" headerText="Quan.">
```

[1] To use a databound Select One Choice component within a Table, you must declare a list binding in the page definition file and associate it with the proper Column attribute. For more information, see Section 6.8.5, "What You May Need to Know About Using a Databound Select One Choice Component Within a Table."

```
      <amc:outputText value="#{row.bindings.Quantity.inputValue}">
         <amc:convertNumber groupingUsed="false"
                            pattern="#{bindings.OrderItems1View2.hints.Quantity.format}"/>
      </amc:outputText>
   </amc:column>
</amc:table>
...
```

Figure 6–28 shows the Table component in the Design page of the visual editor.

Figure 6–28 The Table component in Design Page



The Table component is equipped with the following additional features:

- Context-sensitive menu commands
- Dynamic sorting and filtering

To create a table using a data control, you bind the Table component to a collection. JDeveloper allows you to do this declaratively by dragging and dropping a view object from the Data Controls panel.

> **Tip:** You can also create a table by dragging a Table component from the Component Palette and completing the Create ADF Faces Table wizard.

To create a databound table:

1. From the **Data Controls** panel, select a view object.
2. Drag the view object onto an MCX page or into the **Structure** window.
3. From the context menu, choose the appropriate table.

   When you drag the collection, you can choose from the following types of tables:

   - **ADF Table**: Allows you to select the specific attributes you wish your editable table columns to display, and what UI components to use to display the data. By default, ADF Input Text components are used for most attributes, thus enabling the table to be editable. Attributes that are dates use the Input Date component. Additionally, if a control type control hint has been created for an

attribute, or if the attribute has been configured to be a list, then the component set by the hint is used instead.

- **ADF Read-Only Table:** Same as the **ADF Table**; however, each attribute is displayed in an Output Text component.

4. The ensuing **Edit Table Columns** dialog shows each attribute in the collection, and allows you to determine how these attributes will behave and appear as columns in your table.

   Note: If the collection contains a structured attribute (an attribute that is neither a Java primitive type nor a collection), the attributes of the structured attributes will also appear in the dialog.

   Using this dialog, you can do the following:

   - Allow the ADF Model layer to handle selection by selecting the **Row Selection** checkbox. Selecting this option means that the iterator binding will access the iterator to determine the selected row. Select this option unless you do not want the table to allow selection.
   - Allow the ADF Model layer to handle column sorting by selecting the **Sorting** checkbox. Selecting this option means that the iterator binding will access the iterator, which will perform an `order-by` query to determine the order. Select this option unless you do not want to allow column sorting.
   - Allow the columns in the table to be filtered using entered criteria by selecting the **Filtering** checkbox. Selecting this option allows the end user to enter criteria in text fields above each column. That criteria is then used to build a Query-by-Example (QBE) search on the collection, so that the table will display only the results returned by the query.
   - Group columns for selected attributes together under a parent column, by selecting the desired attributes (shown as rows in the dialog), and clicking the **Group** button.
   - Change the display label for a column. By default, the label is bound to the `labels` property for any control hint defined for the attribute on the table binding. This binding allows you to change the value of a label text once on the view object, and have the change appear the same on all pages that display the label.

     Instead of using this default, you can enter text or an EL expression to bind the label value to something else, for example, a key in a resource file.
   - Change the value binding for a column. You can change the column to be bound to a different attribute. If you simply want to rearrange the columns, you should use the order buttons. If you do change the attribute binding for a column, the label for the column also changes.
   - Change the UI component used to display an attribute. The UI components are set based on the table you selected when you dropped the collection onto the page, on the type of the corresponding attribute (for example, Input Date components are used for attributes that are dates), and on whether or not default components were set as control hints on the corresponding view object. You can change to another component using the context menu.

Tip: If one of the attributes for your table is also a primary key, you may want to choose a UI component that will not allow a user to change the value.

Tip: If you want to use a component that is not listed in the context menu, use this dialog to select the Output Text component, and then manually add the other tag to the page.

- Change the order of the columns using the order buttons.
- Add a column using the **Add** icon. There is no limit to the number of columns you can add. When you first click the icon, JDeveloper adds a new column line at the bottom of the dialog and populates it with the values from the first attribute in the bound collection; subsequent new columns are populated with values from the next attribute in the sequence, and so on.
- Delete a column using the **Delete** icon.

5. Once the Table is dropped on the page, you can use the **Property Inspector** to set other display properties of the Table component (see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*). For example, you may want to set the width of the table to a certain percentage or size. For more information about display properties, see "Using Tables and Trees" chapter in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

   Tip: When you set the table width to 100%, the table will not include borders, so the actual width of the table will be larger. To have the table set to 100% of the container width, expand the **Style** section of the **Property Inspector**, select the **Box** tab, and set the `borderWidth` attribute to 0 pixels.

   You enable sorting by setting a Column's `sortable` attribute to `true`, and its `sortProperty` attribute to a valid value. On BlackBerry smartphones, this allows the end user to invoke sorting from the menu when the table is in either cell or component selection mode: a table-specific "Sort" menu item is automatically added to the displayed menu at the time when the Table component gains focus (see Example 6–17). On Windows Mobile devices, sorting can be invoked by clicking on the column header. The header of sortable columns contains arrows indicating which column is currently sorted and in which direction.

   On BlackBerry smartphones, if you set a table column's `filterable` attribute to `true`, the table-specific "Filter" menu item is automatically added to the displayed menu at the time when the Table component gains focus (see Example 6–17).

6. If you want the end user to be able to edit information in the table and save any changes, you have to provide a way to submit and persist those changes. For more information and procedures on creating tables that allow data input, see "Using Tables and Trees" chapter in *Oracle Fusion Middleware Web User Interface Developer's Guide for Oracle Application Development Framework*.

6.8.2 What Happens When You Create a Table

Dropping a Table from the Data Controls panel has the same effect as dropping an Input Text or Form. Briefly, JDeveloper does the following:

- Creates the bindings for the table and adds the bindings to the page definition file

- Adds the necessary code for the UI components to the MCX page.

6.8.3 What You May Need to Know About Event Listeners and Table Components

You can add the following event listeners to Table components:

- `rangeChangeListener`
- `selectionListener`
- `sortListener`

These event listeners are applicable to the tables for ADF Mobile client run-time description on both BlackBerry and Windows Mobile devices, but they do not have any effect at design time.

For more information, see Section 6.12, "Using Event Listeners."

6.8.4 What You May Need to Know About the Table User Interaction Model

On non-touchscreen BlackBerry smartphones:

The Table supports three user interaction modes: table selection, row selection, and cell selection.

By default, the Table starts in table selection mode. You can change this behavior by specifying the `startingSelectionMode` attribute. Pressing the trackball changes the mode from table to cell to component, while pressing the back key changes the mode from component to cell to table.

In table selection mode, the entire table behaves as one focusable component. When the table has focus, its border is drawn with a different color (black by default). It is not possible to change the selected row or interact with components inside the table in this mode. That means that trackball and key events are consumed by the table and are not passed to the contained components.

In row selection mode it is possible to navigate between rows and columns. Navigating between rows updates the current row in the Table's `CollectionModel`, while changing the column updates the sort and filter menu choices. The current row is indicated by setting the background color to dark grey (`#9CACC9`), while the background of the currently selected cell is set to light yellow (`#FFFCD8`). As in table mode, it is not possible to interact with components inside the cell or to navigate between them.

In cell selection mode it is possible to interact with and navigate between components inside a cell. As in row selection mode, the current cell is indicated by drawing its border in a different color (blue by default). All key and trackball events are handled first by the focused component. If there are several focusable components within a cell, then navigation events first move focus between those components before changing the selected cell. If there are no focusable components inside a cell, then that cell may still become selected as in row selection mode.

On Windows Mobile devices:

The table navigation is performed by either using scroll bars or swiping in the desired direction. Touching a cell or giving focus to a component inside it will set this cell as the current cell, and its row as the current row.

6.8.5 What You May Need to Know About Using a Databound Select One Choice Component Within a Table

To use a databound Select One Choice component within a Table, you must declare a list binding in the page definition and then associate it with the proper Column attribute, as Example 6–18 shows.

Example 6–18 Using Databound Select One Choice Component in Tables

```
<bindings>
   ...
   <!-- added new list binding -->
   <list IterBinding="-iteratorID-" StaticList="false" Uses="-LOVID-"
         id="-listBindingID-" DTSupportsMRU="true"/>
   <tree ...>
      <nodeDefinition ...>
         <AttrNames>
            ...
            <!-- added Binds attribute that references list binding -->
            <Item Value="-lovEnabledAttributeName-" Binds="-listBindingID-"/>
```

For more information, see Section 6.3.2.3, "Adding Data Controls to the View."

6.9 Using Buttons and Links

You use the following components to enable actions, as well as navigation through the mobile client views:

- Button (see Section 6.9.1, "How to Use the Button Component")
- Link (see Section 6.9.2, "How to Use the Link Component")

You add a Button or Link component by dragging and dropping it onto the MCX page from the Component Palette (see Section 6.3.2.1, "Adding UI Components"). Then you use the Property Inspector to set the component's attributes (see Section 6.3.2.4, "Configuring UI Components"). For information on attributes of each particular component, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

You can add event listeners to ADF Mobile client action components (see Section 6.9.1.1, "What You May Need to Know About Event Listeners and Button Components" and Section 6.9.2.1, "What You May Need to Know About Event Listeners and Link Components").

When creating the components, always consider their geometry behavior (see Section 6.9.1.2, "What You May Need to Know About Geometry Management of Button Components" and Section 6.9.2.2, "What You May Need to Know About Geometry Management of Link Components").

The mobile client also supports navigation through the use of the back button (see Section 6.9.3, "How to Enable the Back Button Navigation").

6.9.1 How to Use the Button Component

You use the Button (`commandButton`) component to trigger actions and enable navigation through the view.

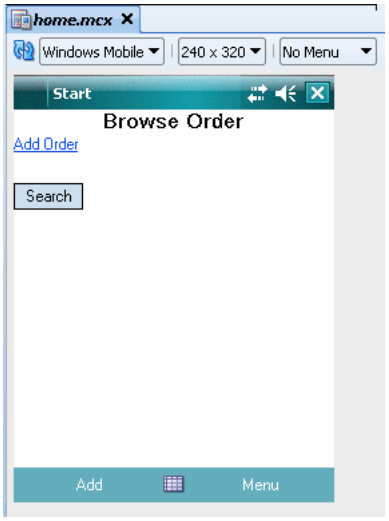
Example 6–19 demonstrates how to declare the `commandButton` element in an MCX file.

Example 6–19 Creating Button

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
      <amc:panelGroupLayout id="panelGroupLayout3"
                            layout="horizontal">
      ...
         <amc:commandButton text="Search"
                            id="commandButton2
                            actionListener="#{bindings.ExecuteWithParams}">
            <amc:setActionListener from="#{true}"
                                   to="#{applicationScope.SearchExecuted}"/>
         </amc:commandButton>
      ...
```

Figure 6–29 shows the **Search** Button in the Design page of the visual editor.

Figure 6–29 Button and Link in Design Page



You can use the Button component within the Panel Group Layout, Panel Form Layout, Panel Label And Message, and Table components, as well as inside any container component.

6.9.1.1 What You May Need to Know About Event Listeners and Button Components

You can add the `actionListener` to a Button.

This event listener is applicable to input components for ADF Mobile client run-time description on both BlackBerry smartphones and Windows Mobile devices, but it does not have any effect at design time.

For more information, see Section 6.12, "Using Event Listeners".

6.9.1.2 What You May Need to Know About Geometry Management of Button Components

The display position of the Button component depends on the panel on which it is layered. The following geometry behavior of the Button is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Button is always positioned below the previous component and horizontally aligned according to the `align` attribute value. If the contents of the component do not fit in a line, it does not wrap to the next lines.
- On horizontal panels: Button is always positioned to the right of the previous component and is vertically aligned according to the `verticalAlign` attribute value. If the contents of the component do not fit in a line, it does not wrap to the next lines.

6.9.2 How to Use the Link Component

You use the Link (`commandLink`) component to enable navigation.

Example 6–20 demonstrates how to declare the `commandLink` element in an MCX file.

Example 6–20 Creating Link

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
      <amc:panelGroupLayout id="panelGroupLayout7">
            <amc:commandLink id="commandButton1"
                             text="Add Order"
                             actionListener="#{bindings.CreateInsert}"
                             action="create"/>
   ...
```

Figure 6–29 shows the **Add Order** Link component in the Design page of the visual editor.

You can use the Link component within the Panel Group Layout, Panel Form Layout, Panel Label And Message, and Table components, as well as inside any container component.

6.9.2.1 What You May Need to Know About Event Listeners and Link Components

You can add the `actionListener` to a Link.

This event listener is applicable to input components for ADF Mobile client run-time description on both BlackBerry smartphones and Windows Mobile devices, but it does not have any effect at design time.

For more information, see Section 6.12, "Using Event Listeners."

6.9.2.2 What You May Need to Know About Geometry Management of Link Components

The display position of the Link component depends on the panel on which it is layered. The following geometry behavior of the Link is identical on BlackBerry and Windows Mobile platforms:

- On vertical panels: Link is always positioned below the previous component and horizontally aligned according to the `align` attribute value.
- On horizontal panels: Link is always positioned to the right of the previous component and is vertically aligned according to the `verticalAlign` attribute value.

6.9.3 How to Enable the Back Button Navigation

The mobile client supports navigation using the back button, with the default behavior of going back to the previously visited page enabled on the Escape-key press on the BlackBerry smartphone.

A combination of the page history implementation and the Form's `onBack` attribute (see Section 6.4.1.1, "How to Add a Form to a Page") provides the basis for this functionality.

You can customize the back button behavior by invoking a custom bean method. To do so, you add the `onBack` attribute to the `form` element (see Section 6.4.1.1, "How to Add a Form to a Page"). Example 6–21 shows an MCX file that uses the `onBack` attribute.

Example 6–21 Usage of onBack Attribute in MCX File

```
<amc:view xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0" onBack="#{applicationScope.Bean.onBackMethod}">
      <amc:panelGroupLayout id="panelGroupLayout1">
         <amc:outputText value="Press the back button" id="outputText1"/>
      </amc:panelGroupLayout>
   </amc:form>
</amc:view>
```

Example 6–22 shows a custom method that you may use to specify whether the framework should continue with the default behavior, or execute the custom behavior.

Example 6–22 Custom onBack Method

```
private void onBackMethod(Object[] params) {
   if (params != null && params.length > 0) {
      String param = params[0].toString();
      Trace.log(Trace.UI_LOGNAME, Level.FINE, this.getClass(),
                "onBackMethod", "params[0] = " + param);
   }
   int result = MessageBox.show("Title", "Select a value",
                new String[] { "Go back", "Cancel" }, new int[] { 20, 30 });
   if (params[0] instanceof ClientEvent) {
      ClientEvent backEvent = (ClientEvent) params[0];
      if (result == 20)
         backEvent.setParameter("isConsumed", new Boolean(false));
      else if (result == 30)
         backEvent.setParameter("isConsumed", new Boolean(true));
      else
         Trace.log(Trace.UI_LOGNAME, Level.FINE, this.getClass(),
                   "onBackMethod", "messageBox returned " + result);
   }
}
```

In the preceding example, the bean method is passed a `ClientEvent` object. The `isConsumed` parameter is set to a boolean value in the `setParameter` method of the `ClientEvent`. Setting the `isConsumed` parameter to `true` informs the framework that the click has been handled completely and that it should not continue with the default back button behavior.

Example 6–23 demonstrates how to invoke the custom `onBack` method shown in Example 6–22.

Example 6–23 Invoking Custom Method

```
public Object invokeMethod(String methodName, Object[] params) {
   if (methodName.equals("onBackMethod")) {
      onBackMethod(params);
   }
   return null;
}
```

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

6.10 Creating and Using Scanners

The Scanner (`scanner`) represents a non-visual UI component that allows you to define an action to react to a scan.

You add a Scanner component by dragging and dropping it onto the MCX page from the Component Palette (see Section 6.3.2.1, "Adding UI Components"). Then you use the Property Inspector to set the component's attributes (see Section 6.3.2.4, "Configuring UI Components" and *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*).

You can declare only one `scanner` element per MCX page. This component should be defined as a child of the Form component.

You can add event listeners to ADF Mobile client Scanner components (see Section 6.10.2, "What You May Need to Know About Event Listeners and Scanner Components").

6.10.1 How to Use the Scanner Component

Example 6–24 demonstrates how to declare the `scanner` element in an MCX file.

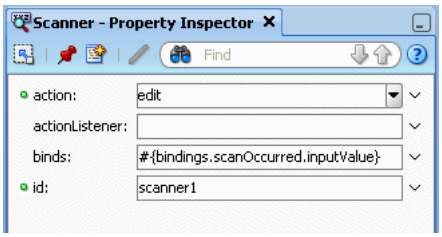
Example 6–24 Creating Scanner

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
   ...
       <amc:panelGroupLayout id="panelGroupLayout1"
                             layout="vertical"
                             rendered="#{!applicationScope.addMode}"
                             width="100%">
       ...
       </amc:panelGroupLayout>
       <amc:scanner id="scanner1" action="edit"/>
   </amc:form>
...
```

The Scanner component is not visible in the Design page.

Figure 6–27 shows the **Property Inspector** settings for the Scanner component.

Figure 6–30 Scanner Component Properties



6.10.2 What You May Need to Know About Event Listeners and Scanner Components

You can add the `actionListener` to Scanner components.

You can also define other `actionListener` components as children of the Scanner component.

This event listener is applicable to Scanner components for ADF Mobile client run-time description on both BlackBerry smartphones and Windows Mobile devices, but it does not have any effect at design time.

For more information, see Section 6.12, "Using Event Listeners."

6.10.3 How to Integrate a Barcode Scanner Into a Mobile Client Application

You can integrate a barcode scanner into an ADF Mobile client application by creating the scanner data control with the Scanner Data Control for ADF Mobile Client Application wizard.

6.10.3.1 Creating a Barcode Scanner Data Control

Using the two-step wizard, you configure the scan objects used in applications and their scan fields. You can also configure prefixes and suffixes that indicate where the scan starts and finishes a barcode.

Before you begin:

1. Know the prefix and suffix for the scan. This information delimits the beginning and the end of the data that is sent by the scanning device. The characters used to mark the prefix and suffix are dependent on the specific device, but can usually be programmed into the device. While some devices allow you to program any combination, others limit this. If you determine the prefix and suffix, then use a short character string that is a non-alphabetic character sequence, one that is not easily entered and that you would not normally see.
2. Know the format of the actual scanned data. This information is comprised of the scan objects and scan fields as well as the field delimiters (that is, what the scan fields are and if they are delimited by prefix and suffix character strings or by length). This information is usually pre-determined because the barcode format is probably already created and being used.

To create a scanner data control:

1. Click **File > New > Business Tier > Scanner Data Control for ADF Mobile Client Application**.
2. Enter a name for the scanner data control or accept the default name, ScannerDC, as shown in Figure 6–31.

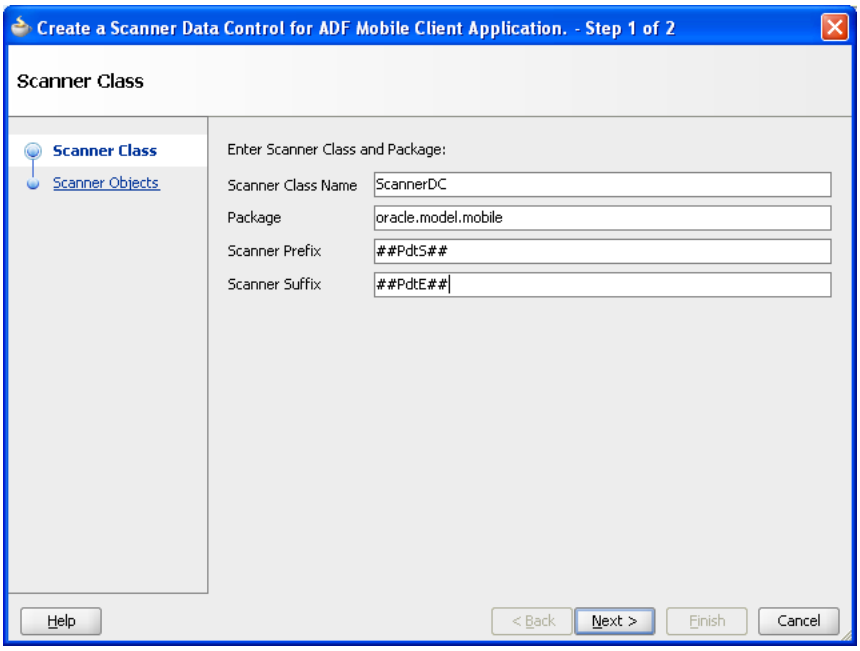
3. Use the **Browse** feature to select package name or accept the default value, which is the current package or enter the package name.
4. Enter the prefix that delimits the start of the scan from the physical scanner device.
5. Enter the delimiter that marks the end of the scanned string.

> **Note:** Do not use words as either the suffix or the prefix. Instead, incorporate hidden characters (such as ç, ÿ, or ~).

6. Click **Next**.

> **Note:** You must define the scanner name, package, prefix and suffix to proceed to the next page of the wizard.

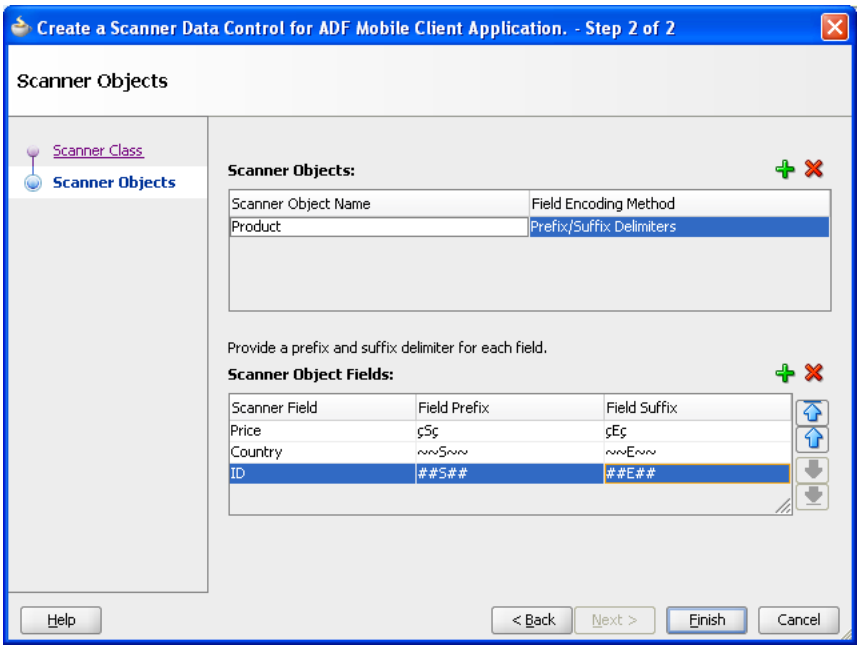
Figure 6–31 Defining the Scanner Class



7. Click **Add** to create a scanner object.
8. Accept the unique name added by JDeveloper, or enter a unique name for the scanner object.
9. Select one of the following methods to parse the scanned barcode:
   - **Prefix/Suffix Delimiters**: Enter the prefix and suffix that denote the start and end of a field. This is an optional parameter; you can define a prefix or suffix only, or opt to define neither the prefix nor suffix. The **Up** and **Down** arrows enable you to reorder the fields.

     Figure 6–32 shows a scanned object called Product with fields named `Price`, `Country`, and `ID`. Because the Product object uses the Prefix/Suffix Delimiters method to parse its fields, each field is differentiated by a character string.

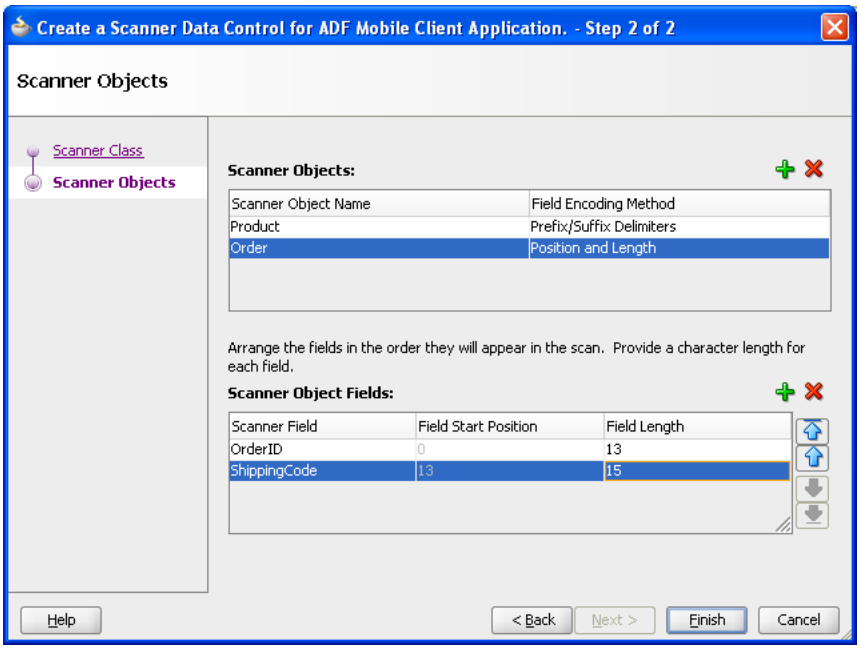
Figure 6–32 Creating Scanner Objects



- **Position and Length**: The **Field Start Position** is a read-only value that indicates where a field starts. For the **Field Length** parameter, enter a number that indicates where the field ends. JDeveloper automatically calculates the start position when you reorder fields. The default length is 1 (the minimum). This is an optional parameter.

  As shown in Figure 6–33, JDeveloper marks the start of the `Shipping Code` field start position at 13, which corresponds to the length of the `OrderID` field. JDeveloper also recalculates the field positions when you use the **Up** and **Down** arrows to reorder the fields. For example, if you moved `OrderID` below `ShippingCode`, then JDeveloper recalculates the start position for `OrderID` at Position 15, the length of the `ShippingCode` field.

Figure 6–33 Setting the Position and Length of the Scanner Object Fields



Note: You must create at least one scanner field to complete the wizard and create a scanner data control.

10. Click **Add** to create a scanner field.
11. Accept the unique name added by JDeveloper, or enter a unique name for the scanner field.
12. Click **Finish**.

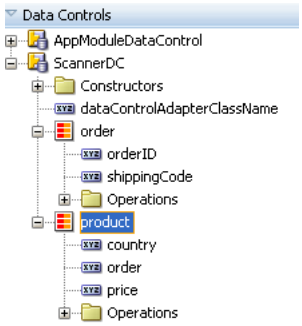
Note: Repeat these steps to add additional scan objects and scan object fields.

6.10.3.2 What Happens When You Create a Scanner Data Control

After you complete the wizard, JDeveloper creates a data control using the name defined in the wizard. This file includes data controls for the scanner objects. Because JDeveloper also updates `DataControls.dcx` file with the scanner data control information, the scanner data controls display in the Data Controls panel. (The design time uses this file to populate the Data Control panel.) Figure 6–34 shows a scanner file called `ScannerDC` (the default name generated by JDeveloper) as well as data collections created from the configured scanner objects, `order` and `project`.

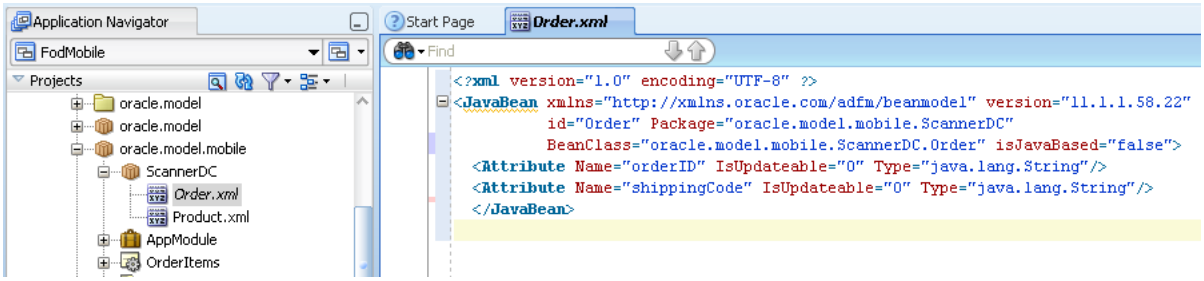
Note: You can add only one scanner data control file per application.

Figure 6–34 Scanner Data Controls



In addition to the data controls, the wizard creates XML files used by the runtime to parse the scanned data. For example, Figure 6–35 shows the resulting XML file created for a scanned object called order in the Application Navigator.

Figure 6–35 The XML File for a Scanned Object



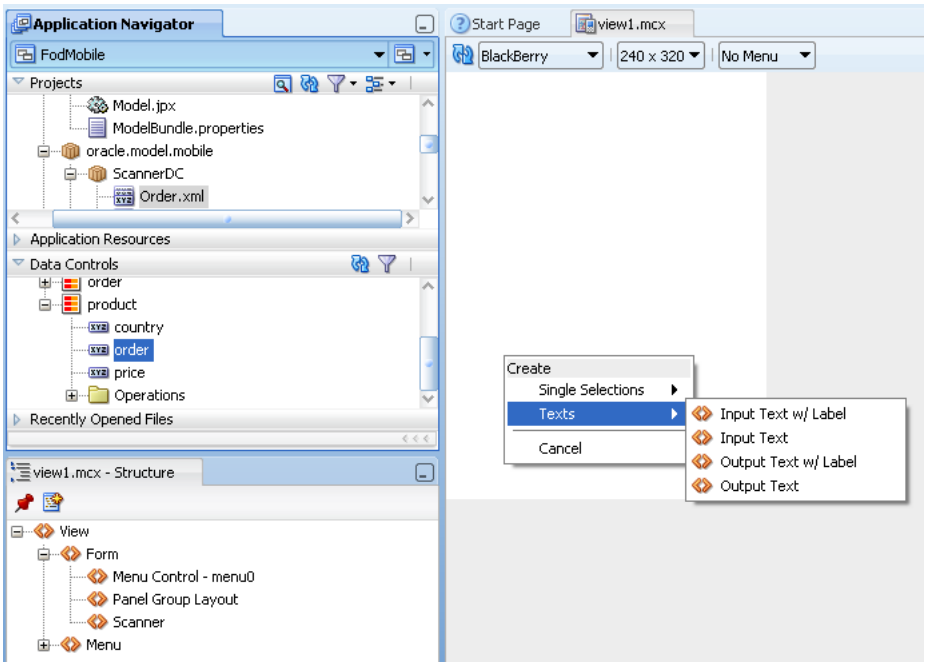
6.10.3.3 Enabling Scanning in Mobile Client Applications

The scanner component, which you nest within a `form` element on an MCX page, enables scanning in the application. This component, which is not visible on the MCX page, essentially ties the page's actions and fields to the scanner data control; when you add the scanner component tag to an MCX page, it enables any of the necessary page-centric listeners and handlers to catch and activate barcode scans. It also contains references to fields and actions that may be triggered by the scanner data control when the page is active. When the scanner component receives data from the physical scanner, it populates the fields on the page with readable data converted from the scanned data. You create these input fields by dragging the scanner data controls into the MCX page.

To enable scanning:

1. Drag and drop the scanner component from the component palette into the `form` element of the MCX page, and then define its properties, as described in Section 6.10.1, "How to Use the Scanner Component."
2. Drag a field of the scanner data control object collection into the `panelGroupLayout` element of the MCX page.
3. Select **Texts**, as Figure 6–36 shows, and then choose from among the input or output text options, as described in "Using Attributes to Create Text Fields" of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Figure 6–36 Creating Input and Output Text Fields from a Scanner Data Control Attribute



4. Click **OK**.

6.11 Creating and Using Menus

The ADF Mobile client Menu Designer enables you to create platform-specific menus for both BlackBerry smartphones and Windows Mobile devices.

> **Note:** Even though the Menu Designer represents the basic structure of menus, such as cascading menus, menu item ordering and menu item presence based on the single selection of components or panels, menus that you design for one platform cannot transform into menus for another platform.

Typically, to create a menu, you make a selection in the Structure window or source editor for your MCX file. The visual editor displays a menu or its children, if any of those are selected. If a component or a panel is selected, then the menu associated with that component or panel is displayed. If multiple components are selected, the menu is not displayed. As menus are platform-specific, BlackBerry-style menus are shown only on BlackBerry smartphones, and Windows Mobile-style menus are displayed only on Windows Mobile devices.

ADF Mobile client's Menu Designer lets you define different types of menus for your application.

6.11.1 Menu Types

The mobile client's Menu Designer supports the following styles of menus:

- Standard menu for Windows Mobile devices
- Full menu for BlackBerry smartphones

You choose whether or not to display a menu by making selections using a combo box located on the UI designer tool bar. The following are valid values for the menu type selector:

- No Menu: The menu is not displayed when a UI component or panel is selected. This is the default option.
- Main: Only full menus are displayed when a UI component or panel is selected. Corresponds to either Windows Mobile Standard menu, or BlackBerry Full menu.

> **Note:** If the menu itself or any of its children is selected, the menu will still be displayed on the appropriate platform regardless of the selection in the menu type selector.

Also note the following:

- If you change the platform at design time, the menu type selector will retain the displayed value.
- If you open a new split screen, the menu type selector will switch to the default value of *No Menu*.
- If a menu or any of its menu items is selected in the Structure window or source editor, this menu will be displayed in the Design page regardless of the menu type selector's value on an applicable platform.

Table 6–7 lists types of menus that you can define. Each type behaves differently depending on the platform on which your application is viewed.

Table 6–7 Menu Types

Type	Behavior on BlackBerry Smartphones	Behavior on Windows Mobile Devices
Main	This menu definition describes the Full menu for the form.	This menu definition describes the right Standard menu button for the form.
Alt	This menu definition will be combined into the Full menu for the form.	This menu definition describes the left Standard menu button for the form.

When displayed, menus with the same `type` and `id` values and that meet the `platform` type criteria are combined together and treated as a single menu. For example, the contents of the following menu definitions in an MCX file would be combined on a Windows Mobile device:

```
<menu type="main" platform="all">
```

and

```
<menu type="main" platform="wm">
```

Note that for Main and Alt menus, the `id` attribute values are implicitly matching.

6.11.1.1 Main

Example 6–25 shows a Main menu defined in an MCX file. Note that the `platform` attribute is specified for one menu item only ("Exit"), which will be displayed on Windows Mobile devices, but not on BlackBerry smartphones. The rest of the menu items in Example 6–25 are applied the default `platform` setting of `all`.

Example 6–25 Defining Main Menu

```
<amc:menu id="main" type="main">
   <amc:menuGroup index="200">
      <amc:commandMenuItem label="View Orders"
                           index="10"
                           weight="0"/>
   </amc:menuGroup>
   <amc:menuGroup index="250">
      <amc:commandMenuItem label="Edit Order"
                           index="20"
                           weight="0"/>
      <amc:commandMenuItem label="Call Customer"
                           index="30"
                           weight="0"/>
      <amc:commandMenuItem label="Exit"
                           index="70"
                           weight="0"
                           platform="wm"
                           action="appExit"/>
   </amc:menuGroup>
</amc:menu>
```

As a general rule, when specifying the Main menu type, you should limit it to only one menu definition per platform.

As Main menus are static to a page, you cannot associate them individually with a UI component. If individual menu items should change on the Main menu, consider using the EL expression of the Menu Item's `visible` attribute (see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*).

Note: Main menus are always of fixed width.

6.11.1.2 Alt

Example 6–26 shows an Alt menu defined in an MCX file.

Example 6–26 Defining Alt Menu

```
<amc:menu id="altMain" type="alt">
   <amc:menuGroup index="200">
      <amc:commandMenuItem label="View Items"
                           index="0"
                           weight="0"/>
   </amc:menuGroup>
</amc:menu>
```

When specifying the Alt menu type, limit it to only one menu definition per platform type.

6.11.2 Menu Components

A Menu component represents a virtual XSD group definition referring to either a `menu` element or a `commandMenuItem` element in an MCX file (see Example 6–26 and Example 6–27), allowing you to create a menu that contains both menu items and submenus.

You add a menu component by dragging and dropping it onto the MCX page from the Component Palette (see Section 6.3.2.1, "Adding UI Components"). Then you use the

Property Inspector to set the component's attributes (see Section 6.3.2.4, "Configuring UI Components"). For information on attributes of each particular menu component, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

Example 6–27 shows how to define menu components in an MCX file.

Example 6–27 Creating Menus Using Various Menu Components

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
       <amc:panelGroupLayout backgroundColor="red" width="100"/>
   </form>
   ...
   <amc:menu id="altMain" type="alt">
      <amc:menuGroup index="200">
         <amc:commandMenuItem label="View Items"
                              index="0"
                              weight="0"/>
      </amc:menuGroup>
   </amc:menu>
</view>
```

6.11.2.1 Menu

ADF Mobile client menu defines a new menu construct, which may be represented by a group of Menus and Menu Items, although you typically use it to define a group of Menu Items.

To create a menu, you define the top-level `menu` (element in an MCX file) directly within the view, as Example 6–27 shows.

The `label` attribute (see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*) lets you assign a recognizable name for those platforms that support submenus.

6.11.2.2 Menu Item

ADF Mobile client's Menu Item (`commandMenuItem`) is the base element of menus. It defines a single actionable item on a menu that the end user can execute. Most of its attributes are general-purpose, with appropriate behavior occurring on all platforms. However, a select group of attributes is platform-specific and requires definition of XSD attribute groups (see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*).

6.11.2.3 Menu Group

You use the Menu Group to group visual elements in a menu. Each Menu element must have at least one child Menu Group element. If you place multiple Menu Group elements under a Menu, then a Separator will be displayed.

You define ADF Mobile client's `menuGroup` (element in an MCX file) within the `menu` element, as Example 6–27 shows.

6.11.2.4 Menu Control

ADF Mobile client's Menu Control defines when menus are displayed.

You use it to create associations between a menu displayed on the menu bar and a UI component. Each such menu must be attached to at least one component in order to be displayed. In an MCX file, you define `menuControl` elements as children of UI components. Depending on the target platform, a menu is attached by default to the

top-level `form` element (see Example 6–28) so that the menu is always present. You can optionally define special menus that appear only for certain components by defining `menuControl` elements as children of those components. For example, you could specify a telephone number input component with a menu item added to "Call", but this menu should only appear when the focus is on the telephone number field.

The Menu Control can contain Select Items components (see Section 6.5.5.2, "What You May Need to Know About Differences Between Select Items and Select Item Components").

ADF Mobile client's Menu Control may only reference definitions of menus displayed on the menu bar when their associated UI component has focus; Main and Alt type menus are statically defined for the form, and all components are implicitly associated with them.

WARNING: If you define a menu without specifying a Menu Control element, this menu will never be displayed.

The Menu Control is represented by the `menuControl` element in an MCX file. Example 6–28 demonstrates how to define this element in an MCX file.

Example 6–28 Defining the Menu Control

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
      <amc:menuControl refId="altMain"/>
      <amc:panelGroupLayout id="home" layout="vertical">
         <amc:panelGroupLayout layout="vertical">
            <amc:inputText id="txtUserName"
                           label="User Name: "
                           value="#{securityContext.userName}"/>
            <amc:inputText id="txtPassword"
                           label="Password: "
                           value="#{securityContext.password}" secret="true"/>
      </amc:panelGroupLayout>
      ...
   </amc:form>
</amc:view>
```

For more information, see Section 6.11.3, "How to Associate Menus with UI Components."

6.11.2.5 Sub Menu

You use ADF Mobile client's Sub Menu (`subMenu`) to create cascading menus for Windows Mobile devices.

Note: Submenus always expand to the left.

In an MCX file, you define `subMenu` elements as children of the `menu`, as Example 6–29 shows.

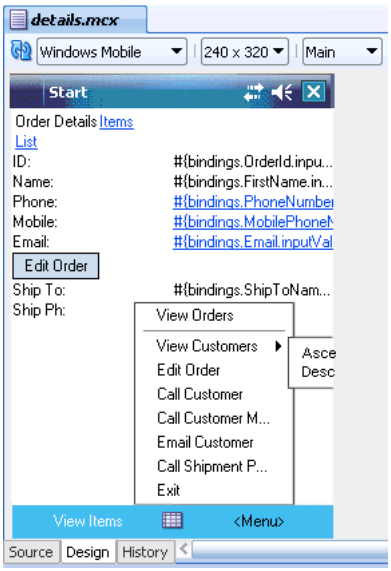
Example 6–29 Defining the subMenu

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   ...
   <amc:menu id="main" type="main">
```

```
      <amc:menuGroup index="200">
         <amc:commandMenuItem label="View Orders"
                              index="10"
                              weight="0"/>
      </amc:menuGroup>
      <amc:menuGroup index="250">
         <amc:subMenu label="View Customers"
                 rendered="true">
            <amc:menuGroup index="200">
               <amc:commandMenuItem label="Ascending"
                                    index="15"
                                    weight="0"/>
               <amc:commandMenuItem label="Descending"
                                    index="16"
                                    weight="0"/>
            </amc:menuGroup>
         </amc:subMenu>
         <amc:commandMenuItem label="Edit Order"
                              index="20"
                              weight="0"/>
         <amc:commandMenuItem label="Call Customer"
                              index="30"
                              weight="0"/>
         <amc:commandMenuItem label="Call Customer Mobile"
                              index="40"
                              weight="0"/>
         <amc:commandMenuItem label="Email Customer"
                              index="50"
                              weight="0"/>
         <amc:commandMenuItem label="Call Shipment Phone"
                              index="60"
                              weight="0"/>
         <amc:commandMenuItem label="Exit"
                              index="70"
                              weight="0"
                              platform="wm"
                              action="appExit"/>
      </amc:menuGroup>
   </amc:menu>
</amc:view>
```

Figure 6–37 shows the Sub Menu component in the Design page of the visual editor.

Figure 6–37 Defining the Sub Menu



6.11.3 How to Associate Menus with UI Components

You can create an association between a UI component and one or more Alt or Main menus by inserting a `menuControl` child element under the component element in your MCX file, and then setting the `refId` attribute of the `menuControl` (see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*) to the `id` attribute of the subject menu item, as the following example shows:

```
...
<amc:outputText id="AccountName" …>
   <amc:menuControl refId="accountContextMenu">
</amc:outputText>
...
```

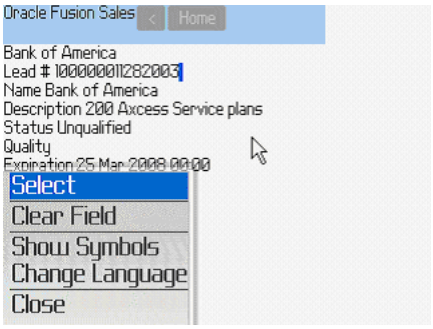
For menus that are meant to be displayed on the top-level menu bar, the mobile client Menu Designer provides an association tool in the form of a popup dialog that you can reach through a context menu selection on the UI component in the Structure window. This dialog displays a tree view of the available menus defined on the page and allows for a multiple selection of them. Upon completion of the dialog, you create a `menuControl` element under the original component element for each selected menu in the dialog. That is, when an association is created between a menu and a UI component, the menu will be displayed on the menu bar when its associated UI component has focus.

6.11.4 How to Create Menus for BlackBerry Smartphones

You can create full menus for BlackBerry smartphones.

The full menu pops up on the side of the screen upon the press of the BlackBerry Menu button, as Figure 6–38 shows.

Figure 6–38 Full Menu Displayed on a BlackBerry Smartphone



Note that BlackBerry smartphones do not natively support submenus.

The mobile client's Menu Designer positions full menus in the center of the design view, and the menus readjust their position according to the selected form factor.

Example 6–30 shows the source view of the menu defined in an MCX file.

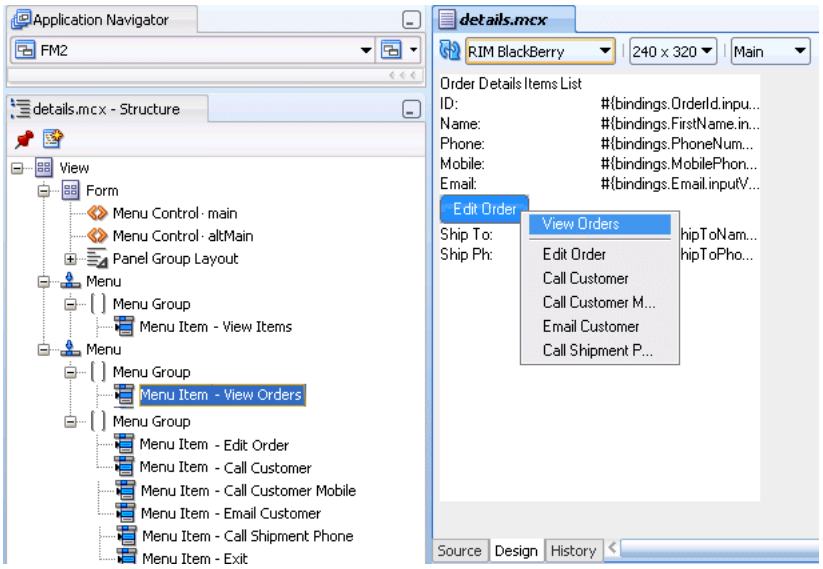
Example 6–30 BlackBerry Menu in Source View

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   ...
   <amc:menu id="altMain" type="alt">
      <amc:menuGroup index="200">
         <amc:commandMenuItem label="View Items"
                              index="0"
                              weight="0"/>
      </amc:menuGroup>
   </amc:menu>
   <amc:menu id="main" type="main">
      <amc:menuGroup index="200">
         <amc:commandMenuItem label="View Orders"
                              index="10"
                              weight="0"/>
      </amc:menuGroup>
      <amc:menuGroup index="250">
         <amc:commandMenuItem label="Edit Order"
                              index="20"
                              weight="0"/>
         <amc:commandMenuItem label="Call Customer"
                              index="30"
                              weight="0"/>
         <amc:commandMenuItem label="Call Customer Mobile"
                              index="40"
                              weight="0"/>
         <amc:commandMenuItem label="Email Customer"
                              index="50"
                              weight="0"/>
         <amc:commandMenuItem label="Call Shipment Phone"
                              index="60"
                              weight="0"/>
         <amc:commandMenuItem label="Exit"
                              index="70"
                              weight="0"
                              platform="wm"
                              action="appExit"/>
      </amc:menuGroup>
   </amc:menu>
```

```
</amc:view>
```

Figure 6–39 shows the menu from Example 6–30 in the Structure window and the Design page.

Figure 6–39 BlackBerry Menu in Structure Window and Design Page



6.11.4.1 Defining a BlackBerry Full Menu

When creating full menus for BlackBerry smartphones, consider the following;

- You can specify only one BlackBerry Main menu per MCX file. If you declare more than one, the Menu Designer will only use the first one while ignoring the rest.
- You cannot create submenus for BlackBerry Main menus.
- A number of UI components listed in the following table, when selected, automatically add their own menu items to the BlackBerry Main menu.

UI Component	Menu Item	index[1]
Input Text	Clear Field	60
	Select	90
	Cancel Selection	50
	Copy	40
	Cut	70
	Paste	80
Input Date	Change Option	30270
	Copy	64
Output Text	Copy	64
Select One Choice	Change Option	40
Select Boolean Checkbox	Change Option	30270

[1] The index attribute of the Menu Item together with the group is used to determine how the menu items are grouped and where separators, if any, should be inserted. The valid values for the index are between 0 and 65535. The default value is 0.

- If you select a BlackBerry Main menu in the Structure window or source editor, every menu item is shown. This is the only way to see the superset of all menu items.
- Since the BlackBerry Main menu is applicable to any component in an MCX file, it is displayed for any selected component, assuming that the menu type selector is set to Main.
- The contents of the BlackBerry Alt menu are appended to the BlackBerry Main menu whenever displayed.

6.11.5 How to Create Menus for Windows Mobile Devices

You can create a standard menu for Windows Mobile devices.

The standard menu is the two-button style of menu, with menu items popping up from either of the two buttons on the bottom of the screen. Note that, instead of a menu, you can assign each button an action. You can also create submenus for menu items on the popup menu, which would create the cascading effect.

The Main menu, which refers to the right-side menu, consists of the following two parts:

1. The appropriate half of the menu bar displayed directly under the visual editor.
2. The associated popup menu displayed directly above the menu bar.

The Alt menu refers to the left-side menu and consists of the appropriate half of the menu bar displayed directly under the visual editor.

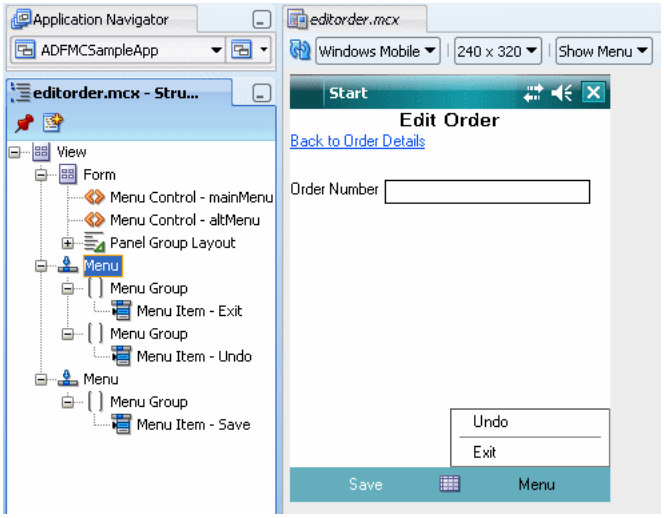
Example 6–31 shows the source view of the menu defined in an MCX file.

Example 6–31 Windows Mobile Menu in Source View

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   ...
   <amc:menu type="main" id="mainMenu">
      <amc:menuGroup id="menuGroup1" index="1">
         <amc:commandMenuItem id="commandMenuItem1"
                              label="Exit"
                              action="exit"/>
      </amc:menuGroup>
      <amc:menuGroup id="menuGroup2">
         <amc:commandMenuItem label="Undo"
                              id="commandMenuItem2"
                              action="goback"
                              actionListener="#{bindings.Rollback}"/>
      </amc:menuGroup>
   </amc:menu>
   <amc:menu id="altMenu" type="alt">
      <amc:menuGroup id="menuGroup3">
         <amc:commandMenuItem label="Save"
                              id="commandMenuItem3"
                              action="goback"
                              actionListener="#{bindings.Commit}"/>
      </amc:menuGroup>
   </amc:menu>
</amc:view>
```

Figure 6–40 shows the menu from Example 6–31 in the Structure window and the Design page.

Figure 6–40 Windows Mobile Menu in Structure Window and Design Page



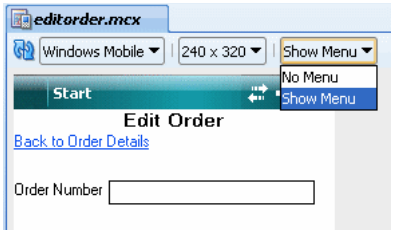
When creating Main and Alt menus for Windows Mobile devices, consider the following:

- The menu bar (without left or right menu buttons) is always visible by default on the Windows Mobile platform, even if there is no definition of the standard menu in the MCX file.
- The Windows Mobile editor height is reduced by the menu bar height.
- You can specify only one Main type menu per MCX file, although you may specify one Main type menu for each platform option.
- An Alt menu for the Windows Mobile platform has a single menu item.
- Both Main and Alt menus are optional: you do not have to define them.
- The right menu button is labeled with the `label` attribute of the Main menu (see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*).
- If you select the Main menu or any of its children in the Structure window or source editor, the right menu button popup will be displayed showing all submenus and menu items.
- If you select a UI component in the Structure window while the menu type selector is set to Main, the left menu button label will be updated (if this component has a menu item associated with it) and the right menu button popup will be shown (with the system menu items and all menu items associated with the selected component).
- You create separator lines similarly to how you create them for BlackBerry menus, using the Group (`group`) component and the `index` attribute (see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*).

6.11.6 What You May Need to Know About Design-Time Menu Usage

Figure 6–41 shows a menu combo box that appears at the top of the preview window at design time, with the values "Main" and "No Menu" (default).

Figure 6–41 Menu Selector at Design Time



If the "No Menu" option is selected and the focus is placed on a Menu Control, the menu will pop up and appear in the preview window displaying all of its submenus. If the focus is moved to another component, the menu will disappear. On Windows Mobile devices, if the Menu Control is either associated with the currently focused component or associated with the Form, the menu label will appear in the menu bar at the bottom. This is an indication and visual cue that the menu has been correctly attached to another component via the Menu Control. On BlackBerry smartphones, there is no visual cue for menus since they only appear when the menu key is pressed.

If the "Main" option is selected, all menus associated with the current component or any parent components, including the Form, are always displayed. This allows to see which menus are associated with which components while scrolling through a page or clicking on a page. Because these menus pop up over other components in the preview, this option is used only to verify menu associations and is usually turned off by default.

6.11.7 What You May Need to Know About Event Listeners and Menus

You can add the `actionListener` to menus.

This event listener is applicable to menus for ADF Mobile client run-time description on both BlackBerry smartphones and Windows Mobile devices, but it does not have any effect at design time.

For more information, see Section 6.12, "Using Event Listeners."

6.12 Using Event Listeners

You may use the following listeners to add awareness of the view-triggered events to your mobile application:

- `valueChangeListener`: This is a method reference to a value change listener. The `ValueChangeEvent` is fired when a value is updated in an input component.
- `selectionListener`: This is a method reference to a selection listener. The `SelectionEvent` is fired when a selection changes in a Table component.
- `actionListener`: This is a method reference to an action listener. The `ActionEvent` is fired when a command component or a menu element is activated.
- `sortListener`: This is a method reference to a sort listener. The `SortEvent` is fired when a table column sort criteria are changed.
- `rangeChangeListener`: This is a method reference to a range change listener. The `RangeChangeEvent` is fired when a navigation happens on a table component.

You define a listener in one of the following ways:

- Manually in the source of an MCX file.
- From the **Property Inspector** of the selected component. For more information, see *Oracle Fusion Middleware Tag Reference Library for Oracle ADF Mobile Client*.

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 6.14.4, "EL Events."

When defining event listeners in your Java code, you need to pass the `oracle.adfnmc.event.ClientEvent` class that contains the following:

- a source value that returns an `Object`;
- a type value that is a `String` representation of the event's name;
- a property map to retrieve relevant information.

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

Example 6–32, Example 6–33, and Example 6–34 demonstrate a Button and a Link components calling the same bean method. The source value of the `ClientEvent` determines which object invoked the event by showing a message box with the component's ID.

Example 6–32 Calling a Bean Method from MCX File

```
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
   <amc:form id="form0">
      <amc:panelGroupLayout id="panelGroupLayout1">
         <amc:commandButton text="commandButton1" id="commandButton1
              actionListener="#{applicationScope.Bean.actionListenerMethod}">
         </amc:commandButton>
         <amc:commandLink text="commandLink1" id="commandLink1
              actionListener="#{applicationScope.Bean.actionListenerMethod}">
         </amc:commandLink>
      ...
```

Example 6–33 Using the ClientEvent

```
private void actionListenerMethod(ClientEvent clientEvent) {
   Component source = (Component) clientEvent.getSource();
   MessageBox.show("actionListener called for " + source.getId());
}
```

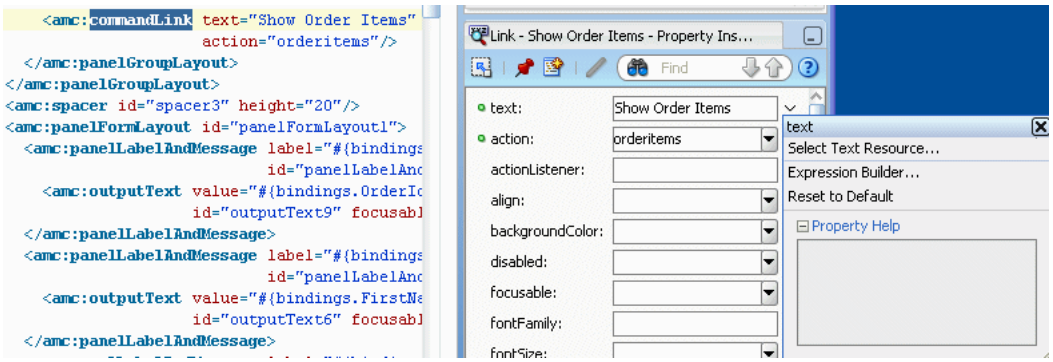
Example 6–34 Invoking the Event Method

```
public Object invokeMethod(String methodName, Object[] params) {
   if (methodName.equals("actionListenerMethod")) {
      actionListenerMethod((ClientEvent) params[0]);
   }
   return null;
}
```

6.13 Localizing UI Components

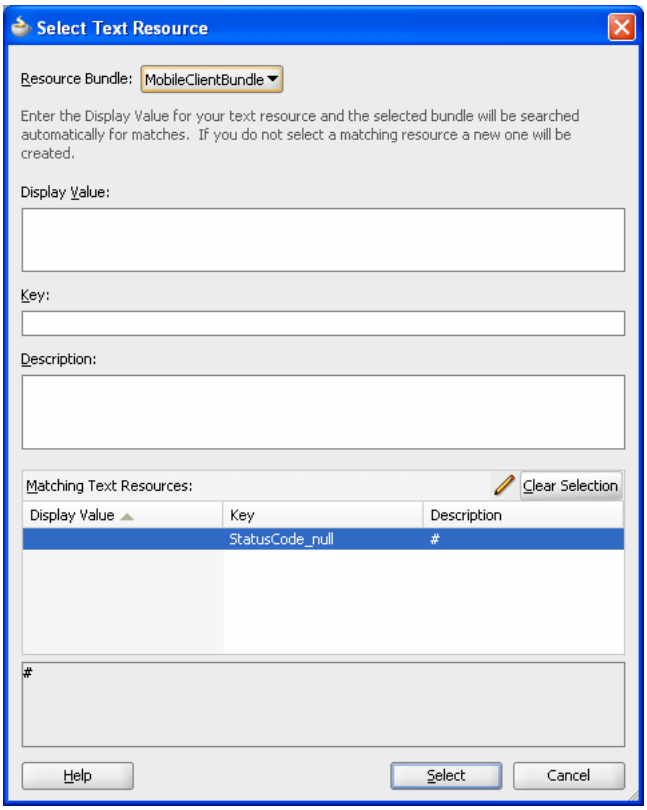
In your ADF mobile client application, you can localize the text that UI components display. You do so by selecting a component and one of its text-presenting properties whose value you intend to localize, and then choosing **Select Text Resource** in the **Property Inspector** (see Figure 6–42).

Figure 6–42 Selecting Text Resource



This will display the standard ADF **Select Text Resource** dialog that Figure 6–43 shows. You use this dialog to enter or find a string reference for the property you are modifying.

Figure 6–43 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 6–44 and Figure 6–45 show respectively the completed Select Text Resource dialog and changes in the MCX file.

Figure 6–44 Completed Select Text Resource Dialog

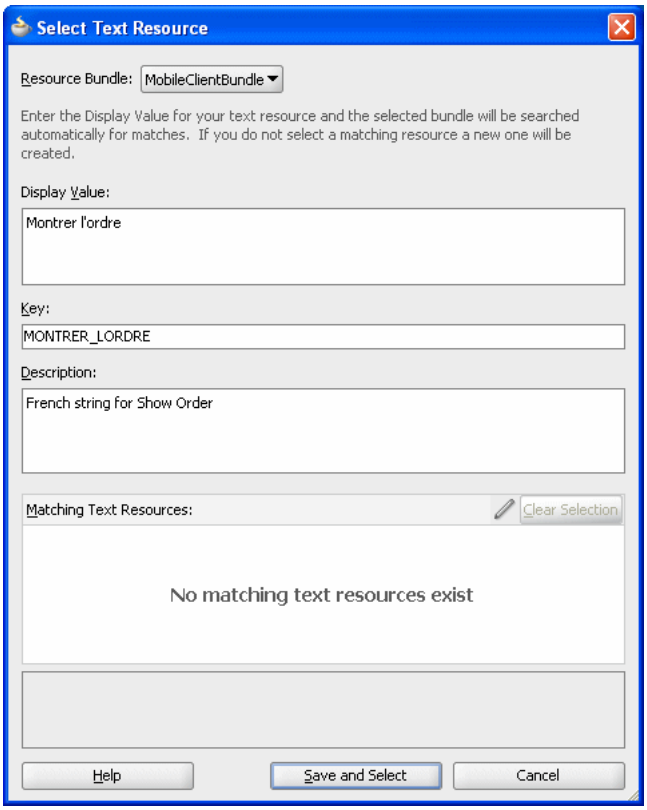


Figure 6–45 Localized String in MCX File

```
      <amc:commandLink text="#{mobileclientBundle['MONTRER_LORDRE']}"
                       id="commandButton2" action="orderitems"/>
    </amc:panelGroupLayout>
  </amc:panelGroupLayout>
  <amc:spacer id="spacer3" height="20"/>
  <amc:panelFormLayout id="panelFormLayout1">
    <amc:panelLabelAndMessage label="#{bindings.OrderId.hints.label}: "
                              id="panelLabelAndMessage8">
      <amc:outputText value="#{bindings.OrderId.inputValue}"
                      id="outputText9" focusable="true"/>
    </amc:panelLabelAndMessage>
```

For more information on localization, see the following:

- Section 5.3.12, "Working with Resource Bundles"
- Section 7.4, "Resource Bundle Support"
- Section 7.4.1, "Managing Locales Using the List ResourceBundle and PropertyResourceBundle Classes"

6.14 Understanding EL Support

ADF Mobile client provides support for Expression Language (EL).

6.14.1 Supported EL Nodes

ADF Mobile client supports the following EL nodes:

- Data bindings: `"#{bindings.*}"`
- Application Global Variable Scope: `"#{applicationScope.*}"`
- Sync Context: "#{syncContext.*}"
- Security Context: "#{securityContext.*}"
- Device Context: "#{device.*}"

6.14.2 What You May Need to Know About ADF Mobile Client EL Implementation

The ADF Mobile client 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

6.14.2.1 Immediate and Deferred Evaluation

In reference to "1.2.1: Eval-expression" in the specification:

In ADF Mobile client, 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 client supports only the deferred semantics, an expression using the immediate construction expression (`"${}"`) will still parse, but will behave the same as deferred expression (`"#{}"`).

6.14.2.2 Enumerated Types

As described "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.

6.14.3 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}"`. ADF Mobile client binding objects are referenced by name from the binding container `"#{bindings.Name}"`.

Table 6–8 lists the properties that you can use in EL expressions to access values of ADF Mobile client binding objects at run time. The table lists these properties in alphabetical order.

Table 6–8 Runtime Properties

Runtime Property	Description	Iterator	Action	Attribute
`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	NA	NA
`dataControl`	Returns the iterator's associated data control.	Yes	No	No
`dataProvider`	Returns the iterator's associated data provider.	Yes	NA	NA
`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).	NA	Yes	NA
`error`	Returns any exception that was cached while updating the associated attribute value for a a value binding or when invoking an operation bound by an operation binding.	No	No	Yes
`execute`	Invokes the named action or `methodAction` binding when resolved.	NA	Yes	No
`findMode`	Returns `true` if the iterator is currently operating in find mode. Otherwise, returns `false`.	Yes	NA	NA

Table 6–8 (Cont.) Runtime Properties

Runtime Property	Description	Iterator	Action	Attribute
`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: ■ `displayHeight`: The height in lines for the current attribute. ■ `displayWidth`: The width in characters for the current attribute. ■ `displayHint`: The display hint for the current attribute. ■ `label`: The label to display for the current attribute. ■ `format`: The format to be used for the current attribute. ■ `formatter`: The formatter object to be used for the current attribute. ■ `mandatory`: Returns true if a value is required for the current attribute. ■ `precision`: The precision to be used for the current attribute. ■ `tooltip`: The ToolTip text to display for the current attribute. ■ `updateable`: Returns true if the current attribute can be written to.	Yes	NA	Yes
`inputValue`	Returns or sets the value of the current attribute.	NA	NA	Yes
`items`	Returns the list of values associated with the current list enabled attribute.	NA	NA	No
`iteratorBinding`	Returns the iterator binding that provides access to the data collection.	NA	Yes	Yes
`label`	Returns the label (if supplied by control hints) for the first attribute of the binding.	No	No	Yes
`labelSet`	Returns an ordered set of labels for all the attributes to which the binding is associated.	No	NA	NA
`rangeSize`	Returns the range size of the iterator binding's row set.	Yes	NA	NA
`rangeStart`	Returns the absolute index in a collection of the first row in range.	Yes	NA	NA
`rowCount`	Returns the total number of rows in the collection.	Yes	NA	NA

Table 6–8 (Cont.) Runtime Properties

Runtime Property	Description	Iterator	Action	Attribute
`updateable`	Returns `true` if the current attribute is updateable.	NA	NA	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 client 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}`.

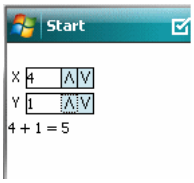
6.14.4 EL Events

EL events play a significant role in the functioning of the ADF Mobile client UI.

EL expressions can refer to values in various contexts. Example 6–35 shows the creation of two Input Number Spinbox 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 6.12, "Using Event Listeners") for the Output Text on those two values. Anytime the value of the underlying expression changes, an event is generated to all listeners for that value.

Example 6–35 Generating EL Events with Two Components

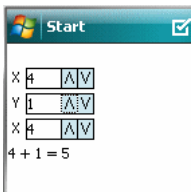
```
<amc:inputNumberSpinbox id="spin1" label="X" value="#{applicationScope.X}"/>
<amc:inputNumberSpinbox id="spin2" label="Y" value="#{applicationScope.Y}"/>
<amc:outputText id="ot1" value="#{applicationScope.X} +
       #{applicationScope.Y} = #{applicationScope.X + applicationScope.Y}"/>
```



In the preceding example two components are updating one value each, and one component is consuming both values. Example 6–36 shows that the behavior would be identical if a third Input Number Spinbox component is added that references one of the existing values.

Example 6–36 Generating EL Events with Three Components

```
<amc:inputNumberSpinbox id="spin1" label="X" value="#{applicationScope.X}"/>
<amc:inputNumberSpinbox id="spin2" label="Y" value="#{applicationScope.Y}"/>
<amc:outputText id="ot1" value="#{applicationScope.X} +
       #{applicationScope.Y} = #{applicationScope.X + applicationScope.Y}"/>
<amc:inputNumberSpinbox id="spin3" label="X" value="#{applicationScope.X}"/>
```



In the preceding example, when either Input Number Spinbox component updates `#{applicationScope.X}`, the other will automatically be updated along with the Output Text. All of the updates are automatic (no coding is required).

6.15 Understanding Binding Layer Components

ADF Mobile client interprets the ADF lifecycle in the context of a disconnected native application. As a consequence, the mobile client runtime uses the `default` behavior (illustrated in Example 6–37) 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 6–37 invokeAction in ADF Mobile Client

```
<invokeAction id="callSetCurrentRowWithKeyValue"
              Binds="setCurrentRowWithKeyValue"
              Refresh="default"
              RefreshCondition="#{requestScope.partyId != null}"/>
```

For information on the recognized values of the `Refresh` attribute and their meaning in a disconnected context, see Table 6–9, " Refresh Values and the Corresponding Conditions to Invoke".

The mobile client 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. Setting an attribute value or navigating to a new row are examples of this.

The runtime will execute an `invokeAction` on the above conditions according to values of its `Refresh` and `RefreshCondition` attributes, as listed in Table 6–9.

Table 6–9 Refresh Values and the Corresponding Conditions to Invoke

Refresh Value	Condition to invoke
`always`	A and B
`deferred`	Not supported on `invokeAction`. For more information, see "What You May Need to Know About Using the Refresh Property Correctly" section of *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.
`default`	A
`ifNeeded`	Not supported. Instead, use `always` and gate with a `RefreshCondition`.
`never`	Not supported
`prepareModel`	Not supported
`prepareModelIfNeeded`	Not supported
`renderModel`	Not supported
`renderModelIfNeeded`	Not supported

If a `RefreshCondition` expression is supplied, it will be evaluated at each potential execution of the `invokeAction`. If it evaluates to `false`, execution will be skipped on that occurrence.

> **Note:** Note that for iterator executables, `deferred` is the standard behavior, that is the iterator will not perform its initial `Refresh` until an EL expression that is dependent on it is evaluated.

6.15.1 What You May Need to Know About Sequencing

By default, `invokeActions` declared with the same `Refresh` value will execute in the order they are declared in the `pagedef`.

To override this behavior, a predecessor iterator or `invokeAction` can be specified with the `RefreshAfter` attribute. Note, however, that the predecessor will still execute or not based on its own `Refresh` and `RefreshCondition` values.

7

Extending ADF Mobile Client Applications with Java

This chapter includes the following sections:

7.1 About Invoking Custom Methods Through EL Expressions

ADF applications typically invoke custom methods defined on an application module or a view object in response to some event or action that occurs on a page. These custom method invocations are often declared in pages as EL expressions assigned to various component attributes, as illustrated by the EL expression `"#{bindings.Foo.execute}"` in Example 7–1.

Example 7–1 Declaring a Method Using EL Expressions

```
<amc:commandButton text="Foo" id="commandButton1"
actionListener="#{bindings.Foo.execute}"/>
```

This type of declarative invocation assumes that the platform on which the application executes binds dynamically and invokes methods by name rather than compiling the method invocation directly into the object code that executes at runtime.

On J2SE- and J2EE-based platforms, dynamic method invocation is enabled through Java reflection (`java.lang.reflect`) classes which use underlying VM support to dynamically link to a method call by name. ADF Mobile client's J2ME platform, unlike J2SE- or J2EE-based platforms, does not have this VM support to implement Java reflection. Consequently, ADF Mobile client applications do not bind methods dynamically; they instead link methods statically at compile time.

7.1.1 Adding Invocation Code for Custom Methods in Application Modules and View Objects

Because the ADF Mobile client framework cannot rely on Java reflection to bind to methods dynamically, it requires the application to provide information about the method to be called. In addition, for the framework to call the method, it must call through a generic interface to code that is linked at compile time to the method call. The framework infrastructure accomplishes these through the bindings declaration in application metadata defined for the page and through the `oracle.adfnmc.util.MethodDispatch` interface.

The ADF Mobile client framework simulates binding to methods dynamically by calling abstractly through `MethodDispatch.invokeMethod`, passing information about the method to be called, and relying on the application to provide the concrete code that links the names to compile time method calls. `MethodDispatch` should be implemented on each application module or view object implementation class that uses custom Java methods.

You can add custom methods using the overview editors for application modules or view objects as follows:

1. Add a method to the application module or view object.

   Access the overview editor by either choosing **Open** from the context menu in the Application Navigator, or by double-clicking the application module or view object. For more information on using the overview editor, see "Creating and Modifying an Application Module" and "Generating Custom Java Classes for a View Object" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

2. Expose this method to the client interface.

3. Implement `MethodDispatch.invokeMethod` on the application module or view object as illustrated in Example 7–2.

Example 7–2 Implementing the MethodDispatch Interface

```
public class MyVOImpl
  extends ViewObjectImpl
  implements MyVO, MethodDispatch
{
  public String myCustomMethod(String param1, boolean param2)
  {
 // implementation...
  }

  public Object invokeMethod(String methodName, Object[] params)
  {
    if (methodName.equals("myCustomMethod"))
    {
  String param1 = (String)params[0];
  boolean param2 = ((Boolean)params[1]).booleanValue())
  return this.myCustomMethod(param1, param2);
    }
  }
}
```

7.2 Java Support for Business Components

ADF Mobile client supports using Java to extend and customize the functionality delivered by the metadata of the standard business components. While you can employ many of the same techniques and high-level API calls that you use in standard ADF Faces applications, the target mobile device platforms impose constraints in the following areas when you develop ADF Mobile client applications:

- Support for Reflection
- JDK 1.3 Compliance
- Alternate Package Names

7.2.1 Support for Reflection

Only the following APIs are available in this category:

- `Class.forName(String)`
- `Class.newInstance()`

7.2.2 JDK 1.3 Compliance

All source code is compiled for compatibility with a JDK 1.3 JVM. As a consequence, there is no support for Java 5 language features, including:

- Generics
- Iterators
- Annotations
- Enumerations

7.2.3 Alternate Package Names

In addition to missing language features, the supported target platforms also lack a number of classes used in developing JavaSE and JavaEE platforms, including JDBC for interfacing with relational databases and collections such as `HashMap` and `ArrayList`. To offset these gaps in functionality, ADF Mobile client provides its own implementations of these constructs, which augment the target platforms' base class libraries. However, because some target devices do not permit modifying the JVM bootclasspath, there is no provision for placing these implementations in the standard `java.*` or `javax.*` package hierarchies. You must instead import these classes from the alternate `oracle.adfnmc.java.*` or `oracle.adfnmc.javax.*` package hierarchies.

For example, instead of:

```
import java.sql.ResultSet;
import java.util.HashMap;
```

Use:

```
import oracle.adfnmc.java.sql.ResultSet;
import oracle.adfnmc.java.util.HashMap;
```

Then use the classes normally:

```
HashMap map = new HashMap();
map.put("Key", "Value");
ResultSet rs = preparedStatement.executeQuery();
```

7.2.4 Supported Java Extension Points for Business Components

Java source code can be used to extend the functionality of various business components, including entity objects, view objects, and application modules. Table 7–1 lists both the ADF Mobile client's supported and unsupported extension points.

Table 7–1 Supported and Unsupported Extension Points

Supported	Not Supported
Entity Object Class	Entity Collection Class
	Entity Definition Class
View Object Class	Service Data Object Class
View Row Class	View Client Row Interface
View Object Client Interface	
View Object Definition Class	
Application Module Class	Application Module Definition Class
Application Module Client Interface	
Application Module Client Class	

7.2.4.1 Unsupported Methods

Not every method in BC4J is available for ADF Mobile client. To determine which methods are supported:

1. Use JDeveloper code completion and syntax highlighting features to determine what is supported at compile-time instead of repeatedly consulting a lengthy list of supported methods.
2. Consult Table 7–2 that lists the unsupported methods.

Table 7–2 Unsupported Methods

Class	Supported Methods
`ViewObjectImpl`	▪ `activateNewRowTracker` ▪ `activateState` ▪ `addEffectiveDateDstAttributes` ▪ `buildEffectiveDateFromClauseFragment`
`EntityImpl`	▪ `doDMLWithLOBs` ▪ `handleEffectiveDateOperations` ▪ `validateDateEffectively` ▪ `removeAndRetain`
`ApplicationModuleImpl`	▪ `activateState` ▪ `passivateStateForUndo`

7.3 Using a Managed Bean in an ADF Mobile Client Application

You can create and use managed beans (Mbeans) in an ADF Mobile client application to store additional data or to execute custom code. Adding an MBean to an ADF Mobile client application is similar to adding a bean to a standard Fusion Web application, except for the following differences:

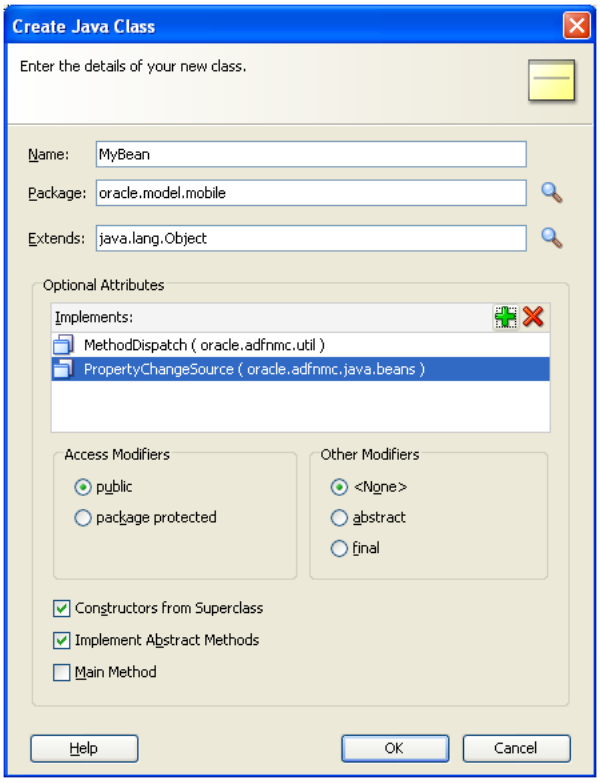
- The MBean class must implement `oracle.adfnmc.util.MethodDispatch` or `oracle.adfnmc.util.PropertiesDispatch`.
- The MBean class must implement `oracle.adfnmc.java.beans.PropertyValueChangeSource`.
- The only valid scopes declared in an MBean are `application`, `backingBean`, and `none`.
- The managed property default values not presently supported.

For more information, see "Using a Managed Bean in a Web Fusion Application" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To add a managed bean:

1. In JDeveloper, click **File** then **New**.
2. Choose **Java** and then **Java Class** and then click **OK**.
3. In the Create Java Class dialog, enter a name for the Java class.
4. In the **Implements** field, click **Add** to open the Class and Package Browser and use it to locate `oracle.adfnmc.util.MethodDispatch`.
5. Click **Add** to open the Class and Package Browser and use it to locate `oracle.adfnmc.java.beans.PropertyValueChangeSource`.
6. Click **OK**.

Figure 7–1 Adding Custom Properties to a Java Class



7. Use the overview editor for the ADF Mobile client task flow to add this Java class as a managed bean as follows:

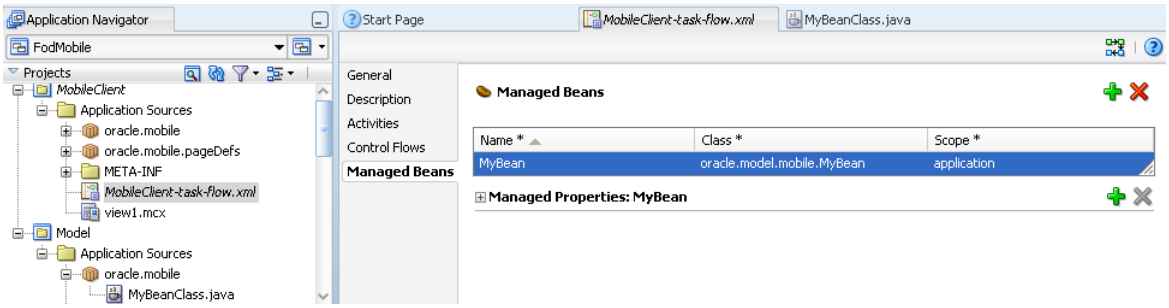
a. In the Application Navigator, select the task flow (typically `MobileClient-task-flow.xml`), click the **Overview** tab and then click **Managed Beans**.

b. In the Managed Beans page, click **New**.

c. Enter a name for the managed bean.

d. Enter the name of the Java class created in Steps 1 through 5.

Figure 7–2 Creating a Managed Bean



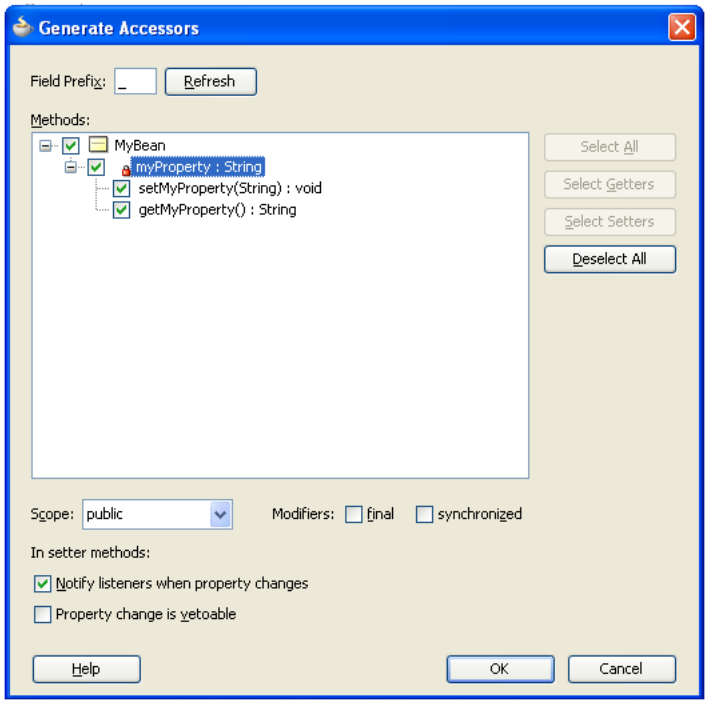
8. Add custom methods and properties to the Java class.

a. To add a custom property, declare the backing member variable for that property and then select **Source** and then **Generate Accessors**.

b. Select each member for which you wish to generate accessors, as illustrated in Figure 7–3.

c. Select **Notify Listeners when property changes**.

Figure 7–3 Adding Custom Methods and Properties



9. Because reflection is not supported in the J2ME version on some platforms such as BlackBerry, provide dispatch code if you want to invoke or access any of the methods or properties from EL. Example 7–3 illustrates using `invokeMethod`.

7.3.1 About MethodDispatch and PropertyDispatch

The managed bean must implement `oracle.adfnmc.util.MethodDispatch`. Because accessing properties and calling methods through EL expressions assumes that the platform on which the application executes binds dynamically and invokes methods by name rather than by compiling the method invocation directly into the object code that executes at runtime, this interface declares a single method, `invokeMethod`, that must be implemented to resolve names to actual property accessors or method calls.

On J2SE- and J2EE-based platforms, dynamic method invocation is enabled through the use of the `java.lang.reflect` classes which use underlying VM support to dynamically link to a method call by name. This is not the case for J2ME platforms, which lack the VM support required to implement the reflection classes. On this platform, methods can only be linked statically at compile time. The ADF Mobile client framework simulates binding to methods dynamically by calling abstractly through `oracle.adfnmc.util.MethodDispatch.invokeMethod`, passing information about the method to be called, and relying on the application to provide the concrete code that links the names to compile-time method calls.

Note: while the second argument to `invokeMethod` is a parameter list to be passed to the named method, with the exception of a "set" property accessor, this argument is not used when resolving a bean method using EL expressions. The ADF Mobile client EL syntax does not support an explicit parameter list when referencing a method. However, values can be passed indirectly to the method through object variable scopes, and accessed within the method implementation.

`oracle.adfnmc.util.PropertyDispatch` derives from `MethodDispatch` and can be implemented to add support for type conversions for properties of types other than `String` through the use of the `getType(String)` method. This method returns the type of a property given its name. Because J2ME platforms do not support reflection, using EL expressions to reference these properties from certain UI components my require using the `getType(String)` method. Example 7–3 provides a sample implementation of `invokeMethod` and `getType(String)`.

Note: You must invoke `invokeMethod` regardless.

7.3.2 About PropertyValueChangeSource and Notifications

`oracle.adfnmc.java.beans.PropertyValueChangeSource` is the interface through which ADF Mobile client EL expressions register themselves as listeners on the objects that they reference, enabling all occurrences of the same expression to be updated when the underlying value to which they evaluate changes. The ADF Mobile client UI components rely on this notification mechanism to keep the view of the active form up to date when values referenced by EL expressions change in the application through normal interaction.

Although implementing this interface is not required to simply reference bean methods or properties through EL expressions, the rendering of any EL expressions in the active form that depend on values stored in the bean must be kept up to date if those values change.

JDeveloper can automatically generate the necessary code to source notifications from your bean's property accessors by selecting **Notify listeners when property changes** in the Generate Accessors dialog, illustrated in Figure 7–3.

Note: You must manually declare `implements PropertyChangeSource` in your class. The generated code does not currently add this, and it is required for ADF Mobile client runtime support. Example 7–3 illustrates `implements PropertyChangeSource`.

You must also change the package in the imports declarations of `PropertyChangeListener`, `PropertyChangeSource`, and `PropertyChangeSupport` from `java.beans` to `oracle.adfnmc.java.beans`.

Example 7–3 Using PropertyDispatch

```
package adfmc.managedbean;

import oracle.adfnmc.component.MessageBox;
import oracle.adfnmc.event.ClientEvent;
import oracle.adfnmc.java.beans.PropertyChangeListener;
import oracle.adfnmc.java.beans.PropertyChangeSource;
import oracle.adfnmc.java.beans.PropertyChangeSupport;
import oracle.adfnmc.java.sql.Timestamp;
import oracle.jbo.domain.Number;
import oracle.adfnmc.util.PropertyDispatch;

// Instead of implementing MethodDispatch here, we are using
// PropertyDispatch which derives from MethodDispatch.  It adds the
// getType method.  This method allows you to return the correct types
// for each expression so the type conversions can be invoked
// automatically.
// PropertyChangeSource allows you to fire notifications when your
// member variables change so other components bound to the same
// properties also get updated.
public class Sample2 implements PropertyChangeSource, PropertyDispatch {

    private transient PropertyChangeSupport propertyChangeSupport =
        new PropertyChangeSupport(this);

    public Sample2() {
    }

    private String prop1 = "";
    private Number prop2 = new Number(0);
    private Timestamp prop3 = new Timestamp(0);
    private Boolean prop4 = Boolean.FALSE;
    private String prop5 = "";

    public Object invokeMethod(String methodName, Object[] params) {
        if (methodName.equals("MyMethod")) {
            Object actionEvent = params[0];
            this.MyMethod((ClientEvent)actionEvent);
```

```
            return null;
        } else if (methodName.equals("prop1")) {
            if (params.length == 0) {
                return this.getProp1();
            } else {
                Object value = params[0];
                this.setProp1((String)value);
                return null;
            }
        } else if (methodName.equals("prop2")) {
            if (params.length == 0) {
                return this.getProp2();
            } else {
                Object value = params[0];
                this.setProp2((Number)value);
                return null;
            }
        } else if (methodName.equals("prop3")) {
            if (params.length == 0) {
                return this.getProp3();
            } else {
                Object value = params[0];
                this.setProp3((Timestamp)value);
                return null;
            }
        } else if (methodName.equals("prop4")) {
            if (params.length == 0) {
                return this.getProp4();
            } else {
                Object value = params[0];
                this.setProp4((Boolean)value);
                return null;
            }
        } else if (methodName.equals("prop5")) {
            if (params.length == 0) {
                return this.getProp5();
            } else {
                Object value = params[0];
                this.setProp5((String)value);
                return null;
            }
        }
        return null;
    }

    // Note that you cannot use the java.bean namespace for these methods
    // and they must use the oracle.adfnmc versions.  The implementations
    // are the same though.
    public void addPropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.addPropertyChangeListener(l);
    }

    public void removePropertyChangeListener(PropertyChangeListener l) {
        propertyChangeSupport.removePropertyChangeListener(l);
    }

    public void MyMethod(ClientEvent valueChangeEvent) {
        MessageBox.show("MyMethod fired!");
    }
```

```
public void setProp1(String prop1) {
    String oldProp1 = this.prop1;
    this.prop1 = prop1;
    propertyChangeSupport.firePropertyChange("Prop1", oldProp1, prop1);
}

public String getProp1() {
    return prop1;
}

public void setProp2(Number prop2) {
    Number oldProp2 = this.prop2;
    this.prop2 = prop2;
    propertyChangeSupport.firePropertyChange("Prop2", oldProp2, prop2);
}

public Number getProp2() {
    return prop2;
}

public void setProp3(Timestamp prop3) {
    Timestamp oldProp3 = this.prop3;
    this.prop3 = prop3;
    propertyChangeSupport.firePropertyChange("Prop3", oldProp3, prop3);
}

public Timestamp getProp3() {
    return prop3;
}

public void setProp4(Boolean prop4) {
    Boolean oldProp4 = this.prop4;
    this.prop4 = prop4;
    propertyChangeSupport.firePropertyChange("Prop4", oldProp4, prop4);
}

public Boolean getProp4() {
    return prop4;
}

public void setProp5(String prop5) {
    String oldProp5 = this.prop5;
    this.prop5 = prop5;
    propertyChangeSupport.firePropertyChange("Prop5", oldProp5, prop5);
}

public String getProp5() {
    return prop5;
}

// For each of the EL properties you can return a type.  This allows
// the framework to do automatic type conversions.  If you only have
// string types, this method isn't necessary.
public Class getType(String propertyName) {
    if (propertyName.equals("prop1")) {
        return String.class;
    } else if (propertyName.equals("prop2")) {
        return Number.class;
    } else if (propertyName.equals("prop3")) {
        return Timestamp.class;
```

```
        } else if (propertyName.equals("prop4")) {
            return Boolean.class;
        } else if (propertyName.equals("prop5")) {
            return String.class;
        }
        return Object.class;
    }

}
```

7.4 Resource Bundle Support

As described in Section 5.3.12, "Working with Resource Bundles," you define the resource bundle configuration through the Project Properties as shown in Figure 5–15. When creating the user interface, you define the string references using JDeveloper's Select Text Resource dialog, which, when completed, results the creation of EL in the UI components that reference these strings. For more information, see Section 6.13, "Localizing UI Components" and "Working with Resource Bundles" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

JDeveloper makes the complete ADF resource bundle containing localized strings available during application compilation and deployment. The resource bundle is translated to the format that is required by the run time. See also Section 8.6, "Deploying a Multi-Language ADF Mobile Client Application." For a list of country names and languages used in resource bundle files, see Appendix A, "Language Abbreviations."

7.4.1 Managing Locales Using the List ResourceBundle and PropertyResourceBundle Classes

ADF Mobile client supports the JavaSE `ListResourceBundle` and `PropertyResourceBundle` classes, but you cannot use these classes as you would in JavaSE or JavaEE. Specifically, you cannot import these classes from the `java.util`. package as shown in Example 7–4 because they are not supported on mobile devices.

Example 7–4 Importing from the java.util Package

```
import java.util.ListResourceBundle;
import java.util.PropertyResourceBundle;
```

Instead, import these resource bundle classes from `oracle.adfmnc.java.util` as shown in Example 7–5.

Example 7–5 Importing from oracle.adfnmc.java.util Package

```
import oracle.adfnmc.java.util.ListResourceBundle;
import oracle.adfnmc.java.util.PropertyResourceBundle;
```

You use these constructs in ADF Mobile client as you would in Java SE or Java EE environments. You can provide localization of selected `String` resources using properties files that include settings appropriate to both non-localized and localized display. The names of the properties files distinguish localized and non-localized resources. For example, Example 7–6 illustrates the non-localized properties file called `localizationStrings.properties` and Example 7–7 illustrates the localized settings in an English-only properties file called `localizationString_`

`en.properties`. The `_en` locale identifier appended to the file name denotes that this properties file is English-only.

Example 7–6 Settings in a Non-Localized File, localizationStrings.properties

```
errorMsg=Problem has occurred
greeting=To Whom It May Concern
```

Example 7–7 shows the settings in a localized, English-only properties file.

Example 7–7 Settings in a Localized File, localizationStrings_en.properties.

```
greeting=Hey there
```

Example 7–8 illustrates the Java code that accesses these properties. The non-localized line always returns a *Problem has occurred* string. The localized line (*Hey There!*) is returned on English language-supported platforms, while *To Whom It May Concern* is returned on all other locales.

Example 7–8 Accessing Properties

```
ResourceBundle resourceBundle = ResourceBundle.getBundle("some.package.foo");
System.out.println(resourceBundle.getString("errorMsg"); // non-localized; always
returns "Problem has occurred"
System.out.println(resourceBundle.getString("greeting"); // localized. returns
"Hey there" on English platforms; "To Whom It May Concern" on all other locales
```

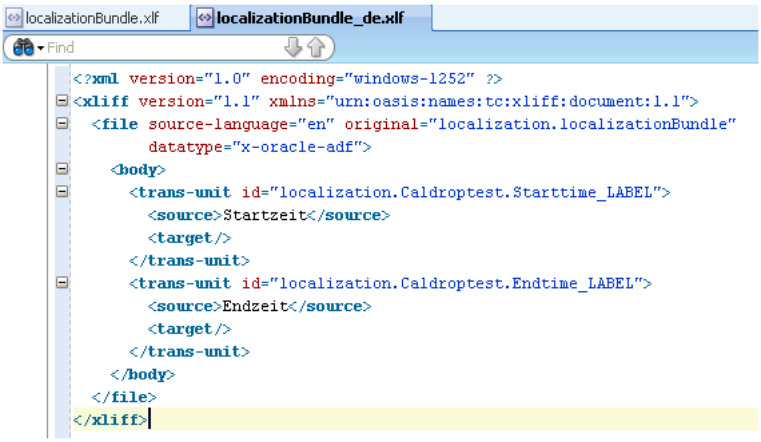
These resource bundles are also used implicitly at run time for attribute control hints. For example, if there is an attribute in a `ViewObject` named `PersonID` and `PersonID=Custom Label` is defined in the Model project's resource bundle, then at run time, whenever this attribute is displayed, it uses the custom label for the label property rather than defaulting to the name of the attribute.

7.4.2 Supporting Localization through XLFF Resource Bundles

ADF Mobile client supports localization by resolving XLIFF (XML Localization Interchange File Format) resource bundles in addition to its support of `.properties` resource bundles. After ADF Mobile client imports the resource bundles from the base (server) application and resolves them, it populates attributes with the appropriate values taken from the JAR of the base application.

Figure 7–5 shows the format of an XLF bundle called `localizationBundle_de.xml`, which is in the German language (as noted by `_de`).

Figure 7–4 A Localization Bundle



The following examples demonstrate how ADF Mobile client supports localization by resolving a resource bundle in English (the default) and one in German.

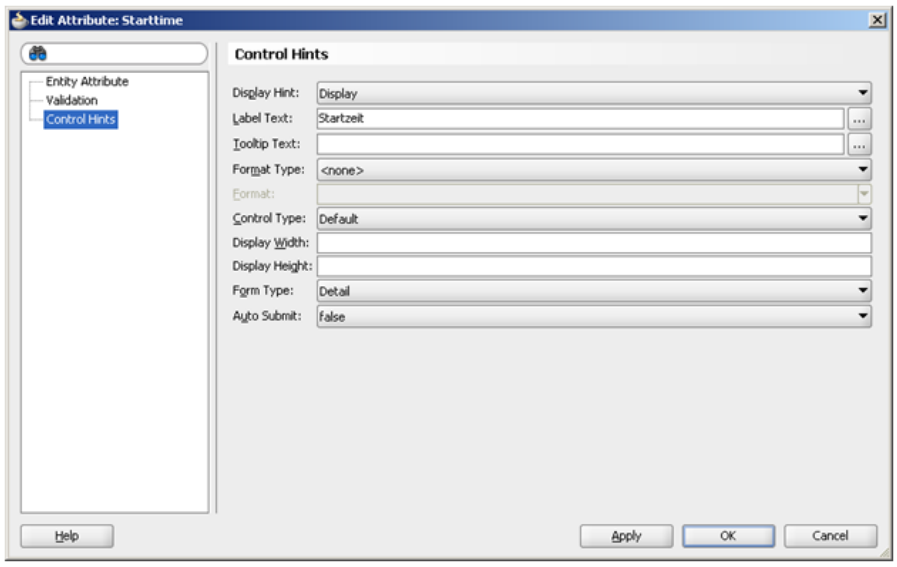
Example 7–9, the source code for an attribute called `Starttime`, illustrates the results of importing a German language resource bundle named `localizationBundle_de` from the base (server) application's JAR file: within the `LABEL` tag for `Starttime` attribute, ADF Mobile client retrieved the value for `ResID` is defined as `${adfBundle['localization.localizationBundle_de'][localization.Caldroptest.Starttime_LABEL']}` from the base application's JAR.

Example 7–9 The Starttime Attribute

```
<Attribute
  Name="Starttime"
  ColumnName="STARTTIME"
  SQLType="TIMESTAMP"
  Type="oracle.jbo.domain.Date"
  ColumnType="DATE"
  TableName="CALDROPTEST">
  <DesignTime>
     <Attr Name="_DisplaySize" Value="7"/
  </DesignTime>
  <Properties>
     <SchemaBasedProperties>
         <LABEL
            ResId=${adfBundle['localization.localizationBundle_
de'][localization.Caldroptest.Starttime_LABEL']}
      </SchemaBasedProperties>
   </Properties>
</Attribute>
```

As shown in Figure 7–5, viewing the attribute displayed in the editor (accessed through the overview editor for the entity object), verifies that ADF Mobile client has retrieved the correct string value from the German language file, as the Label Text value is defined as `Startzeit`.

Figure 7–5 Control Hints for Start Time (Startzeit)



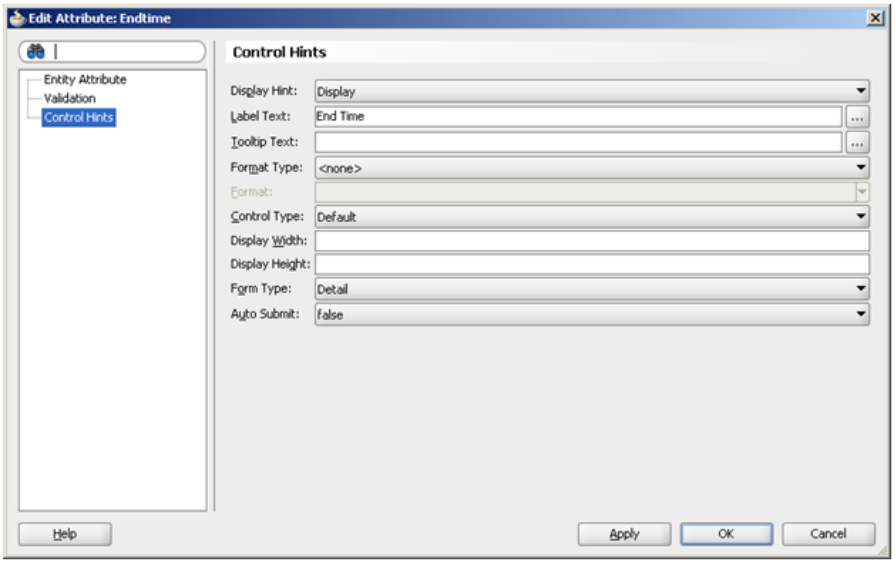
In addition to the German language resource bundle used for the `Startzeit` attribute, the `Endtime` attribute uses the default resource bundle, which is in English. The value of `LABEL` in Figure 7–10 shows that ADF Mobile client likewise imported the resource bundle from the JAR, although this one is the default (`${adfBundle['localization.localizationBundle']['localization.Caldroptest.Endtime_LABEL']}`).

Example 7–10 The Endtime Attribute

```
<Attribute
  Name="Endtime"
  ColumnName="ENDTIME"
  SQLType="TIMESTAMP"
  Type="oracle.jbo.domain.Date"
  ColumnType="DATE"
  TableName="CALDROPTEST">
  <DesignTime>
     <Attr Name="_DisplaySize" Value="7"/
  </DesignTime>
  <Properties>
     <SchemaBasedProperties>
         <LABEL
          ResId=${adfBundle['localization.localizationBundle']['localization.Caldroptest.Endtime_
LABEL']}
      </SchemaBasedProperties>
   </Properties>
 <Attribute>
```

As shown in Figure 7–6, the attribute editor displays the Label Text value as `End Time`.

Figure 7–6 Control Hints for End Time



After deployment to a device, such as BlackBerry smartphone in Figure 7–7, the application displays the column labels derived from the German language and default, English language, resource bundles.

Figure 7–7 Localization of Column Names

7.5 Supported EL Nodes

Supported EL nodes include:

- Data bindings - `"#{bindings.*}"`

- Application global variable scope - `"#{applicationScope.*}"`

> **Note:** `applicationScope` variables, such as beans, can be called either with the `applicationScope` prefix (`#(applicationScope.Bean.method)`) or without `#(Bean.method)`.

- Sync Context - `"#{syncContext.*}"`
- Security Context - `"#{securityContext.*}"`
- Device Context - `"#{device.*}"`

7.5.1 Working with EL in Code

EL Expression strings can be evaluated programmatically from Java code to resolve property values and invoke methods. EL Expressions are represented at runtime as objects that implement either the `ValueExpression` or `MethodExpression` interface. Expression objects are created by passing the EL expression string and some type information to factory methods on an ELContext, as shown in Example 7–11 and Example 7–12.

Example 7–11 illustrates using the `ValueExpression` interface to get or set a named value identified by an EL expression string.

Example 7–11 Using the ValueExpression Interface

```
import oracle.adfnmc.el.ValueExpression;
import oracle.adfnmc.el.impl.SimpleContext;
...

Class expectedType = String.class;
String expressionStr = "#{applicationScope.testBean.testProperty}";

ValueExpression expr = SimpleContext.getValueExpression(expressionStr,
expectedType);
String value = (String) testBeanExpr.getValue();
testBeanExpr.setValue("newValue");
```

Example 7–12 illustrates using the `MethodExpression` interface to invoke a method identified by an EL expression string:

Example 7–12 Using the MethodExpression Interface

```
import oracle.adfnmc.el.impl.SimpleContext;
import oracle.adfnmc.el.MethodExpression;
...

// invoke "String testMethod(String param)" on TestBean.java instance
Class expectedReturnType = String.class;
Class[] expectedParamTypes = new Class[] { String.class };
String expressionStr = "#{applicationScope.testBean.testMethod}";

MethodExpression expr = SimpleContext.getMethodExpression(expressionStr,
expectedReturnType, expectedParamTypes);
String retVal  = (String)expr.invoke(new Object[] { "param1Value" });
```

Method parameters are not declared directly in EL method expression strings. The parameter values may come from other areas, such as EL value expressions or data members in a bean. It is assumed that the caller will have their actual values available to pass at invocation.

7.6 Additional JavaSE Classes Provided by the ADF Mobile Client Framework

Table 7–3 lists JavaSE classes provided by ADF Mobile client.

Table 7–3 Additional JavaSE Classes

JavaSE Class	ADF Mobile Client Equivalent
java.lang.Math	`oracle.adfnmc.java.lang.MathHelper`
`java.sql.Date`	`oracle.adfnmc.java.sql.Date`
`java.sql.DriverPropertyInfo`	`oracle.adfnmc.java.sql.DriverPropertyInfo`
`java.sql.Time`	`oracle.adfnmc.java.sql.Time`
`java.sql.Timestamp`	`oracle.adfnmc.java.sql.Timestamp`
`java.text.SimpleDateFormat`	`oracle.adfnmc.java.text.SimpleDateFormat`
`java.util.Comparator`	`oracle.adfnmc.java.util.Comparator`
`java.util.Locale`	`oracle.adfnmc.java.util.Locale`
`java.io.BufferedReader`	`oracle.adfnmc.java.io.BufferedReader`
`java.io.File`	`oracle.adfnmc.java.io.File`
`java.io.FileInputStream`	`oracle.adfnmc.java.io.FileInputStream`
`java.io.FileOutputStream`	`oracle.adfnmc.java.io.FileOutputStream`
`java.io.LineNumberReader`	`oracle.adfnmc.java.io.LineNumberReader`
`java.io.Serializable`	`oracle.adfnmc.java.io.Serializable`
`java.io.StreamCorruptedException`	`oracle.adfnmc.java.io.StreamCorruptedException`
`java.io.StringReader`	`oracle.adfnmc.java.io.StringReader`
`java.lang.AssertionError`	`oracle.adfnmc.java.lang.AssertionError`
`java.lang.Class`	`oracle.adfnmc.java.lang.ClassUtils`
`java.lang.Comparable`	`oracle.adfnmc.java.lang.Comparable`
`java.util.BitSet`	`oracle.adfnmc.java.util.BitSet`
`java.util.Comparator`	`oracle.adfnmc.java.util.Comparator`
`java.util.EventListener`	`oracle.adfnmc.java.util.EventListener`
`java.util.EventObject`	`oracle.adfnmc.java.util.EventObject`
`java.util.Locale`	`oracle.adfnmc.java.util.Locale`
`java.util.Properties`	`oracle.adfnmc.java.util.Properties`
`java.io.BufferedInputStream`	`oracle.adfnmc.java.io.BufferedInputStream`
`java.io.BufferedOutputStream`	`oracle.adfnmc.java.io.BufferedOutputStream`
`java.io.Closeable`	`oracle.adfnmc.java.io.Closeable`
`java.io.FileNotFoundException`	`oracle.adfnmc.java.io.FileNotFoundException`
`java.io.FilterInputStream`	`oracle.adfnmc.java.io.FilterInputStream`
`java.io.FilterOutputStream`	`oracle.adfnmc.java.io.FilterOutputStream`

Table 7–3 (Cont.) Additional JavaSE Classes

JavaSE Class	ADF Mobile Client Equivalent
java.io.Flushable	oracle.adfnmc.java.io.Flushable
java.io.PrintWriter	oracle.adfnmc.java.io.PrintWriter
java.io.StringWriter	oracle.adfnmc.java.io.StringWriter
java.io.Writer	oracle.adfnmc.java.io.Writer
java.lang.Appendable	oracle.adfnmc.java.lang.Appendable
java.lang.Boolean	java.lang.Boolean, oracle.adfnmc.java.lang.BooleanHelper
java.lang.Byte	java.lang.Byte, oracle.adfnmc.java.lang.ByteHelper
java.lang.Character	java.lang.Character, oracle.adfnmc.java.lang.CharacterHelper
java.lang.CharSequence	java.lang.Char, oracle.adfnmc.java.lang.CharSequence
java.lang.Double	java.lang.Double, oracle.adfnmc.java.lang.DoubleHelper
java.lang.Integer	java.lang.Integer, oracle.adfnmc.java.lang.IntegerHelper
java.lang.Iterable	oracle.adfnmc.java.lang.Iterable
java.lang.Long	java.lang.Long, oracle.adfnmc.java.lang.LongHelper
java.lang.StringBuilder	oracle.adfnmc.java.lang.StringBuilder
java.lang.String	java.lang.String, oracle.adfnmc.java.lang.StringHelper
java.lang.UnsupportedOperationException	oracle.adfnmc.java.lang.UnsupportedOperationException
java.math.BigDecimal	oracle.adfnmc.java.math.BigDecimal
java.math.BigInteger	oracle.adfnmc.java.math.BigInteger
java.math.BitLevel	oracle.adfnmc.java.math.BitLevel
java.sql.Array	oracle.adfnmc.java.sql.Array
java.sql.BaseDate	oracle.adfnmc.java.sql.BaseDate
java.sql.BaseTime	oracle.adfnmc.java.sql.BaseTime
java.sql.BatchUpdateException	oracle.adfnmc.java.sql.BatchUpdateException
java.sql.Blob	oracle.adfnmc.java.sql.Blob
java.sql.CallableStatement	oracle.adfnmc.java.sql.CallableStatement
java.sql.Clob	oracle.adfnmc.java.sql.Clob
java.sql.Connection	oracle.adfnmc.java.sql.Connection
java.sql.DatabaseMetaData	oracle.adfnmc.java.sql.DatabaseMetaData
java.sql.Driver	oracle.adfnmc.java.sql.Driver
java.sql.DriverManager	oracle.adfnmc.java.sql.DriverManager
java.sql.ParameterMetaData	oracle.adfnmc.java.sql.ParameterMetaData
java.sql.PreparedStatement	oracle.adfnmc.java.sql.PreparedStatement
java.sql.Ref	oracle.adfnmc.java.sql.Ref
java.sql.ResultSet	oracle.adfnmc.java.sql.ResultSet
java.sql.ResultSetMetaData	oracle.adfnmc.java.sql.ResultSetMetaData
java.sql.Savepoint	oracle.adfnmc.java.sql.Savepoint
java.sql.SQLException	oracle.adfnmc.java.sql.SQLException

Table 7–3 (Cont.) Additional JavaSE Classes

JavaSE Class	ADF Mobile Client Equivalent
java.sql.SQLWarning	oracle.adfnmc.java.sql.SQLWarning
java.sql.Statement	oracle.adfnmc.java.sql.Statement
java.sql.Struct	oracle.adfnmc.java.sql.Struct
java.sql.Types	oracle.adfnmc.java.sql.Types
java.text.AttributedCharacterIterator	oracle.adfnmc.java.text.AttributedCharacterIterator
java.text.CharacterIterator	oracle.adfnmc.java.text.CharacterIterator
java.text.FieldPosition	oracle.adfnmc.java.text.FieldPosition
java.text.Format	oracle.adfnmc.java.text.Format
java.text.MessageFormat	oracle.adfnmc.java.text.MessageFormat
java.text.ParseException	oracle.adfnmc.java.text.ParseException
java.text.ParsePosition	oracle.adfnmc.java.text.ParsePosition
java.util.AbstractCollection	oracle.adfnmc.java.util.AbstractCollection
java.util.AbstractList	oracle.adfnmc.java.util.AbstractList
java.util.AbstractMap	oracle.adfnmc.java.util.AbstractMap
java.util.AbstractSequentialList	oracle.adfnmc.java.util.AbstractSequentialList
java.util.AbstractSet	oracle.adfnmc.java.util.AbstractSet
java.util.ArrayList	oracle.adfnmc.java.util.ArrayList
java.util.Arrays	oracle.adfnmc.java.util.Arrays
java.util.Calendar	java.util.Calendar, oracle.adfnmc.java.util.CalendarHelper
java.util.Collection	oracle.adfnmc.java.util.Collection
java.util.ConcurrentModificationException	oracle.adfnmc.java.util.ConcurrentModificationException
java.util.Date	java.util.Date, oracle.adfnmc.java.util.DateHelper
java.util.Dictionary	oracle.adfnmc.java.util.Dictionary
java.util.EmptyStackException	oracle.adfnmc.java.util.EmptyStackException
java.util.HashMap	oracle.adfnmc.java.util.HashMap
java.util.HashSet	oracle.adfnmc.java.util.HashSet
java.util.Hashtable	oracle.adfnmc.java.util.Hashtable
java.util.Iterator	oracle.adfnmc.java.util.Iterator
java.util.LinkedHashMap	oracle.adfnmc.java.util.LinkedHashMap
java.util.LinkedList	oracle.adfnmc.java.util.LinkedList
java.util.List	oracle.adfnmc.java.util.List
java.util.ListIterator	oracle.adfnmc.java.util.ListIterator
java.util.Map	oracle.adfnmc.java.util.Map
java.util.MapEntry	oracle.adfnmc.java.util.MapEntry
java.util.MissingResourceException	oracle.adfnmc.java.util.MissingResourceException
java.util.Queue	oracle.adfnmc.java.util.Queue
java.util.Random	oracle.adfnmc.java.util.Random
java.util.RandomAccess	oracle.adfnmc.java.util.RandomAccess

Table 7–3 (Cont.) Additional JavaSE Classes

JavaSE Class	ADF Mobile Client Equivalent
java.util.Set	oracle.adfnmc.java.util.Set
java.util.SortedMap	oracle.adfnmc.java.util.SortedMap
java.util.SortedSet	oracle.adfnmc.java.util.SortedSet
java.util.Stack	oracle.adfnmc.java.util.Stack
java.util.StringTokenizer	oracle.adfnmc.java.util.StringTokenizer
java.util.TreeMap	oracle.adfnmc.java.util.TreeMap
java.util.TreeSet	oracle.adfnmc.java.util.TreeSet
java.util.Vector	oracle.adfnmc.java.util.Vector
java.util.WeakHashMap	oracle.adfnmc.java.util.WeakHashMap This API is compatible, but it does not maintain weak reference semantics.
java.util.logging.ConsoleHandler	oracle.adfnmc.java.util.logging.ConsoleHandler
java.util.logging.DevNullHandler	oracle.adfnmc.java.util.logging.DevNullHandler
java.util.logging.DevNullOutputStream	oracle.adfnmc.java.util.logging.DevNullOutputStream
java.util.logging.ErrorManager	oracle.adfnmc.java.util.logging.ErrorManager
java.util.logging.FileHandler	oracle.adfnmc.java.util.logging.FileHandler
java.util.logging.Filter	oracle.adfnmc.java.util.logging.Filter
java.util.logging.Formatter	oracle.adfnmc.java.util.logging.Formatter
java.util.logging.Handler	oracle.adfnmc.java.util.logging.Handler
java.util.logging.Level	oracle.adfnmc.java.util.logging.Level
java.util.logging.Logger	oracle.adfnmc.java.util.logging.Logger
java.util.logging.LogManager	oracle.adfnmc.java.util.logging.LogManager
java.util.logging.LogRecord	oracle.adfnmc.java.util.logging.LogRecord
java.util.logging.Messages	oracle.adfnmc.java.util.logging.Messages
java.util.logging.NmcPatternFormatter	oracle.adfnmc.java.util.logging.NmcPatternFormatter
java.util.logging.PatternFormatter	oracle.adfnmc.java.util.logging.PatternFormatter
java.util.logging.SimpleFormatter	oracle.adfnmc.java.util.logging.SimpleFormatter
java.util.logging.StreamHandler	oracle.adfnmc.java.util.logging.StreamHandler
java.util.regex.AbstractCharClass	oracle.adfnmc.java.util.regex.AbstractCharClass
java.util.regex.AbstractLineTerminator	oracle.adfnmc.java.util.regex.AbstractLineTerminator
java.util.regex.AbstractSet	oracle.adfnmc.java.util.regex.AbstractSet
java.util.regex.AheadFSet	oracle.adfnmc.java.util.regex.AheadFSet
java.util.regex.AltGroupQuantifierSet	oracle.adfnmc.java.util.regex.AltGroupQuantifierSet
java.util.regex.AltQuantifierSet	oracle.adfnmc.java.util.regex.AltQuantifierSet
java.util.regex.AtomicFSet	oracle.adfnmc.java.util.regex.AtomicFSet
java.util.regex.AtomicJointSet	oracle.adfnmc.java.util.regex.AtomicJointSet
java.util.regex.BackReferencedSingleSet	oracle.adfnmc.java.util.regex.BackReferencedSingleSet
java.util.regex.BackReferenceSet	oracle.adfnmc.java.util.regex.BackReferenceSet
java.util.regex.BehindFSet	oracle.adfnmc.java.util.regex.BehindFSet
java.util.regex.CanClasses	oracle.adfnmc.java.util.regex.CanClasses

Table 7–3 (Cont.) Additional JavaSE Classes

JavaSE Class	ADF Mobile Client Equivalent
java.util.regex.CharClass	oracle.adfnmc.java.util.regex.CharClass
java.util.regex.CharSet	oracle.adfnmc.java.util.regex.CharSet
java.util.regex.CIBackReferenceSet	oracle.adfnmc.java.util.regex.CIBackReferenceSet
java.util.regex.CICharSet	oracle.adfnmc.java.util.regex.CICharSet
java.util.regex.CIDecomposedCharSet	oracle.adfnmc.java.util.regex.CIDecomposedCharSet
java.util.regex.CISequenceSet	oracle.adfnmc.java.util.regex.CISequenceSet
java.util.regex.CompositeGroupQuantifierSet	oracle.adfnmc.java.util.regex.CompositeGroupQuantifierSet
java.util.regex.CompositeQuantifierSet	oracle.adfnmc.java.util.regex.CompositeQuantifierSet
java.util.regex.CompositeRangeSet	oracle.adfnmc.java.util.regex.CompositeRangeSet
java.util.regex.DecomposedCharSet	oracle.adfnmc.java.util.regex.DecomposedCharSet
java.util.regex.DotAllQuantifierSet	oracle.adfnmc.java.util.regex.DotAllQuantifierSet
java.util.regex.DotAllSet	oracle.adfnmc.java.util.regex.DotAllSet
java.util.regex.DotQuantifierSet	oracle.adfnmc.java.util.regex.DotQuantifierSet
java.util.regex.DotSet	oracle.adfnmc.java.util.regex.DotSet
java.util.regex.EmptySet	oracle.adfnmc.java.util.regex.EmptySet
java.util.regex.EOISet	oracle.adfnmc.java.util.regex.EOISet
java.util.regex.EOLSet	oracle.adfnmc.java.util.regex.EOLSet
java.util.regex.FinalSet	oracle.adfnmc.java.util.regex.FinalSet
java.util.regex.FSet	oracle.adfnmc.java.util.regex.FSet
java.util.regex.GroupQuantifierSet	oracle.adfnmc.java.util.regex.GroupQuantifierSet
java.util.regex.HangulDecomposedCharSet	oracle.adfnmc.java.util.regex.HangulDecomposedCharSet
java.util.regex.HashDecompositions	oracle.adfnmc.java.util.regex.HashDecompositions
java.util.regex.HighSurrogateCharSet	oracle.adfnmc.java.util.regex.HighSurrogateCharSet
java.util.regex.I18n	oracle.adfnmc.java.util.regex.I18n
java.util.regex.IntArrHash	oracle.adfnmc.java.util.regex.IntArrHash
java.util.regex.IntHash	oracle.adfnmc.java.util.regex.IntHash
java.util.regex.JointSet	oracle.adfnmc.java.util.regex.JointSet
java.util.regex.LeafQuantifierSet	oracle.adfnmc.java.util.regex.LeafQuantifierSet
java.util.regex.LeafSet	oracle.adfnmc.java.util.regex.LeafSet
java.util.regex.Lexer	oracle.adfnmc.java.util.regex.Lexer
java.util.regex.LowHighSurrogateRangeSet	oracle.adfnmc.java.util.regex.LowHighSurrogateRangeSet
java.util.regex.LowSurrogateCharSet	oracle.adfnmc.java.util.regex.LowSurrogateCharSet
java.util.regex.Matcher	oracle.adfnmc.java.util.regex.Matcher
java.util.regex.MatchResult	oracle.adfnmc.java.util.regex.MatchResult
java.util.regex.MatchResultImpl	oracle.adfnmc.java.util.regex.MatchResultImpl
java.util.regex.MultiLineEOLSet	oracle.adfnmc.java.util.regex.MultiLineEOLSet
java.util.regex.MultiLineSOLSet	oracle.adfnmc.java.util.regex.MultiLineSOLSet
java.util.regex.NegativeLookAhead	oracle.adfnmc.java.util.regex.NegativeLookAhead

Table 7–3 (Cont.) Additional JavaSE Classes

JavaSE Class	ADF Mobile Client Equivalent
java.util.regex.NegativeLookBehind	oracle.adfnmc.java.util.regex.NegativeLookBehind
java.util.regex.NonCapFSet	oracle.adfnmc.java.util.regex.NonCapFSet
java.util.regex.NonCapJointSet	oracle.adfnmc.java.util.regex.NonCapJointSet
java.util.regex.Pattern	oracle.adfnmc.java.util.regex.Pattern
java.util.regex.PatternSyntaxException	oracle.adfnmc.java.util.regex.PatternSyntaxException
java.util.regex.PosAltGroupQuantifierSet	oracle.adfnmc.java.util.regex.PosAltGroupQuantifierSet
java.util.regex.PosCompositeGroupQuantifierSet	oracle.adfnmc.java.util.regex.PosCompositeGroupQuantifierSet
java.util.regex.PositiveLookAhead	oracle.adfnmc.java.util.regex.PositiveLookAhead
java.util.regex.PositiveLookBehind	oracle.adfnmc.java.util.regex.PositiveLookBehind
java.util.regex.PosPlusGroupQuantifierSet	oracle.adfnmc.java.util.regex.PosPlusGroupQuantifierSet
java.util.regex.PossessiveAltQuantifierSet	oracle.adfnmc.java.util.regex.PossessiveAltQuantifierSet
java.util.regex.PossessiveCompositeQuantifierSet	oracle.adfnmc.java.util.regex.PossessiveCompositeQuantifierSet
java.util.regex.PossessiveGroupQuantifierSet	oracle.adfnmc.java.util.regex.PossessiveGroupQuantifierSet
java.util.regex.PossessiveQuantifierSet	oracle.adfnmc.java.util.regex.PossessiveQuantifierSet
java.util.regex.PreviousMatch	oracle.adfnmc.java.util.regex.PreviousMatch
java.util.regex.Quantifier	oracle.adfnmc.java.util.regex.Quantifier
java.util.regex.QuantifierSet	oracle.adfnmc.java.util.regex.QuantifierSet
java.util.regex.RangeSet	oracle.adfnmc.java.util.regex.RangeSet
java.util.regex.RelAltGroupQuantifierSet	oracle.adfnmc.java.util.regex.RelAltGroupQuantifierSet
java.util.regex.RelCompositeGroupQuantifierSet	oracle.adfnmc.java.util.regex.RelCompositeGroupQuantifierSet
java.util.regex.ReluctantAltQuantifierSet	oracle.adfnmc.java.util.regex.ReluctantAltQuantifierSet
java.util.regex.ReluctantCompositeQuantifierSet	oracle.adfnmc.java.util.regex.ReluctantCompositeQuantifierSet
java.util.regex.ReluctantGroupQuantifierSet	oracle.adfnmc.java.util.regex.ReluctantGroupQuantifierSet
java.util.regex.ReluctantQuantifierSet	oracle.adfnmc.java.util.regex.ReluctantQuantifierSet
java.util.regex.SequenceSet	oracle.adfnmc.java.util.regex.SequenceSet
java.util.regex.SingleDecompositions	oracle.adfnmc.java.util.regex.SingleDecompositions
java.util.regex.SingleSet	oracle.adfnmc.java.util.regex.SingleSet
java.util.regex.SOLSet	oracle.adfnmc.java.util.regex.SOLSet
java.util.regex.SpecialToken	oracle.adfnmc.java.util.regex.SpecialToken
java.util.regex.SupplCharSet	oracle.adfnmc.java.util.regex.SupplCharSet
java.util.regex.SupplRangeSet	oracle.adfnmc.java.util.regex.SupplRangeSet
java.util.regex.UCIBackReferenceSet	oracle.adfnmc.java.util.regex.UCIBackReferenceSet
java.util.regex.UCICharSet	oracle.adfnmc.java.util.regex.UCICharSet

Table 7–3 (Cont.) Additional JavaSE Classes

JavaSE Class	ADF Mobile Client Equivalent
java.util.regex.UCIDecomposedCharSet	oracle.adfnmc.java.util.regex.UCIDecomposedCharSet
java.util.regex.UCIRangeSet	oracle.adfnmc.java.util.regex.UCIRangeSet
java.util.regex.UCISequenceSet	oracle.adfnmc.java.util.regex.UCISequenceSet
java.util.regex.UCISupplCharSet	oracle.adfnmc.java.util.regex.UCISupplCharSet
java.util.regex.UCISupplRangeSet	oracle.adfnmc.java.util.regex.UCISupplRangeSet
java.util.regex.UEOLSet	oracle.adfnmc.java.util.regex.UEOLSet
java.util.regex.UMultiLineEOLSet	oracle.adfnmc.java.util.regex.UMultiLineEOLSet
java.util.regex.UnicodeCategory	oracle.adfnmc.java.util.regex.UnicodeCategory
java.util.regex.UnicodeCategoryScope	oracle.adfnmc.java.util.regex.UnicodeCategoryScope
java.util.regex.UnifiedQuantifierSet	oracle.adfnmc.java.util.regex.UnifiedQuantifierSet
java.util.regex.WordBoundary	oracle.adfnmc.java.util.regex.WordBoundary
javax.sql.DataSource	oracle.adfnmc.java.javax.sql.DataSource

8

Deploying ADF Mobile Client Components

This chapter describes how to deploy the ADF Mobile client runtime and applications to Windows Mobile devices and emulators as well as BlackBerry smartphones and simulators.

This chapter includes the following sections:

8.1 Introduction to Deployment

ADF Mobile client enables you to package and deploy an application to either the Windows Mobile or BlackBerry platforms. For Windows Mobile, ADF Mobile client enables you to create and deploy cabinet (CAB) files. For BlackBerry, ADF Mobile client enables you to package applications as COD files. ADF Mobile deployment is a multi-step process that culminates in application deployment. This process involves:

1. Deploying the ADF Mobile client runtime to the BlackBerry smartphone or simulator or to the Windows Mobile device or emulator. For more information, see Section 8.2, "Deploying the ADF Mobile Client Runtime."
2. Creating Sync Data Publications on the Server. You must deploy the application data publication to the runtime to specify the server-side data that synchronizes with the local database on the client. For more information, see Section 8.3, "Creating Data Sync Publications on the Server."
3. Deploying the actual ADF Mobile client application. You can deploy an application through JDeveloper or you can deploy it directly to a BlackBerry smartphone or simulator or to a Windows Mobile device or emulator. For more information, see Section 8.4, "Working with Application Deployment Profiles."

Deployment is also part of the development process: because ADF Mobile client applications only run after you have deployed them, you must therefore deploy an application before testing and debugging it. Unlike applications deployed to a server, ADF Mobile client applications are typically deployed directly to a mobile device without receiving further configuration. As a result, the deployment process enables you to verify the contents of an application.

8.1.1 Application Deployment Prerequisites

Before you deploy applications, you must obtain and configure the software appropriate to the target deployment platform and the Oracle database, Oracle Database Lite, and SQLite databases as described in Chapter 2, "Setting Up the ADF Mobile Client Environment."

8.2 Deploying the ADF Mobile Client Runtime

To enable ADF Mobile client applications to execute properly on a smartphone, mobile device, simulator or emulator, you must deploy the ADF Mobile client runtime and other runtime components to the appropriate target.

8.2.1 How to Deploy the Runtime Components

The **Deploy to ADF Mobile Client Runtime** options enable you to deploy the ADF client runtime, along with the Mobile Sync (mSync) to a Windows Mobile device or emulator or to a BlackBerry smartphone or simulator. For Windows Mobile, you can also deploy the Java Virtual Machine (Java Micro Edition Connected Device Configuration HotSpot Implementation).

Before you begin:

Configure the environment for Windows Mobile emulators or BlackBerry Smartphone simulators as described in Section 2.6, "Setting Up Development Tools for Windows Mobile Platform" and Section 2.7, "Setting Up Development Tools for BlackBerry Platform." For deployment to a BlackBerry smartphone, you must connect the smartphone to the development computer using a compatible USB cable.

To deploy the ADF Mobile client runtime and mSync to BlackBerry smartphones and simulators:

1. Choose **Tools** and then choose **Deploy ADF Mobile Client Runtime**.
2. Depending on the deployment target, select either **to BlackBerry Smartphone** or **to BlackBerry Simulator**.
3. Select the runtime components. Choose either **ADF Mobile Client Runtime** or **Data Sync** (which deploys mSync), or both.
4. Click **Deploy**.
5. View the entries written to JDeveloper's Deployment-Log. Example 8–1 illustrates a deployment log to a BlackBerry Smartphone simulator:

Example 8–1 Deployment-Log Entries for Runtime Component to a BlackBerry Smartphone Simulator

```
[03:41:22 PM] Copying files to simulator located at: C:\Program Files\Research In
Motion\BlackBerry Smartphone Simulators 4.7.1\4.7.1.65 (9630)
[03:41:23 PM] Files copied successfully. You must restart the simulator before
running the Mobile Client application.
```

6. Restart the BlackBerry smartphone simulator (for deployment to a simulator).

To deploy the ADF Mobile client runtime, mSync, and JVM to Windows Mobile devices and emulators:

1. Choose **Tools** and then choose **Deploy ADF Mobile Client Runtime**.
2. Choose **to Windows Mobile**.

3. Select a runtime component, such as **ADF Mobile Client Runtime**, **Data Sync**, or **Java Virtual Machine**.

> **Note:** For Windows Mobile, you must deploy runtime components one after another; you must wait for deployment of one component to complete before you can select the next one.

4. Click **Deploy**.
5. Monitor the component's deployment progress on the Windows Mobile device or emulator.
6. If needed, select another runtime component and then click **Deploy**.
7. Click **Close**.
8. View the entries written to JDeveloper's Deployment-Log. Example 8–2 illustrates deployment of mSync (resulting from selecting the **Data Sync** option):

Example 8–2 Deployment of Data Sync (mSync) to a Windows Mobile Emulator

```
[01:00:25 PM] Copying file:
C:\JDev\jdeveloper\jdev\extensions\oracle.adfnmc.core\WindowsMobile\deploy\wm6\sql
ite.us.ppc60.armv4i.CAB
[01:02:18 PM] Data Sync file copied. Monitor installation progress from the device
or emulator...
```

8.3 Creating Data Sync Publications on the Server

When you create data sync publications, you are essentially telling Oracle Database Lite Mobile Server (Mobile Server) which data should be synchronized to a BlackBerry smartphone or Windows Mobile device. Data sync publications are not required for applications that use a custom local database. For more information, see Section 8.5, "Specifying the Client Database Location for an Application" and Chapter 11, "Working Directly with the Database." In most situations, however, ADF Mobile client applications must synchronize data with a back-end database. Data sync publications provide ADF Mobile client applications with the appropriate data from the server. Although you can successfully deploy an ADF Mobile client application without first creating the data sync publications, doing so prevents the application from finding the required data at runtime.

8.3.1 How to Create Data Sync Publications

The Sync Publication page (Figure 8–4) enables you to create the connection to the Mobile Server.

Before you begin:

You must install Oracle Database Lite Mobile Server. (Mobile Server). For more information, see Section 2.5, "Setting Up Oracle Database Lite."

You must create the following two types of connections:

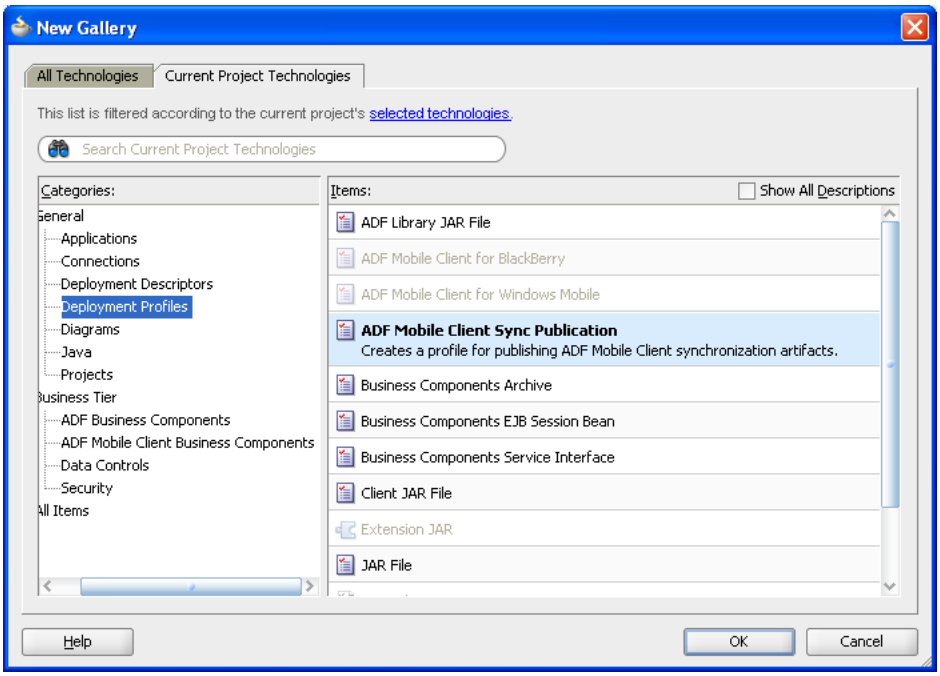
- A database connection for the Oracle Lite Mobile Server Repository schema. If you installed Mobile Server with the default settings, this database schema belongs to the `MOBILEADMIN` user. See also "Installation of Mobile Server" in *Oracle Database Lite Getting Started Guide*.

- A connection to a local database that synchronizes with the back-end database using Mobile Server. You create this connection using the Create Database Connection dialog illustrated in Figure 8–3.

To create data sync publications

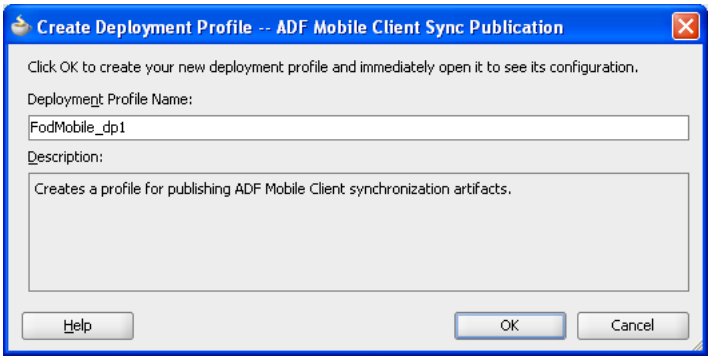
1. Right-click the Model project in the Application Navigator, choose **New** and then **Deployment Profiles**.
2. Choose **ADF Mobile Client Sync Publication** as shown in Figure 8–1 and then click **OK**.

Figure 8–1 Selecting the Mobile Client Sync Publication Profile



3. Enter a name for the data sync publication (or accept the default name, as shown in Figure 8–2) and then click **OK**.

Figure 8–2 Naming the Data Sync Publication



4. In the Sync Publication page, select the connection used by the client database that synchronizes with the back-end database through Mobile Server. If no such connection exists:

a. Click **Add**.

b. In the Create Database Connection dialog, enter the name for the connection for the synchronized application, such as MySynchronizedConnection shown in Figure 8–3.

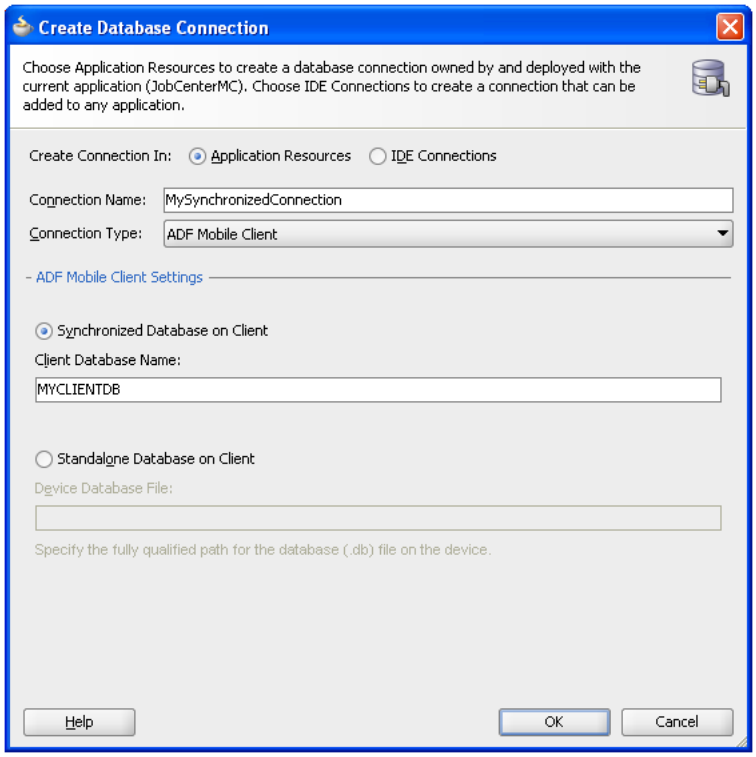
c. Select **ADF Mobile Client** as the connection type.

d. Select **Synchronized Database on Client**.

e. Enter the name of the local (client) database that synchronizes with a back-end database through Mobile Server.

f. Click **OK**. For more information on the Create Database Connection dialog, click **Help** to see the JDeveloper online help

Figure 8–3 Creating a Synchronized Connection



5. Enter the following:

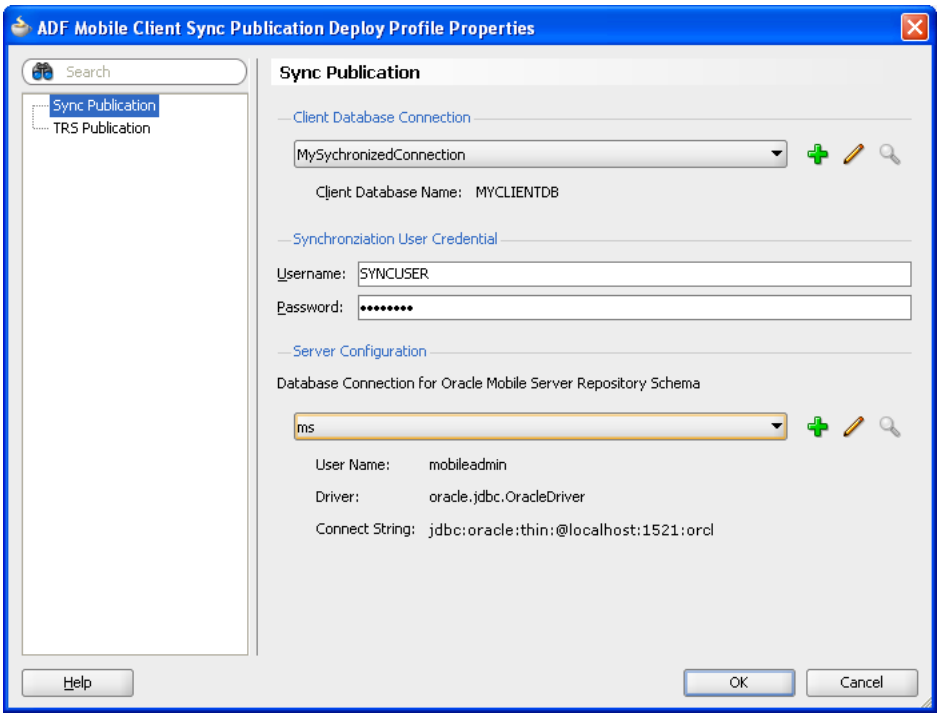
- The **User Name** and **Password** parameters specify the account associated with this sync publication. This is the same user name and password combination that users enter on the device when prompted for credentials for data synchronization. If this account does not exist on Mobile Server, it will be created automatically when you publish synchronization artifacts.
- The **Database Connection for Oracle Mobile Server Repository Schema** parameters represent the Oracle database schema where the Mobile Server artifacts are stored.

Note: This is not the same schema for the base application as described in Section 5.2, "Extending an ADF Application to Mobile Client."

Click **Add** to create a connection if one does not already exist or click **Edit** to update the connection information.

6. Click **OK**.

Figure 8–4 The Sync Publication Page



8.3.2 What Happens When You Create a Database Connection

After you create an ADF Mobile client connection using the Create Database Connection dialog, JDeveloper stores the connection information in the `connections.xml` file, a file that is created when you create a new connection using JDeveloper. This file is packaged with other meta-data files and deployed to the actual device or simulator. The ADF Mobile client runtime reads the `connection.xml` file as needed. `connections.xml` is located in the Application Navigator's Application Resources panel, under either the Descriptors or ADF META-INF nodes.

8.4 Working with Application Deployment Profiles

Preparing ADF Mobile client applications for deployment is primarily comprised of creating platform-specific deployment profiles. A deployment profile defines how an application is packaged into the archive that will be deployed to either a BlackBerry smartphone and simulator or Windows Mobile device and emulator. The deployment profile:

- Specifies the format and contents of the archive. For Windows Mobile, the archive format is a cabinet (CAB) file. For BlackBerry, the format is a COD file.
- 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.

The ADF Mobile client extension adds ADF Mobile client-specific pages for both CAB and COD deployment to the standard ADF deployment profiles that include:

- JAR Options
- File Groups
- Library Dependencies
- Profile Dependencies

For more information on these standard ADF deployment profile pages, click **Help** to see the JDeveloper online help.

Table 8–1 lists the ADF Mobile client-specific pages in the Deployment Profile dialog.

Table 8–1 ADF Mobile Client-Specific Deployment Profile Pages

Page	Function
Windows Mobile Options	Enables you to modify the settings for an application to be deployed on a Windows Mobile device and emulator.
BlackBerry Options	Enables you to modify the settings for an application deployed to a BlackBerry smartphone and simulator.
Application Icon	Enables you to assign custom icons to an application by adding the appropriate graphics file.
Client Database	Enables you to specify if the database is a standalone on the device, or synchronized.
Localization	Enables you to select localized resources for user-facing strings.

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 8–1.

8.4.1 How to Create a Deployment Profile for BlackBerry Applications

ADF Mobile client applications are deployed to BlackBerry smartphones as COD files. ADF Mobile client enables you to create a deployment profile.

Before you begin:

Set up the BlackBerry environment as described in Chapter 2, "Setting Up the ADF Mobile Client Environment." Deployment requires that you complete this configuration, which includes:

- To enable users to add an application using the ALX file, download BlackBerry Desktop Manager (for the development environment) and BlackBerry Enterprise Server (if you plan on posting the COD to a server in a production environment).
- Download BlackBerry Java Development Environment (JDE 5.0) and a BlackBerry simulator. (The simulator is optional, as you typically use the simulator included in the JDE.) If the application synchronizes with the back-end database using Oracle Database Lite Mobile Server, download the Oracle Database Lite 10*g* Mobile Development Kit (MDK). Using the ADF Mobile client preferences dialog (accessed by clicking **Tools** then **Preferences** then **ADF Mobile Client**), you set the locations for the JDE and the simulator. When you deploy an application to a

BlackBerry smartphone simulator, JDeveloper places the resulting COD file in the simulator's directory. If the application uses Mobile Server, entering the location of the MDK directory adds `osync_rim.jar` and `mSync.jar` which enable on-device synchronization. You do not need to specify the MDK location for standalone applications or for applications using web servers for server communications.

- If you deploy to an actual device (that is, a BlackBerry smartphone), you must connect the smartphone to the development computer using a compatible USB cable.

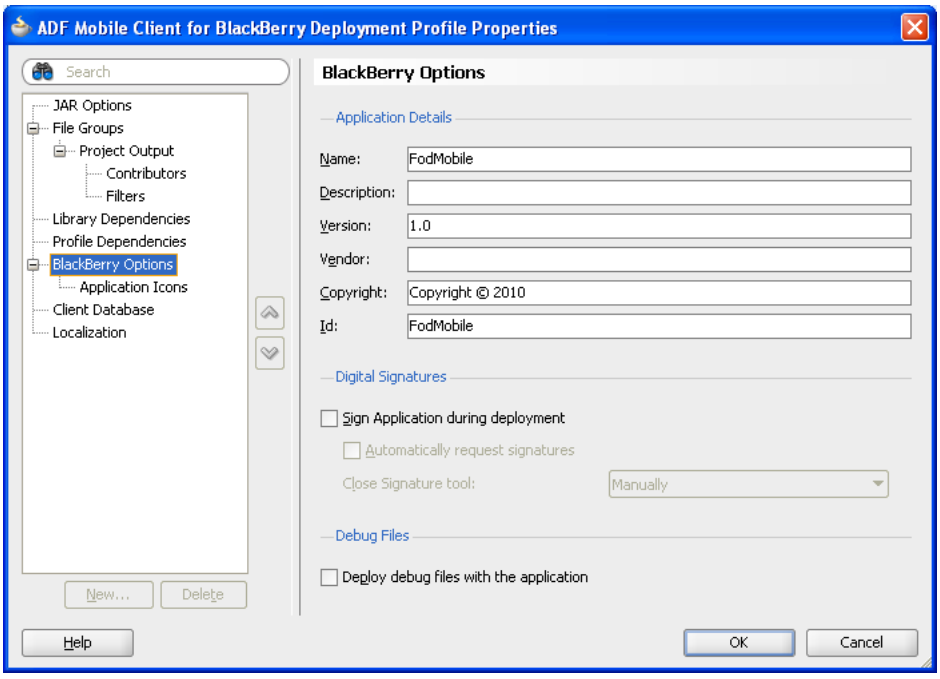
To Create a BlackBerry Deployment Profile

1. Select **Application** then **Application Properties**.
2. Click **New** in the Deployment Page and then choose **ADF Mobile Client for BlackBerry** as the Archive Type.
3. Enter a name for the deployment profile, or accept the default name, and then click **OK**.

8.4.1.1 Setting and Modifying Application Details

The Options page enables you to edit the properties of a deployment file and to set the behavior of the BlackBerry Signature tool.

Figure 8–5 The BlackBerry Options Page



To edit the deployment file options:

1. Choose **BlackBerry Options**.
2. Accept the default values, or define the following options:
   - **Name**—The name of the BlackBerry application. This value is populated by default and matches the name of the application workspace.
   - **Description**—A description of the application.

- **Version**—The version number of the application.

  Note: You cannot overwrite the same application installed on a BlackBerry smartphone that has the same or higher version number.

- **Vendor**—The name of the application vendor.
- **Copyright**—The copyright year.
- **ID**—An ID for the application.

3. If needed, set the digital signature options as described in Section 8.4.1.2, "Setting the BlackBerry Digital Signature Tool Options" and then click **OK**.

8.4.1.2 Setting the BlackBerry Digital Signature Tool Options

For security purposes, RIM requires that any application that uses controlled APIs be signed, or it will fail to run on a smartphone.

Note: Applications can run on the BlackBerry simulator without signing.

Before you begin:

Before you can use the BlackBerry Signature Tool, you must register for the RIM Signing Authority Account. For more information, see *Blackberry - Java Code Signing Keys* at `http://na.blackberry.com/eng/`.

To set the digital signatures options:

Select from among the following and then click **OK**:

- **Sign Application During Deployment**—Select this option to sign the application. If you select this option, the BlackBerry signature tool launches before the application is deployed to the BlackBerry smartphone or simulator.
- **Automatically request signatures**—Select to enable the BlackBerry Signature Tool to automatically send code-signing requests to the BlackBerry Signing Authority Tool.
- **Close Signature tool**—Select the option type to close the BlackBerry Signature Tool. The Signing Tool closes after it returns request results and applies any granted permissions to the application package. Application deployment then begins.
  - Select **Manually** to manually close the signing tool after the singing results have been received.
  - Select **After Requesting Signatures** to close the BlackBerry Signature Tool after the signing results have been received, irrespective of success or failure.
  - Select **After Requesting Signatures, if no errors** to close the BlackBerry Signature Tool after the signing tool results have been received and the signing completed successfully without any errors.
- **Deploy debug files with the application**—Select this option to deploy debugging files along with the application. The application debug files are deployed to the BlackBerry simulator folder both for the simulator and for the BlackBerry smartphone itself. These files enable debugging on both the BlackBerry smartphone and simulator.

8.4.1.3 Adding a Customized Icon to a BlackBerry Application

BlackBerry applications require two image files, one for a standard icon and one for showing that the application has focus. RIM recommends using images measuring 80x80 pixels for both of these images. See "Default Themes and Dimensions for Screens and Application Icons on BlackBerry Devices" in *BlackBerry Smartphones UI Guidelines Version: 2.4.* available at:

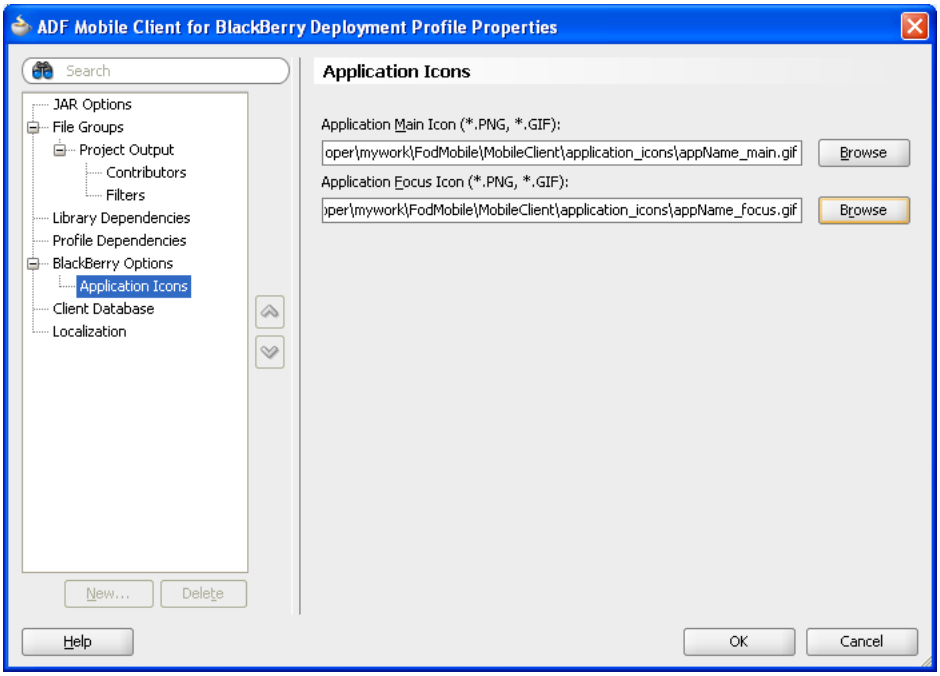
`http://docs.blackberry.com/`

See also "Icons and Indicators" in *BlackBerry Smartphones UI Guidelines Version: 2.4* for design guidelines.

Note: Oracle recommends a size limit under 16K.

8.4.1.3.1 How to Add Custom Icons to a BlackBerry Application The Application Icons page of the ADF Mobile Client for BlackBerry Deployment Profiles Properties dialog enables you to add custom icons by adding the PNG- or GIF-formatted images for the main and focus icons. If you do not add a custom image file, then the default Oracle icon is used.

Figure 8–6 Adding Custom Icons to a BlackBerry Application



Before you begin:

Obtain the images in the file format, dimensions, pixels, and components appropriate to the BlackBerry theme as described icons in *BlackBerry Smartphones UI Guidelines Version: 2.4*, available at:

`http://docs.blackberry.com/`

You must also add the image files to the view controller (MobileClient) project. For example, copy the file containing the application icon PNG and GIF files to:

```
C:\JDeveloper\mywork\<application_name>\MobileClient\<icon_image
file>
```

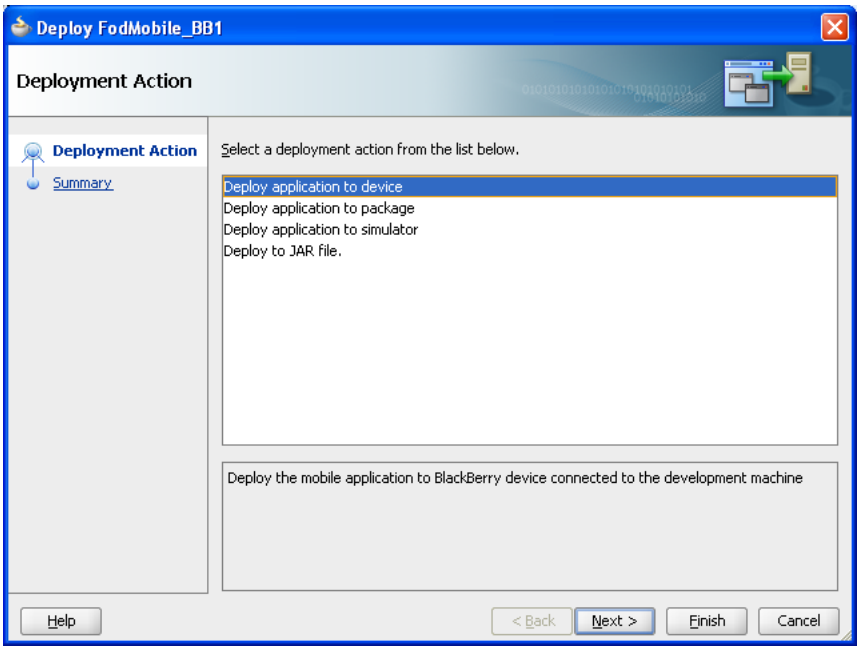
To add custom main and focus icons to a BlackBerry application:

1. Click **Application Icons**.
2. Use the **Browse** function to select the main and focus icon image files from the project file.
3. Click **OK**.

8.4.1.4 Deploying BlackBerry Applications

The Deployment Action page, shown in Figure 8–7, enables you to deploy an application to a BlackBerry smartphone or simulator or deploy to a JAR file. It also enables you to deploy the application to a package that end users can manually add to smartphones or simulators by copying it directly or by using the BlackBerry's ALX application loader file. When you package an application, JDeveloper creates an ALX file along with other application objects.

Figure 8–7 The Deployment Action Page with Application-Level Deployment Options



Before you begin:

Set up the BlackBerry environment as described in Section 2.7, "Setting Up Development Tools for BlackBerry Platform."

To deploy an ADF Mobile application to a Blackberry smartphone or simulator:

1. From the main menu, select **Application** then **Deploy**, and then choose the BlackBerry deployment profile you created earlier. For more information on creating a deployment profile, see Section 8.4.1, "How to Create a Deployment Profile for BlackBerry Applications."
2. Select **Deploy application to device** or **Deploy application to a simulator**. Click **Next**.
3. Review the Summary page, which lists the name of output JAR, main class and compression level. Click **Finish**.

4. View the Deployment-Log in JDeveloper. JDeveloper writes lines similar to the following for a successful deployment of a BlackBerry application to a simulator.

```
[14:15:55 PM] Deploying Mobile Client application to BlackBerry simulator
located at C:\Program Files\Research In Motion\BlackBerry JDE 5.0.0\simulator.
[14:15:55 PM] Mobile Client application successfully deployed to BlackBerry
simulator. Please start or restart the simulator before running the Mobile
Client application.
[14:15:55 PM] Elapsed time for deployment: 5 second
[14:15:55 PM]  --- Deployment finished. ---
```

To deploy an ADF Mobile application directly to BlackBerry smartphone or simulator:

You can deploy an application directly by copying the application files to the BlackBerry smartphone or simulator. Depending on what files are in use at the time, installing the application may trigger a restart on the smartphone. On the simulator, the application files that you deploy are available when you next start the simulator.

Selecting the **Deploy Application to Package** option results in the creation of COD and ALX files.

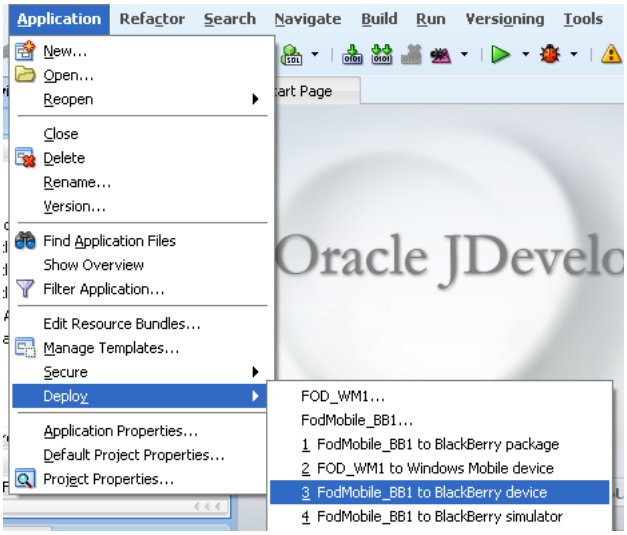
To deploy an ADF Mobile application directly to an ALX file:

1. Choose **Deploy Application to Package**.
2. If you are deploying to the actual smartphone, connect the smartphone to the computer using a compatible USB cable.
3. In the Main Menu of BlackBerry Desktop Manager, select **Application Loader**.
4. Select **Add/Remove Applications**.
5. Click **Browse** to locate ALX file. It is at the same location where JDeveloper created the package.
6. Select the ALX file.
7. Click **Next** in the Application Loader screen.
8. Click **Finish**.

8.4.1.4.1 Selecting 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. Figure 8–8, for example, shows shortcuts to recently used deployment profiles called *FodMobile_BB1 to BlackBerry device* and *FodMobile_BB1 to BlackBerry simulator* that were created as a result of choosing the **Deploy application to device** and **Deploy application to simulator** options. Choosing these shortcuts redeploys the application.

Figure 8–8 Deployment Shortcuts



8.4.2 How to Create a Deployment Profile for Windows Mobile

ADF Mobile client applications are deployed to Windows Mobile devices and emulators as CAB (cabinet) files.

Before you begin:

Set up the Windows Mobile environment as described in Chapter 2, "Setting Up the ADF Mobile Client Environment."

To create a deployment profile:

1. Open the deployment properties for the application by clicking **Application** then **Application Properties** and then **Deployment**.
2. Click **New** in the Deployment page.
3. In the Create Deployment Profile Dialog, choose **ADF Mobile Client for Windows Mobile**.
4. Enter a name for the deployment profile, click **OK**.
5. Click **OK**.

8.4.2.1 Setting the JAR File Options

The CAB file includes an application JAR file, a launcher executable file, and an options file. These components bear the same name as the deployment profile. For example, the CAB for a profile called `MyApplication` would include `MyApplication.exe` (the launcher), `MyApplication.options`, and `MyApplication.jar`.

8.4.2.1.1 About the Launcher Executable File The launcher executable (the launcher) simplifies the configuration of ADF Mobile client applications by dynamically building a command line that starts the Oracle Java Micro Edition Connected Device Configuration HotSpot Implementation (a JVM) and launches the application JAR, thus eliminating the need to specify this information at buildtime. The launcher contains the application icon in Windows Mobile applications. When you specify an application icon using the deployment profile's Application Icon page, it is embedded in the application's copy of the launcher executable before it is packaged.

The launcher has its own set of command line arguments (listed in Table 8–2) and can use the classpath-related Java command line arguments to rewrite the classpath. The launcher passes all other command line arguments (that is, those not specific to a Java class path or to the launcher itself) directly to Java unaltered. These arguments can be entered on the command line, or specified in the options file.

> **Note:** The typical ADF Mobile client application does not require these arguments.

Table 8–2 Command Line Arguments

Argument	Description
`-options <filename>`	Instructs the launcher file to read a file containing command line options. This argument cannot be placed in the options file itself.
`-adfmc <directory>`	Specifies the directory where ADF Mobile client is installed. Use this argument to override the value configured by the registry.
`-java <directory>`	Specifies the directory where Java is installed. Use this argument to override the value configured by the registry.
`-olite <directory>`	Specifies the directory where Oracle Database Lite is installed. Use this argument to override the default directory.
`-stdout <filename>`	Specifies the file that receives the standard output stream.
`-stderr <filename>`	Specifies the file that receives the standard error output stream.
`-wait`	Use this argument to instruct the launcher to wait until the Java process has exited before it itself exists.
`-silent`	Use this argument to instruct the launcher to not use a modal message box to report errors. This argument is useful for running applications in a scripted environment.

8.4.2.1.2 About the Options File The options file contains the command line arguments used at application startup. The launcher automatically finds this file. Example 8–3 illustrates a typical options file called `MyApplication.options`. This example shows that a full path is not required for the JAR file (`MyApplication.jar`) because it resides in the same directory as both the launcher and options files. The launcher expands the path and adds it to the classpath when invoking the Java executable.

> **Tip:** Because the options file is regenerated each time you deploy an application, enter command line arguments in the **Command Line Options** or **Additional Java Options** fields of the Window Mobile deployment profile's Options page instead of the options file itself.

Example 8–3 The Options File

```
-classpath MyApplication.jar
oracle.adfmc.Main
```

Example 8–4 shows an options file that uses JDWP (Java Debugging Wireline Protocol) interface to debug an application running on a device.

> **Note:** You can also set these arguments using the **Additional Java Options** field as described in Section 8.4.2.1.4, "How to Set the JAR Options."

Example 8–4 Debugging Running Applications with JDWP

```
-classpath MyApplication.jar
-Xdebug -agentlib:jdwp=transport=dt_socket, server=y, address=4041
oracle.adfmnc.Main
```

In the options file, all options before the startup class name are issued to either Java or the launcher itself. To pass arguments to the application instead, supply them after the startup class, as illustrated by `"argument one"` in Example 8–5.

Example 8–5 Setting Application-Specific Arguments

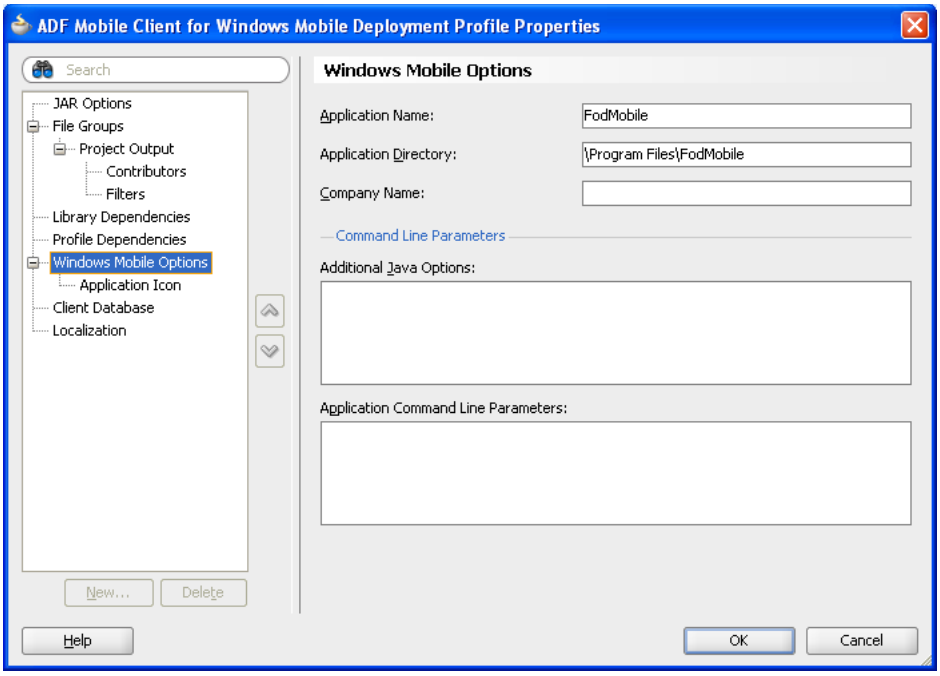
```
-classpath MyApplication.jar
oracle.adfmc.Main
"argument one" arg2 arg3
```

These options are set using the **Application Command Line Parameters** field in the Windows Mobile Options page. For more information, see Section 8.4.2.1.4, "How to Set the JAR Options."

8.4.2.1.3 About Debugging A Windows Mobile Application ADF Mobile client enables you to run an application in debug mode on both Windows Mobile devices or emulators. When you select **Generate Debug Launch Shortcut** in the Windows Mobile Options page. For more information, see Section 10.3.3, "What Happens When You Choose to Generate the Debug Launcher."

8.4.2.1.4 How to Set the JAR Options The Windows Mobile Options page, shown in Figure 8–9, enables you to set the information for the information for the application JAR as well as application start-up or JVM debugging commands.

Figure 8–9 Windows Mobile Options



To set the JAR file options:

1. Choose **Windows Mobile Options**.

2. Accept the default values, or define the following options for the JAR file:
   - **Application Name**—The name of the application on the device. When the CAB installer runs, this value clarifies which application, if any, that you are overriding.
   - **Application Directory**—The location for the directory on the Windows Mobile device or emulator.
   - **Company Name**—The name of the company that created the ADF Mobile client application.
   - **Additional Java Options**—Typically, you use this field for the Java Micro Edition Connected Device Configuration HotSpot Implementation debugging options. These options include `-Xdebug`, the command for running JVM in debugger mode, and `-Xrunjdwp`, which loads the Java Debugger Wireline Protocol (JDWP) and its suboptions. Enter `-Xrunjdwp` and its subotptions using the following format:

     `-Xrunjdwp:transport=dt_socket,server=y,address=<port>`.

     Using the `-classpath` option, you can also use this field update application classpaths with additional JARs.

     For more information about the debugging options and Java SE-command line options, refer to "Application Debugging Command-Line Options" and "Options" in *CDC Runtime Guide*, which is included in the ADF Mobile client extension ZIP at `adfmc_bundle.zip\jdev\extensions\oracle.adfnmc.core\doc` and also available from the Oracle Technology Network (`http://www.oracle.com/technetwork/index.html`)
   - **Application Commandline Parameters**—Enter the application-specific command line arguments for such functions as setting the path to the database directory or writing to the log file.
   - **Generate Debug Launcher**—Generates the executable and a corresponding options file that enable you to run the application in debug mode. For more information see Section 10.3.3, "What Happens When You Choose to Generate the Debug Launcher."
3. Click **OK**.

8.4.2.2 Adding Custom Icons to a Windows Mobile Application

Windows Mobile devices display the same icon image in different sizes depending on context and device resolution. For example, Windows Mobile may present a small (16x16 pixels) version as a toolbar icon and a larger (32x32 pixels) version of the icon elsewhere. To ensure that icons display correctly on all Windows Mobile devices, including those using high DPI, the ICO file should contain various image files in the following sizes:

- 16x16 pixels
- 22x22 pixels
- 32x32 pixels
- 44x44 pixels
- 64x64 pixels

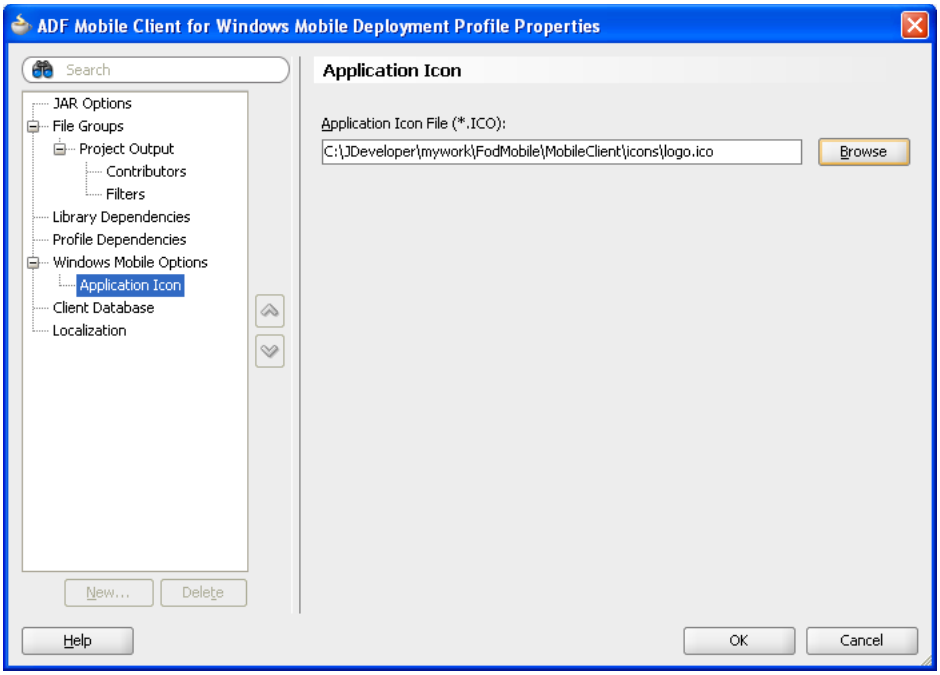
For more information, see the MSDN Library entries for both *Icons* and *Compatible Icons* at

http://msdn.microsoft.com

8.4.2.3 How to Add Custom Icons to a Windows Mobile Application

The Application Icons page, shown in Figure 8–10, of the ADF Mobile Client for Windows Mobile Deployment Profiles Properties dialog enables you to add an ICO file. If you do not add a custom image file, then the default Oracle icon is used.

Figure 8–10 Adding Custom Icons to a Windows Mobile Application



Before you begin:

Create, or obtain, an ICO file for Windows Mobile devices. The ICO file must contain an icon resource that includes the image files appropriate to the Windows Mobile device.

You must also add the image files to the view controller (MobileClient) project. For example, copy the file containing the ICO files to:

`C:\JDeveloper\mywork\<application_name>\MobileClient\<icon_image file>`

To add a custom icon to a Windows Mobile application:

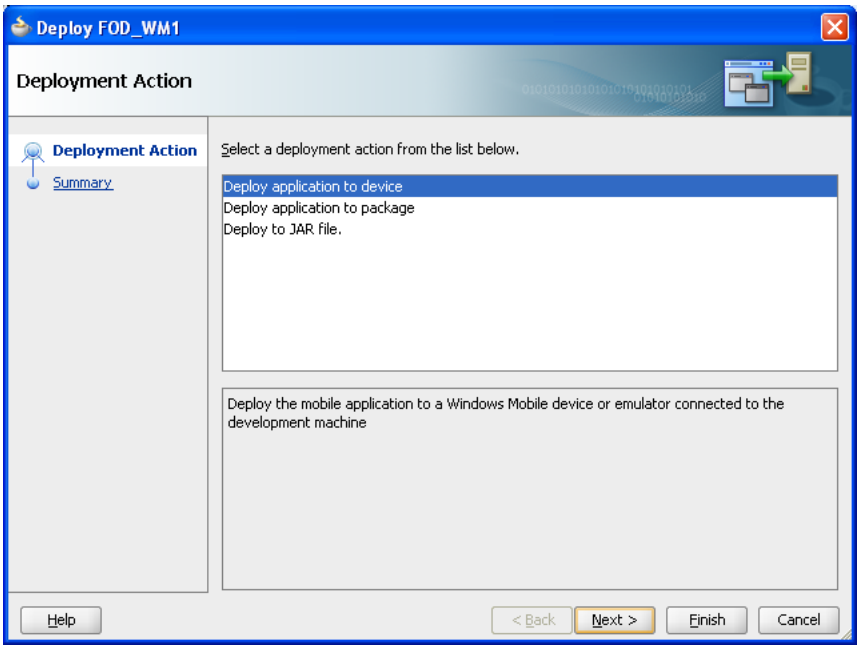
1. Click **Application Icons**.
2. Use the **Browse** function to select the ICO files from the project file.
3. Click **OK**.

> **Note:** If you deploy an application using the default Oracle icon and then subsequently re-deploy the application with a custom icon, you must restart the Windows Mobile device or emulator for the custom icon to appear.

8.4.2.4 Deploying a Windows Mobile Application

The Deployment Action page enables you to deploy an application directly to a Windows Mobile device or to create a CAB file by selecting the **Deploy application to package option**.

Figure 8–11 Windows Mobile Deployment Actions



Before you begin:

Connect a Windows Mobile device to your computer, or start an emulator. Also ensure that ActiveSync or Windows Mobile Device Center has established a link with the device or emulator. For more information, see Section 2.6, "Setting Up Development Tools for Windows Mobile Platform."

To deploy an application to a Windows Mobile device or emulator:

1. Click **Application** then **Deploy** and then choose the Windows Mobile deployment profile.
2. Select **Deploy Application to Device**. Click **Next**.
3. Review the Summary page, which lists the name of output JAR, the main class, and the compression level. Click **Finish**.
4. View Deployment-Log in JDeveloper. JDeveloper writes lines similar to the following for a successful deployment of a Windows Mobile application.

```
[10:50:12 AM] Windows Mobile package successully deployed.
[10:50:12 AM] If installing to a physical device, please refer to device to
complete installation.
[10:50:12 AM] Elapsed time for deployment: 1 second
[10:50:12 AM]  --- Deployment finished. ---
```

5. Complete the installation using the Windows Mobile device or emulator.

To deploy an application directly to the Windows Mobile device or emulator

1. Click **Application** then **Deploy**.

2. Select the Window Mobile profile.
3. Select **Deploy to Application Package** and then click **Next**.
4. Click **Finish**.
5. Copy the CAB file to the Mobile device or simulator.
6. Click the CAB file.
7. Follow the prompts to complete the installation.

8.5 Specifying the Client Database Location for an Application

Usually, you do not have to specify the location of the database if the application uses Oracle Database Lite Mobile Server (Mobile Server) to synchronize the mobile database with a back-end enterprise database. In these cases, ADF Mobile client handles the database location details automatically at runtime. However, you may need greater control over the database location for certain usage scenarios, including:

- An application that uses a database not managed by Mobile Server.
- An application that does not use a database.
- To simplify the early stages of application development by removing the complexity of data synchronization.

8.5.1 How to Specify the Client Database Location

You can override the default behavior by specifying the client database location in the Deployment Profile Properties dialog. The fields that appear on this page depend on the type of connection selected for the application. Figure 8–12, for example, shows properties for a synchronized connection used for an application that synchronizes with a back-end server through Mobile Server.

Figure 8–12 The Client Database Page for a Synchronized Connection

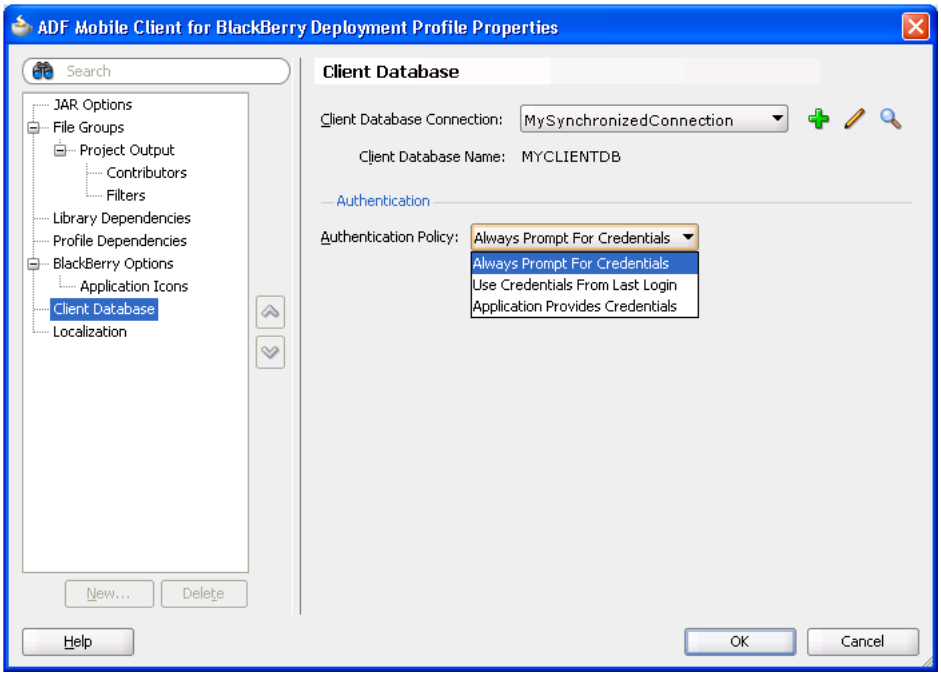
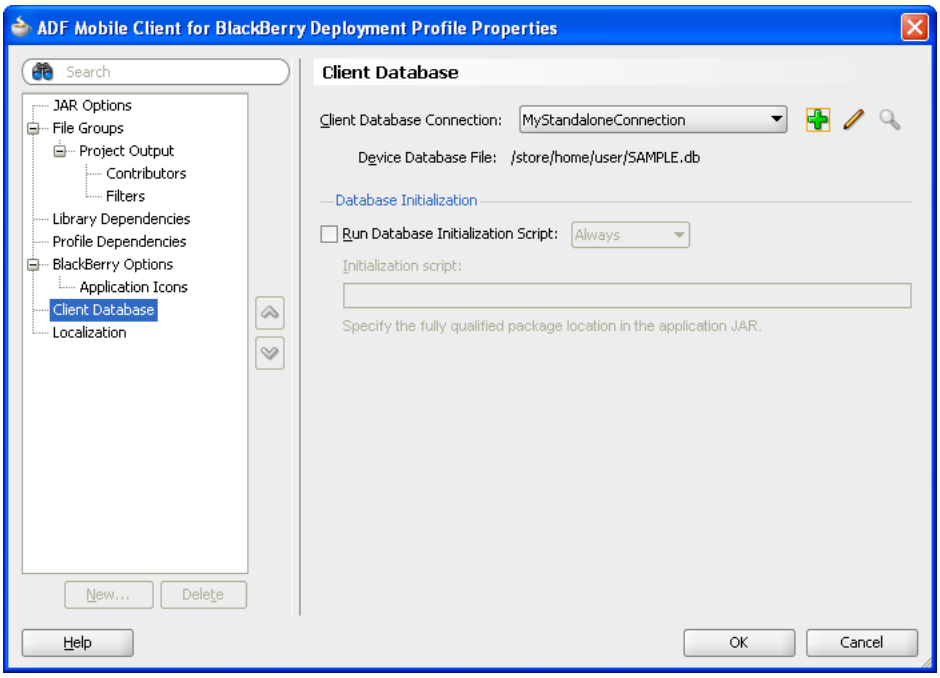


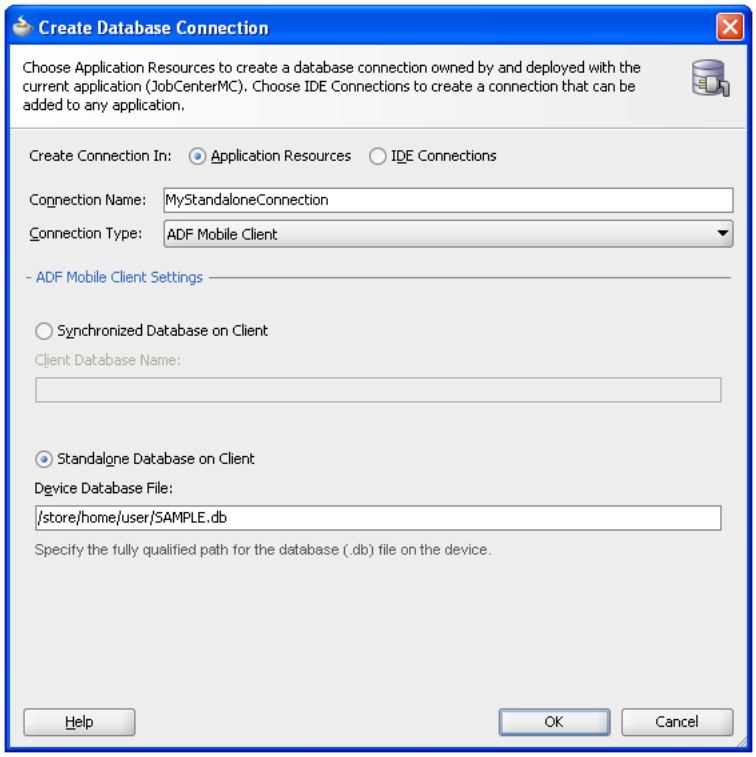
Figure 8–13 shows the Client Database page for applications that use a standalone connection and therefore do not synchronize using Mobile Server.

Figure 8–13 Client Database Page for a Standalone Connection



You can use the Client Database page to select the type of connection used by the application or create or edit the connection using the Create Database Connection Dialog, shown in Figure 8–14. See Figure 8–3 for an illustration of using the Create Database Connection page to create a synchronized connection, the default connection type.

Figure 8–14 Creating a Standalone Connection



Before you begin:

For applications that use a custom (local) database and use a SQL script to initialize the database, you may add initialization-related parameters and also provide the SQL script itself. The SQL script is described in Chapter 11, "Working Directly with the Database."

If you select **ADF Mobile Client** in the Create Database Connection page as shown in Figure 8–3, then you enable the ADF Mobile client framework to use Mobile Server-enabled data synchronization to populate the device's local database when you start an application.

To specify the client database location for synchronizing applications:

1. Accept the default values that appear in the Client Database page, or click **Edit** (or **Add**) to define the following properties in the Create Database Connection dialog:
   - **Connection Name**—Enter a name for the connection, such as one denoting synchronization illustrated by MySynchronizedConnection in Figure 8–3.
   - **Connection Type**—Select **ADF Mobile Client**.
   - **Synchronized Database on Client**—Select this option if the application uses a client (local) database that synchronizes with a back-end database using Mobile Server.
2. In the Client Database page, select one of the following options for retrieving the credentials:
   - **Always Prompt for Credentials**—Select this option to prompt the user for synchronization credentials each time before initiating synchronization. Although this mode is less convenient for end users, use it if multiple users share a single device and each user requires his or her own subset of data.

- **Use Credentials from Last Login**—Select this option to direct the ADF Mobile client framework to use the last-known parameters to automatically initiate synchronization and only prompt the user if synchronization has never been run before. Use this mode in scenarios where each end user has a separate device.
- **Application Provides Credentials**—Select this option so that the user is never prompted for synchronization parameters. The application developer provides these credentials in program code instead.

3. Click **OK**.

To specify the client database location for standalone applications:

1. Accept the default values that appear in the Client Database page, or click **Edit** (or **Add**) to define the following properties in the Create Database Connection dialog:
2. **Connection Name**—Enter a name for the connection, such as one denoting the use of a standalone database illustrated by MyStandaloneConnection in Figure 8–14.
3. **Connection Type**—Select **ADF Mobile Client**.
4. **Standalone Database on Client**—Select this option if the application uses a custom database that is not synchronized using Mobile Server.

   In the **Device Database File** field, enter the fully qualified path to the database on the device's file system. For BlackBerry, the leading character for this path must be a forward slash (`/`) character. Table 8–3 lists example path specifications for different usage scenarios:

Table 8–3 Fully Qualified Paths for BlackBerry

Usage Scenario	Path Format in Device Database File Field
BlackBerry with a database on an internal file system	`/store/home/user/SAMPLE.db`
BlackBerry with a database on an external SD card	`/SDCard/SAMPLE.db`

Note: The name of the removable storage card on Windows Mobile devices is not always *Storage Card*. On some device models, it may be *SD Card*, or it may be translated to another language (*SD-Karte*). Check your target device for the actual name.

Note: In general, SQLite databases on BlackBerry smartphones can only be created on an SD card. While some BlackBerry smartphones permit databases on internal flash memory, you should always specify a database that resides on an SD card to ensure maximum compatibility.

   For Windows Mobile, enter this fully qualified path using a back slash (`\`) as the leading character for the path. Table 8–4 lists example path specifications for different usage scenarios:

Table 8–4 Fully Qualified Paths for Windows Mobile

Usage Scenario	Path Format in Device Database Field
Windows Mobile device with a database on an internal file system	`\SAMPLE.db`
Windows Mobile device with a database on an external storage card	`\Storage Card\SAMPLE.db`

Note: In general, you can only create SQL databases in flash memory on Windows Mobile emulators, not on the storage card. Windows Mobile devices, however, do not have this limitation.

5. Click **OK**.
6. In the Client Database page, select **Run Database Initialization Script** for applications that do not synchronize with Mobile Server but still require a database. This option enables these applications to use a database created by a simple SQL initialization script that uses a subset of SQL syntax. For more information, see Section 11.2, "Enabling Applications to Use SQL Initialization Scripts."
   - Choose **Always** if the application should run the specified SQL initialization script every time it starts.
   - Choose **if no database** if the application should only run the SQL script when the required database does not exist. This is useful in cases where you want the SQL script to initialize the database only when the application starts for the first time.

   The **Initialization script** field specifies the location of the SQL script. Because this script is embedded as a resource in the application's JAR file during deployment, specify a path relative to the root of this JAR file. For example, if you added a file called `SqlScript.sql` to the `res` subfolder of the application, then enter `/res/SqlScript.sql`.
7. Click **OK**.

8.5.2 What Happens When You Specify a Client Database

When you deploy an ADF Mobile client application, the settings you entered are written to the `adf-config.xml file` as a series of key-value pairs. When you run the application, it attempts to connect to the database in one of the following ways:

- Synchronized—The default setting. The **Client Database Name** field specifies a database that is managed by Mobile Server. At runtime, the ADF Mobile client framework receives the location of this database from the Oracle Database Lite library. The framework invokes data synchronization to populate the database if necessary. For more information, see Section 8.5.3, "What Happens When Oracle Database Lite Mobile Server Manages an Application's Database" and Section 8.5.4, "How the ADF Mobile Client Framework Retrieves Mobile Server Credentials at Application Startup."
- Standalone, with **Device Database File** parameter specified—The local database will not be synchronized with a back-end database. The **Device Database File**

field specifies an absolute path to the SQLite database on the device's file system. At runtime, the ADF Mobile client framework will open a connection to this database file and optionally initialize it according to the specified **Run Database Initialization Script** parameters. For more information, see Section 8.5.1, "How to Specify the Client Database Location."

- Standalone, (you do not enter a value into the **Device Database File** field)—This indicates the application does not require a database at all, and the Mobile client framework will not attempt to open one.

8.5.3 What Happens When Oracle Database Lite Mobile Server Manages an Application's Database

Oracle Database Lite manages the location of the Mobile Server-managed database. The general form for this location is: `SQLite.DATA_DIRECTORY/sqlite_db/syncUsername/applicationName.db`, where:

- `SQLite.DATA_DIRECTORY` is a value stored in `OSE.TXT`, which is found in a platform-dependent location:
  - BlackBerry—`/store/home/user/oracle/sync`

    Note: In general, `SQLite.DATA_DIRECTORY` points to `/SDCard/Databases/oracle` on BlackBerry.

  - Windows Mobile—`mobile_client_install_root\sqlite`, where `mobile_client_install_root` is usually `"\Program Files\ADFmc"`

    Note: In general, `SQLite.DATA_DIRECTORY` will be the same as `mobile_client_install_root`.

- `syncUsername` is a value gathered at runtime.

When the application requests data for the first time, it attempts to connect to the database at `SQLite.DATA_DIRECTORY/sqlite_db/syncUsername/applicationName.db`. If this database exists at the specified location, then the application connects to it and continues running. Otherwise, the user is prompted to enter synchronization credentials to allow Mobile Server to connect to the back-end database and perform an initial synchronization to populate the mobile database.

8.5.4 How the ADF Mobile Client Framework Retrieves Mobile Server Credentials at Application Startup

To enable synchronization between Mobile Server and multiple mobile devices, the ADF Mobile client framework requires that users provide the following authentication credentials supplied at runtime:

- Sync username
- Sync password
- Mobile Server hostname/IP address
- Whether to save the password

After a user starts an application on a smartphone, device, simulator, or emulator, the ADF Mobile client framework collects the username, password, Mobile Server URL, and password saving option that it prompts from the user when the application attempts to synchronize data with the Mobile Server. Figure 8–15 shows the page on a BlackBerry simulator that prompts users for these credentials.

Figure 8–15 Prompting User-Provided Mobile Server Credentials

The type of authentication policy you select in the Client Database page dictates which, if any, of the credentials that users must provide to synchronize application data. Figure 8–15, for example, illustrates how selecting the **Always Prompt for Credentials** option requires users to enter the username, password, and Mobile Server URL credentials when running an ADF Mobile client application on a BlackBerry simulator.

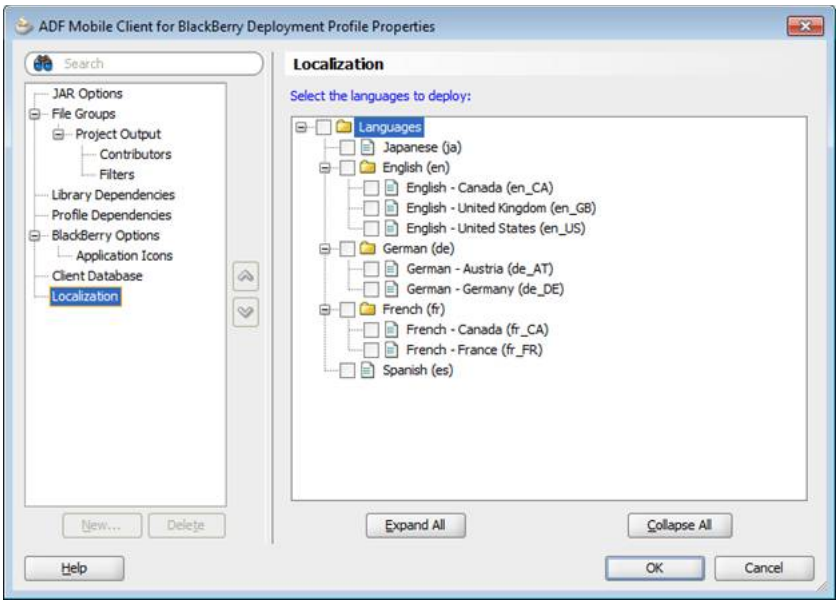
8.6 Deploying a Multi-Language ADF Mobile Client Application

You can select the language resource bundles for an application using the Localization panel shown in Figure 8–16. The Localization panel displays all of the resource bundles included in the base (server) application's JAR file that was imported to create the ADF Mobile client application using the ADF Components from ADF Library wizard described in Section 5.2, "Extending an ADF Application to Mobile Client."

8.6.1 How to Select the Language Resource Bundles for an ADF Mobile Client Application

The Localization page of the Deployment Profile Properties dialog enables you to set the locales for the deployment profile.

Figure 8–16 The Localization Page



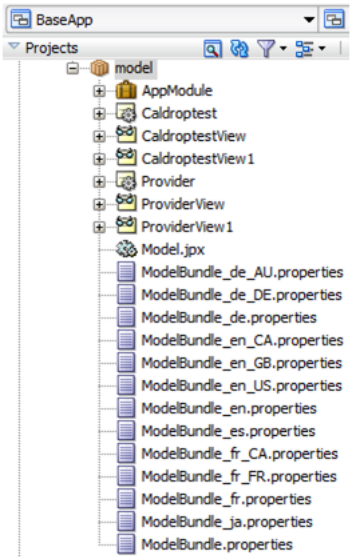
Before you begin:

Create a mobile client application comprised of the business components from the base application as described in Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects." This base application must include multi-language resource bundles.

Figure 8–16 shows the project structure of a base application called BaseApp whose language resource bundles include:

- `ModelBundle_de.properties`
- `ModelBundle_en_GB.properties`
- `ModelBundle_en_US.properties`
- `ModelBundle_fr.properties`

Figure 8–17 Project Structure of a Multi-Language Base (Server) Application

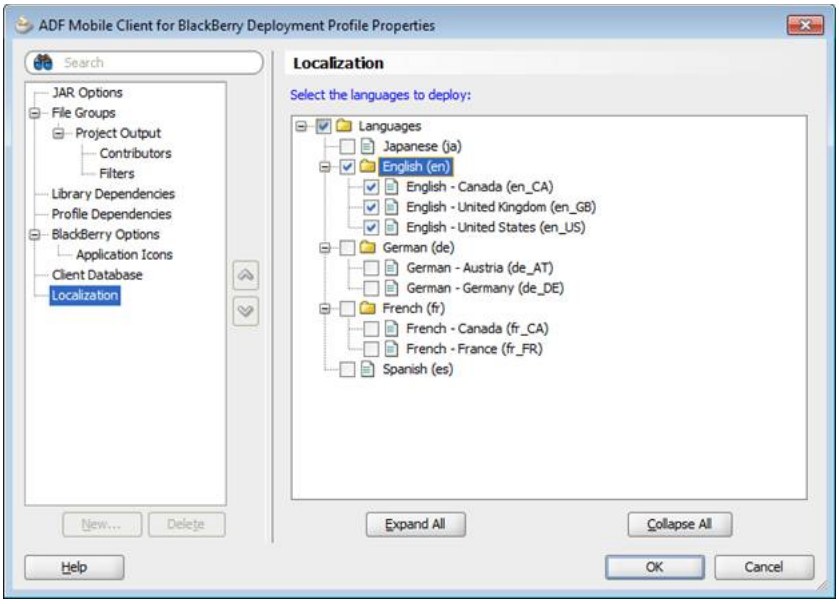


To select language resource bundles:

1. Open the deployment properties for the application by clicking **Application** then **Application Properties** and then **Deployment**.
2. Click **New** in the Deployment page.
3. In the Create Deployment Profile Dialog, choose either **ADF Mobile Client for BlackBerry** or **ADF Mobile Client for Windows Mobile**.
4. Enter a name for the deployment profile and then click **OK**.
5. For Windows Mobile applications, define the Windows Mobile Options and Client Database options as needed.
6. Click **Localization**.
7. Choose the appropriate language resource bundle taken from the base application's JAR. Figure 8–18 shows the Localization page lists the resource bundles of the base application JAR. Using the base application called BaseApp shown in Figure 8–17 as an example, an ADF Mobile client application derived from this application would include its language resource bundles, including `ModelBundle_de.properties`, `ModelBundle_en_GB.properties`, `ModelBundle_en.US.properties`, and `ModelBundle_fr.properties`.

   As shown in Figure 8–18, selecting a folder (such as *English (en)* in this illustration) automatically selects the locale-specific child nodes.

Figure 8–18 Selecting Resource Bundles



8. Click **OK**.
9. Deploy the application as described in Section 8.4.1.4 and Section 8.4.2.4.

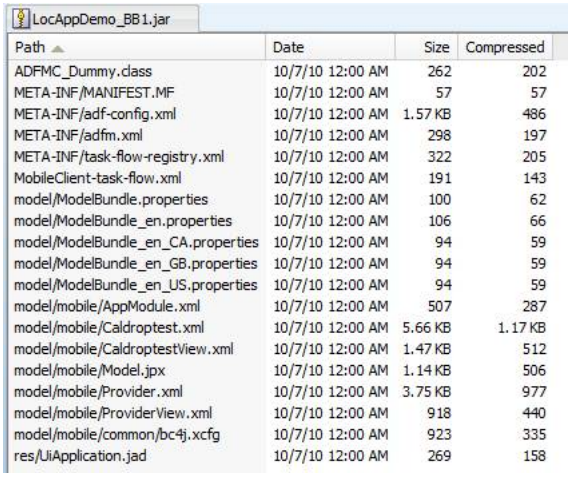
8.6.2 What Happens When You Add Language Resource Bundles to a Deployment Profile

The ADF Mobile client runtime automatically deploys `ModelBundle.properties`, the base language resource bundle to a JAR file. When you deploy an application, this

base language resource bundle, along with the resource bundles selected in the Localization page, are deployed the archive file.

Figure 8–19 shows the JAR file of an ADF Mobile client application called LocAppDemo, which was created from the base application called BaseApp. The resulting JAR file created by deploying LocAppDemo demonstrates that the base resource bundle, `ModelBundle.properties` along with the language resource bundles selected in the Localization page shown in Figure 8–18 (`ModelBundle_en.properties`, `ModelBundle_en_CA.properties`, `ModelBundle_en_GB.properties`, and `ModelBundle_en_US.properties`) and were deployed to a JAR file called `LocAappDemo_BB1.jar`.

Figure 8–19 Localization Bundles Deployed to a JAR


LocAppDemo_BB1.jar

Path	Date	Size	Compressed
ADFMC_Dummy.class	10/7/10 12:00 AM	262	202
META-INF/MANIFEST.MF	10/7/10 12:00 AM	57	57
META-INF/adf-config.xml	10/7/10 12:00 AM	1.57 KB	486
META-INF/adfm.xml	10/7/10 12:00 AM	298	197
META-INF/task-flow-registry.xml	10/7/10 12:00 AM	322	205
MobileClient-task-flow.xml	10/7/10 12:00 AM	191	143
model/ModelBundle.properties	10/7/10 12:00 AM	100	62
model/ModelBundle_en.properties	10/7/10 12:00 AM	106	66
model/ModelBundle_en_CA.properties	10/7/10 12:00 AM	94	59
model/ModelBundle_en_GB.properties	10/7/10 12:00 AM	94	59
model/ModelBundle_en_US.properties	10/7/10 12:00 AM	94	59
model/mobile/AppModule.xml	10/7/10 12:00 AM	507	287
model/mobile/Caldroptest.xml	10/7/10 12:00 AM	5.66 KB	1.17 KB
model/mobile/CaldroptestView.xml	10/7/10 12:00 AM	1.47 KB	512
model/mobile/Model.jpx	10/7/10 12:00 AM	1.14 KB	506
model/mobile/Provider.xml	10/7/10 12:00 AM	3.75 KB	977
model/mobile/ProviderView.xml	10/7/10 12:00 AM	918	440
model/mobile/common/bc4j.xcfg	10/7/10 12:00 AM	923	335
res/UiApplication.jad	10/7/10 12:00 AM	269	158

8.6.3 Adding Language Resource Bundles for Multiple Base Application JAR Files

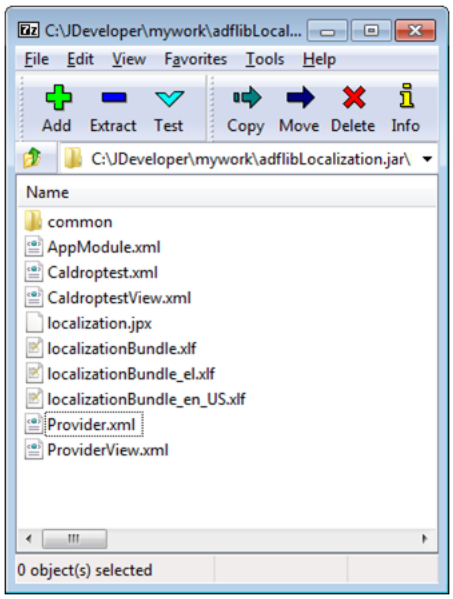
As stated in Section 7.4.2, "Supporting Localization through XLFF Resource Bundles," ADF Mobile client supports both XLIFF (XML Localization Interchange File Format) and `.properties` resource bundles.

8.6.3.1 How to Add Language Resource Bundles from Another Base Application

Using the Localization page, you can integrate the resource language bundles from different base application JARs.

Figure 8–20 shows a base application JAR called `adflibLocalization.jar` that contains XLFF-formatted resource bundles `localizationBundle_el.xlf` and `localizationBundle_en_US.xlf` as well as the base language resource bundle, `localizationBunlde.xlf`. ADF Mobile client's support of this format enables you to import these resource bundles into an application that also supports `.properties`.

Figure 8–20 Base Application JAR with XLFF-Formatted Resource Bundles



To add language resource bundles from another base application JAR file:

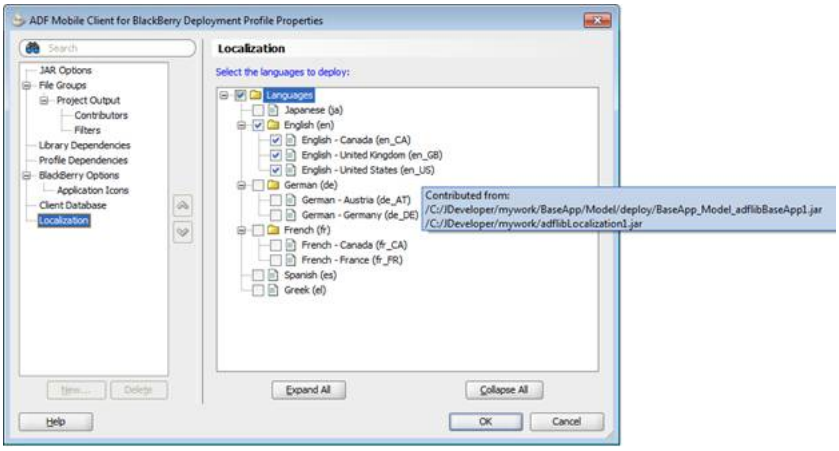
1. In the Application Navigator, right-click the MobileClient project.
2. Click **New**, select the **All Technologies** tab and then choose **Business Components from ADF Library** (located under Business Tier).
3. Complete the Create Business Components from ADF Library wizard as described in Section 5.2.1, "How to Create Subsets of Entity Objects and View Objects."
4. Select **Application** and then **Application Properties**.
5. Choose **Deployment** and the choose a deployment profile and then click **Edit**.
6. Choose **Localization** and check the resource bundles imported from the new base application JAR as well as those from the first base application JAR.

   As shown in Figure 8–21, the Localization page for the deployment profile of the LocAppDemo application displays resource bundles for Greek, French, and second `En_us` resource bundle from the second base application, `adflibLocalization.jar`. Figure 8–21 also shows how the tooltip displays the sources that contributed both of these `en_us` resource bundles:

   ```
   Contributed from:
   C:/JDeveloper/mywork/BaseApp/Model/deploy/BaseApp_Model_adflibBaseAppl.jar
   C:/JDeveloper/mywork/adflibLocalization1.jar
   ```

   As shown in this figure, the Localization page for LocAppDemo application shows that the United States English Language bundle was imported from resource bundles contributed by `adflibLocalization1.jar` and `BaseApp_Model_adflibBaseApp1.jar`.

Figure 8–21 Tooltips Showing Origin of Resource Bundle



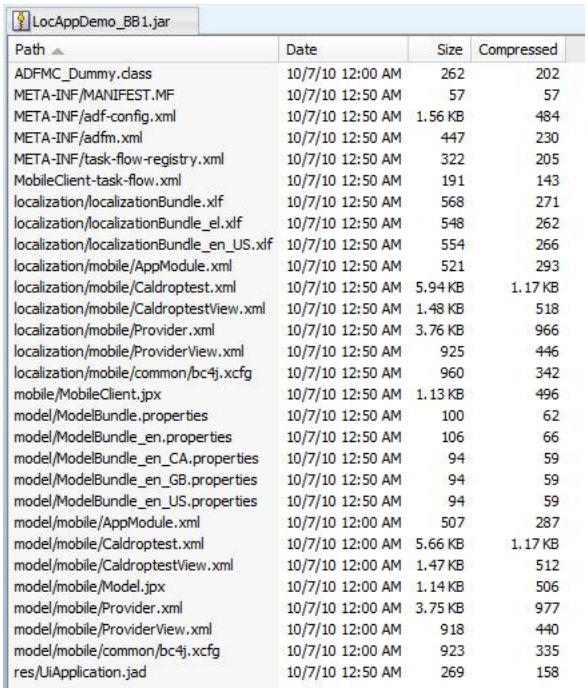
7. Select the resource bundle and then click **OK**.
8. Redeploy the application.

8.6.3.1.1 What Happens When You Add Language Resource Bundles from Another Base Application JAR

As shown in Figure 8–22, the resulting JAR file contains deployed language resource bundles from both base application JARs; it contains both base language bundles (`localizationBundle.xlf` and `ModelBundle.properties`). The JAR also contains the XLFF bundles from the second base application JAR as well as the English resource bundles (`ModelBundle_en.properties`, `ModelBundle_en_CA.properties`, `ModelBundle_en_GB.properties`, and `ModelBundle_en_US.properties`) from the first JAR.

As shown in Figure 8–22, LocAppDemo's JAR now has two United States English (`en_US`) resource bundles after the second deployment: because there are two corresponding language resource bundles for `en_US` from two different base application JARs, ADF Mobile client added `localizationBundle_en_US.xlf` along with `ModelBundle_en_US.properties`.

Figure 8–22 JAR Containing XLFFs and .properties Resource Bundles



LocAppDemo_BB1.jar

Path	Date	Size	Compressed
ADFMC_Dummy.class	10/7/10 12:00 AM	262	202
META-INF/MANIFEST.MF	10/7/10 12:50 AM	57	57
META-INF/adf-config.xml	10/7/10 12:50 AM	1.56 KB	484
META-INF/adfm.xml	10/7/10 12:50 AM	447	230
META-INF/task-flow-registry.xml	10/7/10 12:50 AM	322	205
MobileClient-task-flow.xml	10/7/10 12:50 AM	191	143
localization/localizationBundle.xlf	10/7/10 12:50 AM	568	271
localization/localizationBundle_el.xlf	10/7/10 12:50 AM	548	262
localization/localizationBundle_en_US.xlf	10/7/10 12:50 AM	554	266
localization/mobile/AppModule.xml	10/7/10 12:50 AM	521	293
localization/mobile/Caldroptest.xml	10/7/10 12:50 AM	5.94 KB	1.17 KB
localization/mobile/CaldroptestView.xml	10/7/10 12:50 AM	1.48 KB	518
localization/mobile/Provider.xml	10/7/10 12:50 AM	3.76 KB	966
localization/mobile/ProviderView.xml	10/7/10 12:50 AM	925	446
localization/mobile/common/bc4j.xcfg	10/7/10 12:50 AM	960	342
mobile/MobileClient.jpx	10/7/10 12:50 AM	1.13 KB	496
model/ModelBundle.properties	10/7/10 12:50 AM	100	62
model/ModelBundle_en.properties	10/7/10 12:50 AM	106	66
model/ModelBundle_en_CA.properties	10/7/10 12:50 AM	94	59
model/ModelBundle_en_GB.properties	10/7/10 12:50 AM	94	59
model/ModelBundle_en_US.properties	10/7/10 12:50 AM	94	59
model/mobile/AppModule.xml	10/7/10 12:00 AM	507	287
model/mobile/Caldroptest.xml	10/7/10 12:00 AM	5.66 KB	1.17 KB
model/mobile/CaldroptestView.xml	10/7/10 12:00 AM	1.47 KB	512
model/mobile/Model.jpx	10/7/10 12:00 AM	1.14 KB	506
model/mobile/Provider.xml	10/7/10 12:00 AM	3.75 KB	977
model/mobile/ProviderView.xml	10/7/10 12:00 AM	918	440
model/mobile/common/bc4j.xcfg	10/7/10 12:00 AM	923	335
res/UiApplication.jad	10/7/10 12:50 AM	269	158

8.6.3.2 Manually Adding Resource Bundles

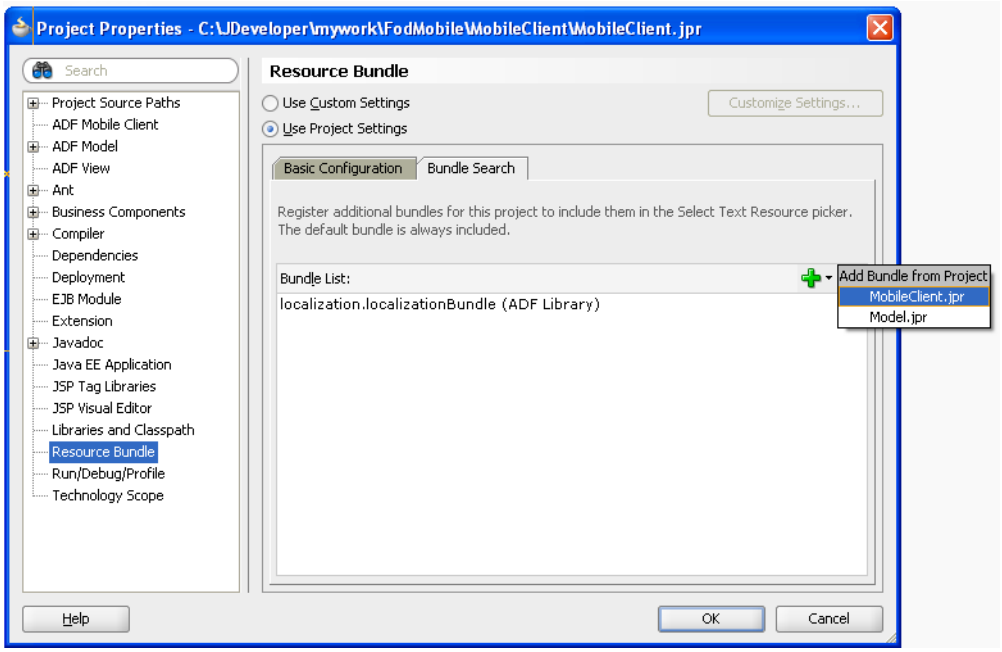
You can add a base language resource bundle to a MobileClient project and use it for a component, such as command button, using the Resource Bundle page accessed through Project Properties.

To add a resource bundle manually:

1. In the Application Navigator, right-click the MobileClient project.
2. In Project Properties, choose **Resource Bundles**.
3. Click the **Bundle Search** tab.
4. Click **Add** and then choose **MobileClient.jpr**.

   Figure 8–23 shows selecting the MobileClient project as the source of the resource bundle.

Figure 8–23 Selecting MobileClient.jpr

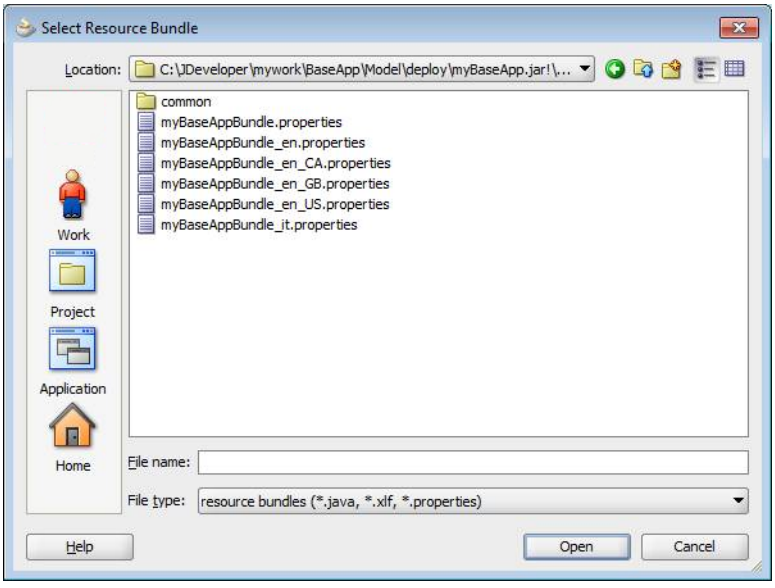


5. Browse to the base bundle label within the JAR and then click **Open**.

   As shown in Figure 8–24, the base JAR contains the following five language resource bundles:

```
myBaseAppBundle_en.properties
myBaseAppBundle_en_CA.properties
myBaseAppBundle_en_GB.properties
myBaseAppBundle_en_US.properties
myBaseAppBundle_en_it.properties
```

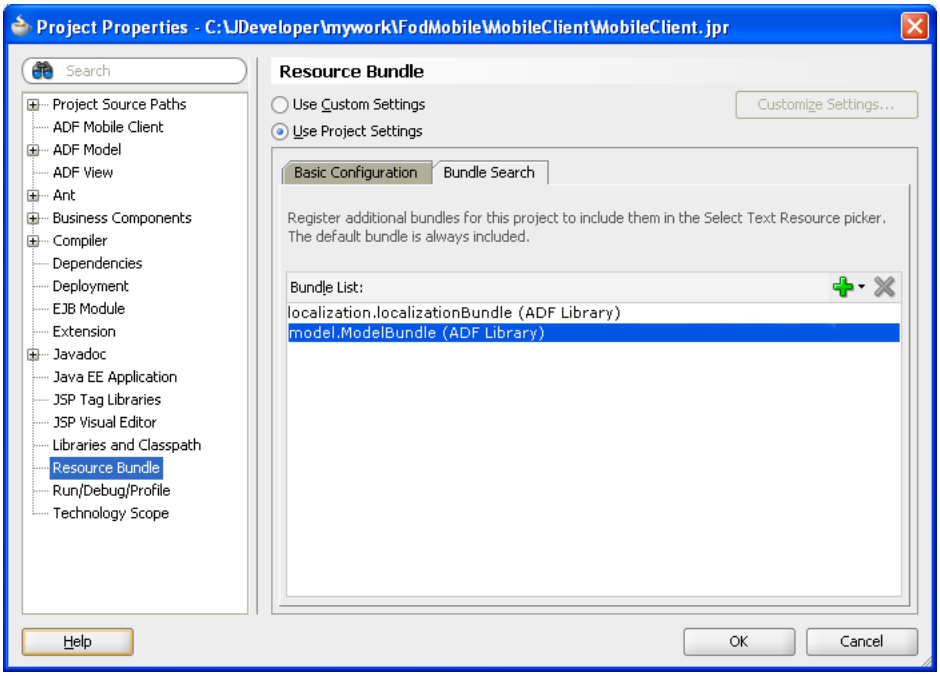
Figure 8–24 Selecting Language Resource Bundles



6. Click **OK**.

   Figure 8–25 shows the resource bundle added from the MobileClient project.

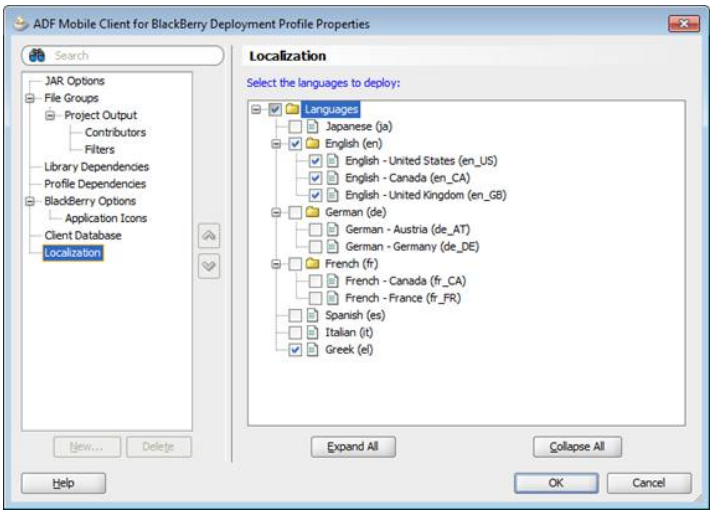
Figure 8–25 Adding the Resource Bundle from the MobileClient Project



8.6.3.2.1 What Happens When You Manually Add a Resource Bundle As shown in Figure 8–26, the Localization panel of the application deployment profile is updated to show the addition of the new resource bundle. This figure shows the Italian language resource bundle was imported from the base JAR, as was another `en_US` resource bundle.

> **Tip:** Using the tooltip enables you to see the origin of the resource bundle.

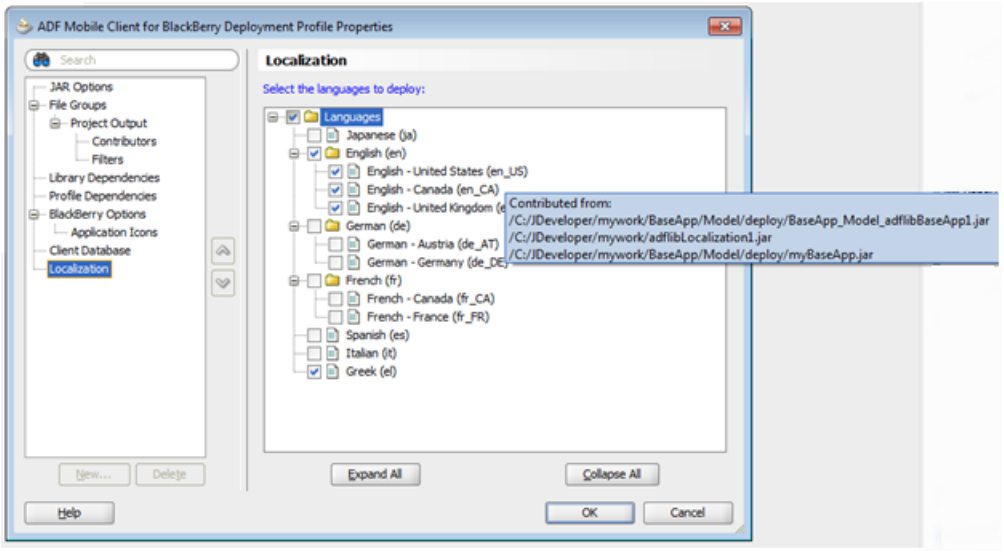
Figure 8–26 Localization Panel Updated with Manually Added Resource Bundles



Using the tooltips, as shown in Figure 8–27, reveals that `en_US` has been contributed from the following three sources, including the MobileClient project (`myBassApp.jar`):

```
C:/JDeveloper/mywork/BaseApp/Model/deploy/BaseApp_Model_adflibBaseApp1.jar
C:/JDeveloper/mywork/adflibLocalization1.jar
C:/JDeveloper/mywork/BaseApp/Model/myBaseApp.jar
```

Figure 8–27 Tooltips Showing Source of Manually Added Resource Bundle



After you redeploy the application, the updated JAR will contain all of the added resource bundles.

As shown in Figure 8–28, the various `en_US` resource bundles are those included in the deployed JAR file.

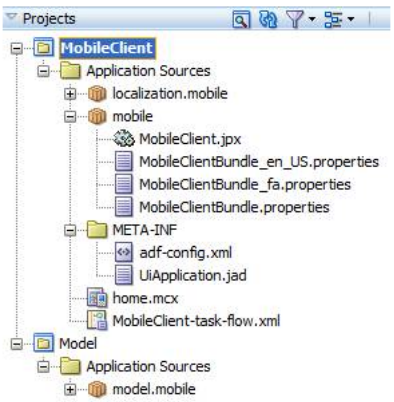
Figure 8–28 JAR Containing Resource Bundles from Various Sources

Path	Date	Size	Compressed
ADFMC_Dummy.class	10/7/10 2:16 AM	262	202
META-INF/MANIFEST.MF	10/7/10 2:20 AM	57	57
META-INF/adf-config.xml	10/7/10 2:20 AM	1.57 KB	488
META-INF/adfm.xml	10/7/10 2:20 AM	447	230
META-INF/task-flow-registry.xml	10/7/10 2:20 AM	322	205
MobileClient-task-flow.xml	10/7/10 2:17 AM	191	143
localization/localizationBundle.xlf	10/7/10 2:20 AM	568	271
localization/localizationBundle_en_US.xlf	10/7/10 2:20 AM	554	266
localization/mobile/AppModule.xml	10/7/10 2:17 AM	521	293
localization/mobile/Caldroptest.xml	10/7/10 2:17 AM	5.94 KB	1.17 KB
localization/mobile/CaldroptestView.xml	10/7/10 2:17 AM	1.48 KB	518
localization/mobile/Provider.xml	10/7/10 2:17 AM	3.76 KB	966
localization/mobile/ProviderView.xml	10/7/10 2:17 AM	925	446
localization/mobile/common/bc4j.xcfg	10/7/10 2:17 AM	960	342
mobile/MobileClient.jpx	10/7/10 2:17 AM	1.13 KB	496
model/ModelBundle.properties	10/7/10 2:20 AM	100	62
model/ModelBundle_en.properties	10/7/10 2:20 AM	106	66
model/ModelBundle_en_CA.properties	10/7/10 2:20 AM	94	59
model/ModelBundle_en_GB.properties	10/7/10 2:20 AM	94	59
model/ModelBundle_en_US.properties	10/7/10 2:20 AM	94	59
model/mobile/AppModule.xml	10/7/10 2:16 AM	507	287
model/mobile/Caldroptest.xml	10/7/10 2:16 AM	5.66 KB	1.17 KB
model/mobile/CaldroptestView.xml	10/7/10 2:16 AM	1.47 KB	512
model/mobile/Model.jpx	10/7/10 2:16 AM	1.14 KB	506
model/mobile/Provider.xml	10/7/10 2:16 AM	3.75 KB	977
model/mobile/ProviderView.xml	10/7/10 2:16 AM	918	440
model/mobile/common/bc4j.xcfg	10/7/10 2:16 AM	923	335
model/myBaseAppBundle.properties	10/7/10 2:20 AM	100	62
model/myBaseAppBundle_en.properties	10/7/10 2:20 AM	106	66
model/myBaseAppBundle_en_CA.properties	10/7/10 2:20 AM	94	59
model/myBaseAppBundle_en_GB.properties	10/7/10 2:20 AM	94	59
model/myBaseAppBundle_en_US.properties	10/7/10 2:20 AM	94	59
model/myBaseAppBundle_it.properties	10/7/10 2:20 AM	106	66
res/UiApplication.jad	10/7/10 2:20 AM	278	161

8.6.3.3 Adding Local Resource Bundles

If an ADF Mobile client application includes local resource bundles, such as Persian (`MobileCientBundle_fa.properties`, shown in Figure 8–29), the Localization page also displays these resource bundles.

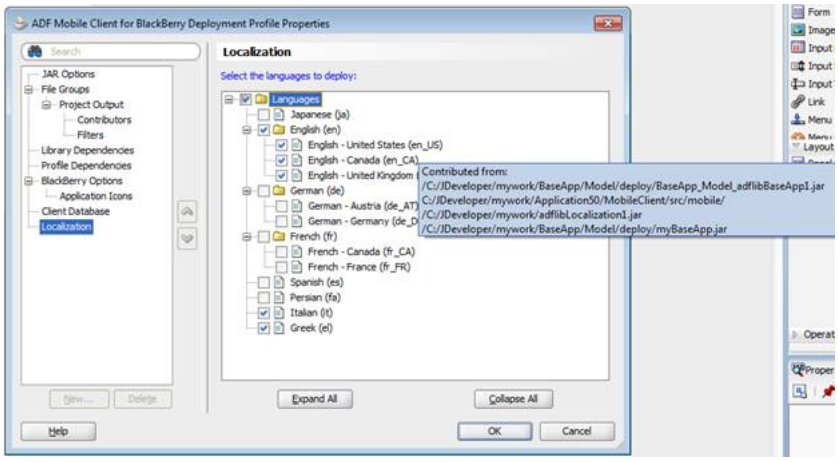
Figure 8–29 A MobileClient Project with Local Language Resource Bundles



You can verify the origin of the local resource bundle using the tooltip as shown in Figure 8–30. For example, the tooltips in this figure reveal the local source for the `en_US` resource bundle as `Application50`:

```
Contributed from:
C:/JDeveloper/mywork/BaseApp/Model/deploy/BaseApp_Model_adflibBaseApp1.jar
C:/JDeveloper/mywork/Application50/MobileClient/src/mobile
C:/JDeveloper/mywork/adflibLocalization1.jar
C:/JDeveloper/mywork/BaseApp/Model/myBaseApp.jar
```

Figure 8–30 Verifying the Origins of the Resource Bundles Using Tooltips



When you deploy the application, the resulting JAR includes the local language bundles. As shown in Figure 8–31 includes four different `en_US` resource bundles.

Figure 8–31 Local Resource Bundles Included in JAR

Path	Date	Size	Compressed
ADFMC_Dummy.class	10/7/10 2:16 AM	262	202
META-INF/MANIFEST.MF	10/7/10 2:34 AM	57	57
META-INF/adf-config.xml	10/7/10 2:34 AM	1.57 KB	488
META-INF/adfm.xml	10/7/10 2:34 AM	447	230
META-INF/task-flow-registry.xml	10/7/10 2:34 AM	322	205
MobileClient-task-flow.xml	10/7/10 2:34 AM	338	206
home.mcx	10/7/10 2:34 AM	643	309
localization/localizationBundle.xlf	10/7/10 2:34 AM	568	271
localization/localizationBundle_el.xlf	10/7/10 2:34 AM	548	262
localization/localizationBundle_en_US.xlf	10/7/10 2:34 AM	554	266
localization/mobile/AppModule.xml	10/7/10 2:34 AM	521	293
localization/mobile/Caldroptest.xml	10/7/10 2:34 AM	5.94 KB	1.17 KB
localization/mobile/CaldroptestView.xml	10/7/10 2:34 AM	1.48 KB	518
localization/mobile/Provider.xml	10/7/10 2:34 AM	3.76 KB	966
localization/mobile/ProviderView.xml	10/7/10 2:34 AM	925	446
localization/mobile/common/bc4j.xcfg	10/7/10 2:34 AM	960	342
mobile/MobileClient.jpx	10/7/10 2:34 AM	1.13 KB	496
mobile/MobileClientBundle.properties	10/7/10 2:34 AM	4	6
mobile/MobileClientBundle_en_US.properties	10/7/10 2:34 AM	4	6
model/ModelBundle.properties	10/7/10 2:34 AM	100	62
model/ModelBundle_en.properties	10/7/10 2:34 AM	106	66
model/ModelBundle_en_CA.properties	10/7/10 2:34 AM	94	59
model/ModelBundle_en_GB.properties	10/7/10 2:34 AM	94	59
model/ModelBundle_en_US.properties	10/7/10 2:34 AM	94	59
model/mobile/AppModule.xml	10/7/10 2:16 AM	507	287
model/mobile/Caldroptest.xml	10/7/10 2:16 AM	5.66 KB	1.17 KB
model/mobile/CaldroptestView.xml	10/7/10 2:16 AM	1.47 KB	512
model/mobile/Model.jpx	10/7/10 2:16 AM	1.14 KB	506
model/mobile/Provider.xml	10/7/10 2:16 AM	3.75 KB	977
model/mobile/ProviderView.xml	10/7/10 2:16 AM	918	440
model/mobile/common/bc4j.xcfg	10/7/10 2:16 AM	923	335
model/myBaseAppBundle.properties	10/7/10 2:34 AM	100	62
model/myBaseAppBundle_en.properties	10/7/10 2:34 AM	106	66
model/myBaseAppBundle_en_CA.properties	10/7/10 2:34 AM	94	59
model/myBaseAppBundle_en_GB.properties	10/7/10 2:34 AM	94	59
model/myBaseAppBundle_en_US.properties	10/7/10 2:34 AM	94	59
model/myBaseAppBundle_it.properties	10/7/10 2:34 AM	106	66
res/UiApplication.jad	10/7/10 2:34 AM	278	161

9

Synchronizing ADF Mobile Client Data and Transactions

This chapter provides an overview of how to set up Oracle Database Lite Mobile Server (Mobile Server) to work with ADF Mobile client so that you can publish data and synchronize data between back-end servers and mobile devices and smartphones or their respective simulators and emulators.

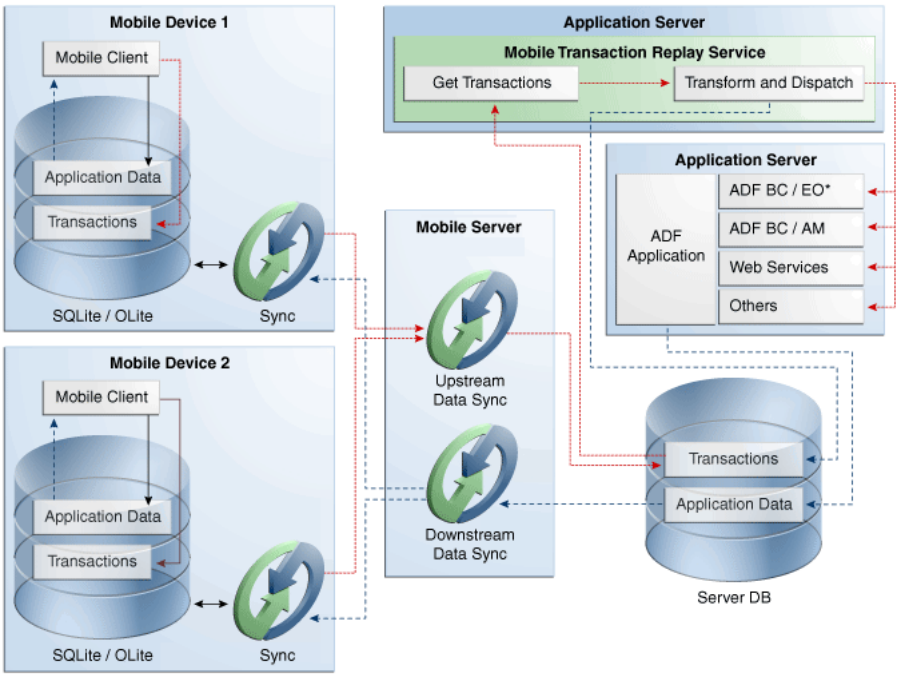
This chapter includes the following sections:

For general information on Mobile Server, see *Oracle Database Lite Getting Started Guide*.

9.1 About Synchronizing Data with Oracle Mobile Server

As illustrated in Figure 9–1, Oracle Database Lite Mobile Server (Mobile Server) serves as a bi-directional data conduit between your mobile device and the base ADF application server. When synchronizing data downstream from server to device, SQL statements are executed directly against the mobile database to bring its state in line with that of the enterprise database. When synchronizing data upstream, however—from device to server—it is not safe to push the data directly through unchecked. Even though ADF Mobile client supports data validation on the device, there may be additional validation rules and business processes that must execute before the enterprise database can be safely updated with changes from the mobile device. While these validation rules and business processes can be executed immediately in the context of the base ADF Faces application, this is not the case for mobile applications, which may not have network connectivity, or which may otherwise be unable to access corporate resources. This is where ADF Mobile transaction replay service comes in. Transaction replay service encapsulates database changes in XML records known as *Replay Items*. Replay items are then "replayed" in the context of the base ADF Faces application and a given user so that the same validations and business processes that would normally be applied to a Web application can also be applied to an ADF Mobile client application.

Figure 9–1 Upstream and Downstream Synchronization



9.1.1 About ADF Mobile Transaction Replay Service

ADF Mobile transaction replay service is a server-side component which provides a general mechanism for deferred execution (replay) of arbitrary server-side actions that are unavailable to a client. Clients generate Replay Items, XML records containing information about which method to call on the server and which parameters should be passed to that method. Later, those XML records are pushed back to the server, parsed by transaction replay service, and finally converted into appropriate method calls which are executed.

9.1.2 About the Connection Between Client and Server

Mobile Server receives requests and passes data between the ADF Mobile client application and the Windows Mobile device or BlackBerry smartphone. To publish application data, you create a connection between the mobile client application and the Oracle Mobile Server Repository Schema by specifying login credentials to Mobile Server.

The credentials required to connect the mobile client application to Mobile Server are not the same as the login credentials for the database connection to the Mobile Server repository schema. The login credentials for the mobile application's connection to the Oracle Mobile Server repository schema are also used later in the deployment on the device or device emulator when you synchronize changes back to the server. These login credentials are set during the installation of Oracle Database Lite. For more information, see "Installation of Oracle Database Lite" in *Oracle Database Lite Getting Started Guide*.

For the publication process, you specify where data will be stored.

9.1.3 About Publishing Data

Application data is published for deployment to the mobile device through Mobile Server, using publications and publication items that are based on mobile entity objects.

When you deploy the ADF Mobile client from JDeveloper, ADF Mobile client automatically creates publications used by Mobile Server to determine the server-side and client-side data schemas. ADF Mobile client creates a mobile sync user as well as subscriptions to the publications. Finally, the ADF Mobile client application is deployed to the target mobile device, smartphone, emulator, or simulator. For more information, see Section 8.3, "Creating Data Sync Publications on the Server."

9.1.4 What Happens When You Make Changes to the Mobile Database

When a user interacts with your mobile client application in the ADF Mobile client framework, this is what happens:

1. The user makes changes to the data and saves.
2. `DBTransaction.commit()` is invoked, which in turn generates appropriate DML statements for all changes to the data model.
3. As each statement is posted to the database, an event is fired, which in turn generates an XML record—an entity Replay Item—which encapsulates the details of the INSERT, UPDATE, or DELETE statement.
4. After the transaction is committed, another event is fired, which causes the previously-generated XML records to be written to the database in a separate transaction.
5. At some later point, data synchronization is initiated, either on-demand or in the background, and Oracle Mobile Server transfers the Replay Item(s) up to the Mobile Server instance.
6. Transaction replay service monitors the Replay Item queue and converts the XML records into appropriate method calls as they arrive.
7. These method calls effectively simulate a specific user making the same changes to the data model in the context of a standard ADF application. All standard validations and business processes are applied.
8. If data satisfies all necessary criteria, then it is committed to the enterprise database. Otherwise, an error is recorded in Mobile Server and then delivered to the mobile client the next time synchronization occurs.

9.1.5 What Happens When You Import Entity Objects into the Mobile Client Application

When you import entity objects from the base ADF application into your mobile client application, JDeveloper automatically inserts additional metadata into the entity definitions to support replay-item generation at runtime. This metadata takes the form of `EventPub` elements and `EventDef` elements. `EventDef` elements define the payload of required attributes that must be serialized into the Replay Item when an event fires. This payload can be thought of as the parameters that will be passed to transaction replay service, so that it in turn can invoke the appropriate server-side method. `EventPub` elements map a named `EventDef` element to an event generation point, such as `CREATE`, `UPDATE`, or `DELETE`. By default, every imported entity object will contain three event definitions:

- `TRS_Delete` —This is triggered in response to the deletion of a row. The payload is comprised of just the primary key attributes.

- `TRS_Create`—This is triggered in response to the creation of a new row. The payload consists of all attributes.
- `TRS_Update`—This is triggered in response to the modification of an existing row. The payload consists of all modified attributes.

9.2 Configuring Oracle Mobile Server

To deploy and test an ADF Mobile client application, you must provide JDeveloper with Mobile Server information. For JDeveloper, you use the Mobile Server user name, password, and host name to:

- Create a database connection for Mobile Server.
- Specify database credentials for an administrative user to be able to create publications and publication items for synchronizing data.
- Set up a sync user and the sync user's database credentials to be used by the test application to synchronize with the server. The mobile sync user credentials you specify subscribe the mobile sync user to the publications.

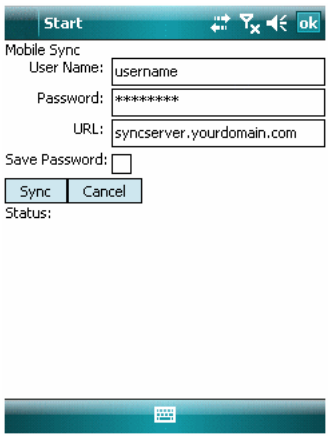
For details on how to install and configure Mobile Server, see "Installation of Oracle Database Lite" in *Oracle Database Lite Getting Started Guide*.

Note: A sync user cannot have two schemas on the same device.

9.3 Initiating Data Synchronization

Although you can create ADF Mobile client applications that do not require synchronization because they either use a standalone database or no database at all, most ADF Mobile client applications require synchronization with a back-end database. For applications that require this type of synchronization, the ADF Mobile client runtime attempts to locate a database on the mobile device, smartphone or their respective simulators and emulators the first time that the application requires data. If the runtime is unable to find this client database, it creates one through synchronization with Oracle Database Lite Mobile Server (Mobile Server). To begin this process, the runtime prompts the end user to define synchronization credentials and various connection parameters in a dialog similar to the one shown in Figure 9–2.

Figure 9–2 Synchronization Credentials Dialog



After a user enters the correct values and clicks **Sync**, the ADF Mobile client runtime retrieves this data from the Mobile Server and creates a SQLite database instance that it stores on the mobile device or smartphone. The application continues to run after synchronization completes.

9.4 Enabling Data Synchronization at Application Startup

As described in Section 8.5.3, "What Happens When Oracle Database Lite Mobile Server Manages an Application's Database," if you configure the application's deployment profile to use Mobile Server, ADF Mobile client automatically uses data synchronization to populate the device's local database when you start an application for the first time. The ADF Mobile client framework requires the following to enable synchronization between Mobile Server and multiple mobile devices:

- Sync username
- Sync password
- Mobile Server hostname/IP address
- Whether to save the password

ADF Mobile client uses the following three modes for collecting these parameters at application startup. You can set these modes by selecting the Authentication Policy options in the Client Database page as described in Section 8.5.3, "What Happens When Oracle Database Lite Mobile Server Manages an Application's Database."

9.4.1 How to Invoke Data Synchronization Programmatically

You activate data synchronization after your application has started by constructing a `SyncOptions` object as illustrated in Example 9–1.

Example 9–1 Calling a SyncOptions Object for Provided Sync Parameters

```
SyncOptions options = new SyncOptions("username", "password",
"http://syncserver.my.domain.com");
myAppModule.sync(options);
```

Constructing a `SyncOptions` object enables the display of a synchronization UI and automatically initiates data synchronization with the specified parameters while displaying progress and error messages as necessary without prompting the user for credentials.

If the data synchronization operation is successful, the synchronization UI will automatically be dismissed and returns users to the previous page of the application. If any errors occurred, the synchronization UI remains visible, enabling the user to either cancel the operation or correct any issues and try again. After a successful synchronization operation, the client database has the latest changes from the server database. Those changes, however, are not reflected in the application until you refresh view objects by re-executing them.

9.4.1.1 Providing Parameters for Data Synchronization

The `SyncOptions` object encapsulates the parameters required by the synchronization operation.

- `Username`— The sync user name.
- `Password`—The sync password.

- `Server`—The URL of the Mobile Server. For example: `http://syncserver.my.domain.com`
- `SyncProgressListener`—Enables you to specify your own UI for handling sync progress events and errors by implementing `oracle.adfnmc.sync.SyncProgressListener` as illustrated in Example 9–2.
- `SavePassword`—Whether the password should be saved for future use; default is `false`.
- `SyncMode`—One of the following constants, defined in `oracle.adfnmc.sync.SyncOptions`, that specifies how sync credentials are to be provided:
  - `SYNC_CREDENTIALS_MODE_PROMPT`—Prompt the user for all necessary parameters.
  - `SYNC_CREDENTIALS_MODE_EL`—Use EL expressions which have already been populated with the required parameters. See also Table 9–2.
  - `SYNC_CREDENTIALS_MODE_LASTUSER`—Reuse the previously saved credentials.
  - `SYNC_CREDENTIALS_MODE_PARAMS`—The caller is providing all required parameters directly in the `SyncOptions` container.
  - `SYNC_CREDENTIALS_MODE_SAME_AS_STARTUP`—Use the same mode that was used at application startup, specified in the `sync-credentials-mode` key; this value is `SYNC_CREDENTIALS_MODE_LASTUSER` by default.

The user will be prompted to enter any missing values if not all of the required parameters have been provided, regardless of which `SyncMode` is specified. Table 9–1 lists examples of various synchronization modes.

Table 9–1 Sync Options

Sync Mode	Example
Developer-Provided Credentials	`SyncOptions options = new SyncOptions("username", "password", "http://syncserver.my.domain.com");`
User-Provided Credentials	`SyncOptions options = new SyncOptions(SyncOptions.SYNC_CREDENTIALS_MODE_PROMPT);`

Use Data Controls	`// Bind UI controls to the following expressions:`
	`// #{syncContext.userName}`
	`// #{syncContext.password}`
	`// #{syncContext.url}`
	`SyncOptions options = new SyncOptions(SyncOptions.SYNC_CREDENTIALS_MODE_EL);`
	`// When sync begins, the required parameters are read from the EL expressions above`

Use Previously Save Parameters	`SyncOptions options = new SyncOptions(SyncOptions.SYNC_CREDENTIALS_MODE_LASTUSER);`
Use Mode Specified at Application Startup	`SyncOptions options = new SyncOptions(SyncOptions.SYNC_CREDENTIALS_MODE_SAME_AS_STARTUP);`

9.4.1.1.1 Sync Progress Events Mobile Server notifies the application with status messages, progress, and errors as the sync operation proceeds. By default, ADF Mobile client uses `oracle.adfnmc.sync.ELExpressionSyncProgressListener` to receive these events and display them in the default sync UI, but you can create your own listener by implementing `oracle.adfnmc.sync.SyncProgressListener` and registering it with the `SyncOptions` container. Example 9–2 illustrates a simple progress listener that logs data sync events to the console.

Example 9–2 Progress Listener

```
public class ConsoleSyncProgressListener implements SyncProgressListener
{
  public void progress(String message, int progress, int phase)
  {
    System.out.println("Sync phase: " + phase + ", progress %" + progress + ",
message = " + message);
  }
}
SyncOptions options = new SyncOptions(SyncOptions.SYNC_CREDENTIALS_MODE_PROMPT);
options.setSyncProgressListener(new ConsoleProgressListener());
myAppModule.sync(options);
```

9.4.2 SQLite Database Locking and Mobile Server

By design, SQLite employs very coarse-grained locking, so any writer must acquire an exclusive lock on the whole database. This has two implications:

1. There can be either multiple read-only connections or a single read-write connection to any given database.
2. A commit operation will succeed if there are any open cursors, but a rollback operation will not.

In a typical ADF Mobile client application, the application module owns a read-write database connection through which you indirectly access the database. When you call `ApplicationModule.sync()`, this database connection is implicitly shared with Mobile Server so that it can make modifications to the client database. If sync succeeds, these modifications are committed to the database, and your application continues running. If sync fails for any reason, Mobile Server rolls back the pending changes. Because the ADF Mobile client framework may have open cursors, the rollback operation will often fail. Under such circumstances the ADF Mobile client framework does the following:

1. Preserves the state of the existing view object.
2. Disconnects from the database.
3. Reconnects to the database.

This effectively rolls back the Mobile Server transaction and enables the user to continue using the application as though sync had never been invoked.

9.5 Customizing the Synchronization Setup

You can customize the setup for mobile synchronization by creating a page that prompts users when they need to synchronize.

9.5.1 Creating a Custom Page for Mobile Synchronization

To equip an application with a page that tells users when they need to synchronize the application, you create an additional task flow for the application that includes the MCX file for the synchronization page itself. You then update the application task flow to point to the custom synchronization task flow.

9.5.1.1 How to Create a Custom Synchronization Page

To enable this page (the custom sync page), you must create an additional ADF Mobile client task flow as well as an MCX file using the ADF Mobile client page and task flow dialogs.

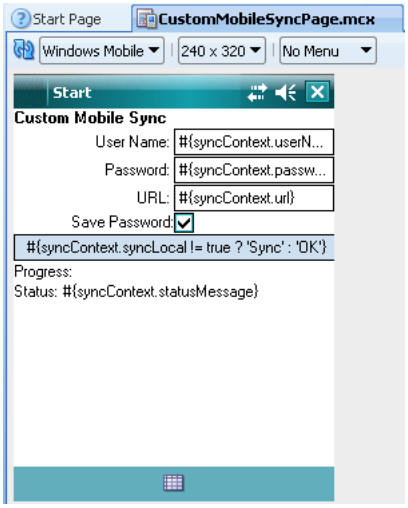
Before you begin:

Familiarize yourself with creating MCX pages and task flows as described in Chapter 6, "Creating the ADF Mobile Client User Interface."

To create the custom sync page:

1. Right-click the `MobileClient` project and choose **File** then **New** then **ADF Mobile Client** and then **ADF Mobile Client Page**.
2. Create an MCX file. For example, create a page called `CustomMobileSyncPage.mcx`, shown in Figure 9–3 with code similar to Example 9–3.

Figure 9–3 The Custom Sync Page



Example 9–3 A Custom Sync Page

```
<?xml version='1.0' encoding='windows-1252'?>
<amc:view xmlns:amc="http://xmlns.oracle.com/jdev/amc">
         <amc:form onShow="#{syncContext.MobileSyncBean.onShow.execute}">
            <amc:panelGroupLayout id="home" layout="vertical">
                   <amc:outputText id="lblTitle" value="Custom Mobile Sync" fontWeight="bold" />
                   <amc:panelFormLayout>
                     <amc:inputText id="txtUserName" label="User Name: "
value="#{syncContext.userName}" />
                     <amc:inputText id="txtPassword" label="Password: "
value="#{syncContext.password}" secret="true" />
                     <amc:inputText id="txtUrl" label="URL: " value="#{syncContext.url}" />
```

```
                    <amc:selectBooleanCheckbox id="cbSave" label="Save Password:"
value="#{syncContext.saveCredentials}" />
                </amc:panelFormLayout>
                <amc:panelGroupLayout layout="horizontal">
                    <amc:commandButton id="btnGo" disabled="#{syncContext.syncInProgress}"
actionListener="#{syncContext.MobileSyncBean.sync.execute}" text="#{syncContext.syncLocal != true ?
'Sync' : 'OK'}" />
                    <amc:commandButton id="btnStop"
actionListener="#{syncContext.MobileSyncBean.cancel.execute}" text="Cancel" />
                </amc:panelGroupLayout>
                <amc:panelGroupLayout layout="horizontal">
                        <amc:outputText id="lblProgress"
rendered="#{syncContext.syncInProgress}" value="Progress:" />
                </amc:panelGroupLayout>
                <amc:outputText id="lblStatus" value="Status: #{syncContext.statusMessage}"
foregroundColor="#{syncContext.error == null ? '#000000' : '#FF0000'}"/>
            </amc:panelGroupLayout>
        </amc:form>
</amc:view>
```

For the page to function properly, it must include the following:

- `onShow` handling to reset values on the page:

  ```
  <amc:form onShow="#{syncContext.MobileSyncBean.onShow}">
  ```

- User name, password, and URL information are defined within `inputText` components with the following values:

  ```
  <amc:inputText id="txtUserName" label="User Name: "
  value="#{syncContext.userName}" />
  <amc:inputText id="txtPassword" label="Password: "
  value="#{syncContext.password}" secret="true" />
  <amc:inputText id="txtUrl" label="URL: " value="#{syncContext.url}" />
  ```

- To save credentials, create a `checkbox` component with the following value:

  ```
  <amc:selectBooleanCheckbox id="cbSave" label="Save Password:"
  value="#{syncContext.saveCredentials}" />
  ```

- To initiate the sync process, you must have the following listener tied to a control:

  ```
  <amc:commandButton id="btnGo" disabled="#{syncContext.syncInProgress}"
  actionListener="#{syncContext.MobileSyncBean.sync}"
  text="#{syncContext.syncLocal != true ? 'Sync' : 'OK'}" />
  ```

- To terminate the sync process (which is optional), you must have the following listener tied to a control:

  ```
  <amc:commandButton id="btnStop"
  actionListener="#{syncContext.MobileSyncBean.cancel}" text="Cancel" />
  ```

The remaining content in Example 9–3 is optional. Table 9–2 lists the sync-related EL expressions used in this page.

Table 9–2 Sync EL Expression Reference

Expression	Description
`#{syncContext.error}`	Contains any exception that may have been thrown by a synchronization operation.

Table 9–2 (Cont.) Sync EL Expression Reference

Expression	Description
`#{syncContext.syncInProgress}`	Inquires if a synchronization operation is currently running.
`#{syncContext.userName}`	The sync user name.
`#{syncContext.password}`	The sync password.
`#{syncContext.saveCredentials}`	Inquires if the password should be saved.
`#{syncContext.url}`	The address of the Mobile Server against which to synchronize.
`#{syncContext.statusMessage}`	A brief message describing the current synchronization phase (for example, sending data, receiving data).
`#{syncContext.currentProgress}`	The percent complete of the current sync phase.

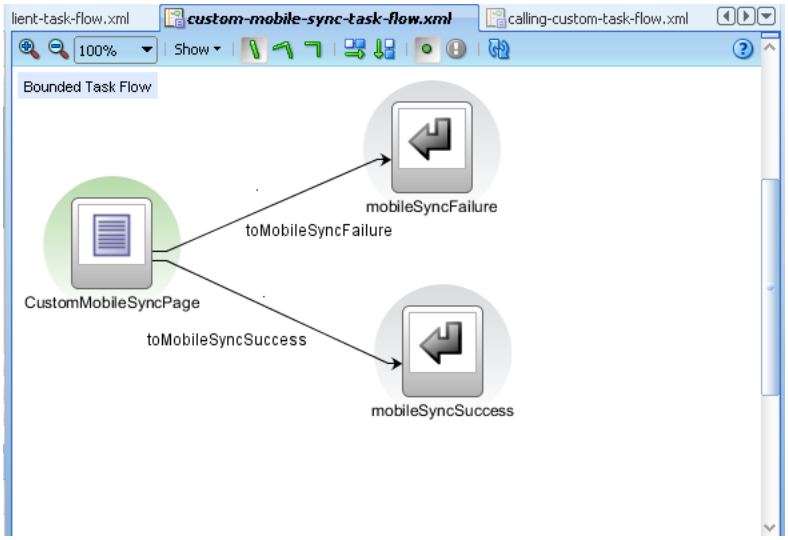
3. Right-click the MobileClient project and choose **File** then **New** then **ADF Mobile Client** and then **ADF Mobile Client Task Flow**.
4. Create the bounded task flow. For example, create a task flow called `custom-mobile-sync-task-flow.xml` using code similar to that illustrated in Example 9–4.

Example 9–4 The Sync Task Flow

```
<?xml version="1.0" encoding="windows-1252" ?>
<adfc-config xmlns="http://xmlns.oracle.com/adf/controller" version="1.2">
  <task-flow-definition id="custom-mobile-sync-task-flow">
    <default-activity id="__1">CustomMobileSyncPage</default-activity>
    <view id="CustomMobileSyncPage">
      <page>/CustomMobileSyncPage.mcx</page>
    </view>
    <task-flow-return id="mobileSyncSuccess">
      <outcome id="__2">
        <name>mobileSyncSuccess</name>
      </outcome>
    </task-flow-return>
    <task-flow-return id="mobileSyncFailure">
      <outcome id="__3">
        <name>mobileSyncFailure</name>
      </outcome>
    </task-flow-return>
    <control-flow-rule id="__4">
      <from-activity-id id="__5">CustomMobileSyncPage</from-activity-id>
      <control-flow-case id="__6">
        <from-outcome id="__7">toMobileSyncSuccess</from-outcome>
        <to-activity-id id="__8">mobileSyncSuccess</to-activity-id>
      </control-flow-case>
      <control-flow-case id="__9">
        <from-outcome id="__10">toMobileSyncFailure</from-outcome>
        <to-activity-id id="__11">mobileSyncFailure</to-activity-id>
      </control-flow-case>
    </control-flow-rule>
  </task-flow-definition>
</adfc-config>
```

Figure 9–4 shows the task flow created from the code illustrated in Example 9–4.

Figure 9–4 Bounded Task Flow for Sync



5. In the application task flow, add a task flow call that points to the sync task flow, such as `CustomMobileSyncTaskFlowCall` in Example 9–5. For more information on task flow calls, see "Using Task Flow Call Activities" in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

Example 9–5 The Task Flow Call to the Sync Task Flow

```
<task-flow-call id="CustomMobileSyncTaskFlowCall">
        <task-flow-reference>
            <document>/custom-mobile-sync-task-flow.xml</document>
            <id>custom-mobile-sync-task-flow</id>
        </task-flow-reference>
    </task-flow-call>
```

9.5.1.2 Updating the Application Task Flow

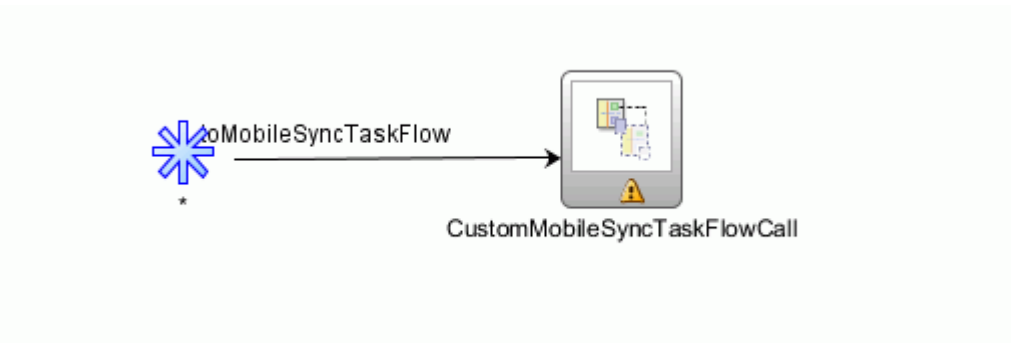
Add a control flow rule to override the `toMobileSyncTaskFlow` outcome as illustrated in Example 9–6.

Example 9–6 The Override Control Flow Rule

```
<control-flow-rule>
      <from-activity-id>*</from-activity-id>
      <control-flow-case>
        <from-outcome>toMobileSyncTaskFlow</from-outcome>
          <to-activity-id>CustomMobileSyncTaskFlowCall</to-activity-id>
      </control-flow-case>
    </control-flow-rule>
```

Figure 9–5 shows the task flow override in the diagrammer.

Figure 9–5 Task Flow Call Pointing to Custom Task Flow



9.6 Setting Up ADF Mobile Transaction Replay Service

You must set up and configure ADF Mobile transaction replay service, a server-side component that is a mechanism for replaying server-side actions that are unavailable to a client when it is offline. It parses specially formatted XML records that contain information about what method to call on the server and what parameters to pass that method. For instructions on setting up and configuring transaction replay service, see *Installation Guide for ADF Mobile Transaction Replay Service*.

Transaction replay service allows the changes made on a mobile device to be processed with the full validation logic and downstream trigger behavior of changes made on the server or online client. After a user makes a change on the ADF Mobile client application, ADF Mobile client automatically makes a transaction replay service Replay Record corresponding to the ADF BC changes made by the user. The Replay Item is brought to the server, either through the Mobile Server's synchronization or through the transaction replay service web service. Transaction replay service processes the Replay Item, invoking the server-side business logic corresponding to the changes made on the client.

Setup includes enabling the transaction replay service technology to the base ADF application and configuring the ADF Mobile client application's entity objects to use it as described in Section 5.8, "Enabling ADF Mobile Transaction Replay Service for an ADF Application." You also deploy the transaction replay service components to a web application server. Several transaction replay service-related wizards are available in the ADF Mobile client extension to help with this deployment and configuration.

10

Testing and Debugging ADF Mobile Client Applications

This chapter provides an overview of the process you will take to finalize the development of your ADF Mobile client application.

This chapter includes the following sections:

10.1 Introduction to Testing and Debugging ADF Mobile Client Applications

Before you start any testing and debugging of your ADF Mobile client application, you have to deploy it to one of the following:

- Windows Mobile device
- Windows Mobile device emulator
- BlackBerry smartphone
- BlackBerry smartphone simulator

> **Note:** You cannot run the mobile client application until it is deployed. For more information, see Chapter 8, "Deploying ADF Mobile Client Components."

There are two approaches to testing a mobile client application:

1. Testing on a mobile device or smartphone: 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 or smartphones

on which you wish to test, making device testing impractical. In practice, most developers use a combination both approaches

2. Testing on a mobile device emulator or smartphone simulator: This method usually offers better performance and faster deployment, as well as convenience. However, even though a device emulator or smartphone simulator closely approximates the corresponding physical device, there are might be differences in behavior and limitations on the capabilities that can be emulated.

Typically, a combination of both approaches yields the best results.

10.2 Testing ADF Mobile Client Applications

To test and debug a mobile client application, you generally take the following steps:

1. Test the application's logic and page flows.
2. Make changes to the application as necessary.
3. Reconnect the mobile device, smartphone or simulator, then deploy and run the application for further testing.

For more information, see Section 10.3, "Debugging ADF Mobile Client Applications for Windows Mobile Platform" and Section 10.4, "Debugging ADF Mobile Client Applications for BlackBerry Platform."

10.3 Debugging ADF Mobile Client Applications for Windows Mobile Platform

You can debug your mobile client application on either a Windows Mobile device or an emulator. 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.

Note: In the current release, you can only debug your Java code. Debugging of EL expressions or other declarative elements is not supported.

10.3.1 How to Configure a Window Mobile Device or Emulator for Debugging

Prior to debugging a mobile client application, you have to configure a mobile device or emulator by performing the following:

- Increasing the Internal Storage Capacity of the Device or Emulator
- Configuring the Device or Emulator for Network Access

10.3.1.1 Increasing the Internal Storage Capacity of the Device or Emulator

You can increase the internal storage capacity of a mobile device or emulator from the command line. For example, the following command line creates a Windows Mobile 6.0 Professional emulator instance with significantly increased RAM and storage capacity:

```
"C:\Program Files\Microsoft Device
Emulator\1.0\DeviceEmulator.exe" "C:\program files\Windows
Mobile 6 SDK\PocketPC\Deviceemulation\0409\PPC_USA.bin"
```

```
/defaultsave /memsize 256 /skin "C:\program files\Windows Mobile
6 SDK\PocketPC\Deviceemulation\Pocket_pc\Pocket_PC.xml" /p
```

The following parameters are essential in achieving the desired effect:

- `/defaultsave` causes a new `.DESS` file to be written to `C:\Documents and Settings\<user>\Application Data\Microsoft\Device Emulator`. Once written, this file can be copied anywhere, and then double-clicked to start an instance of this preconfigured emulator.
- `/memsize 256` causes the emulator to be created with the maximum amount of available memory. Once the `.DESS` file is created, this option is locked to the save-state and cannot be changed.

10.3.1.2 Configuring the Device or Emulator for Network Access

Since debugging of a Java application on Windows Mobile platform occurs over the standard JDWP protocol, which uses TCP sockets for communication, it is necessary for the development computer to establish a TCP connection to the Windows Mobile device or emulator.

When a Windows Mobile device is connected via ActiveSync, it is assigned an IP address of 169.254.2.1, and it can see the host computer at 169.254.2.2.

> **Note:** If a connected device is not addressable in this manner, ensure that the **Enable Advanced Network Functionality** setting is selected. This setting is accessible from **Settings > Connections > USB to PC**.

When a Windows Mobile device emulator is connected through ActiveSync, one of the following happens:

- Windows Mobile 6.0 and later emulators maintain the IP address that the emulator was assigned previously.
- Windows Mobile 5.0 emulators are assigned the non-routable address of 192.168.55.101.

To debug your application, you may find it necessary to uncradle the emulator in order to disable ActiveSync.

10.3.2 How to Deploy the Application to the Window Mobile Device or Emulator for Debugging

To deploy the mobile client application to a Windows Mobile device or emulator for debugging, you start with establishing a connection between you development computer and the device or emulator. For more information, see Section 2.6.3, "How to Connect the Mobile Device or Emulator."

If, at any point, you find that you cannot ping the emulator from your development computer, perform the following steps:

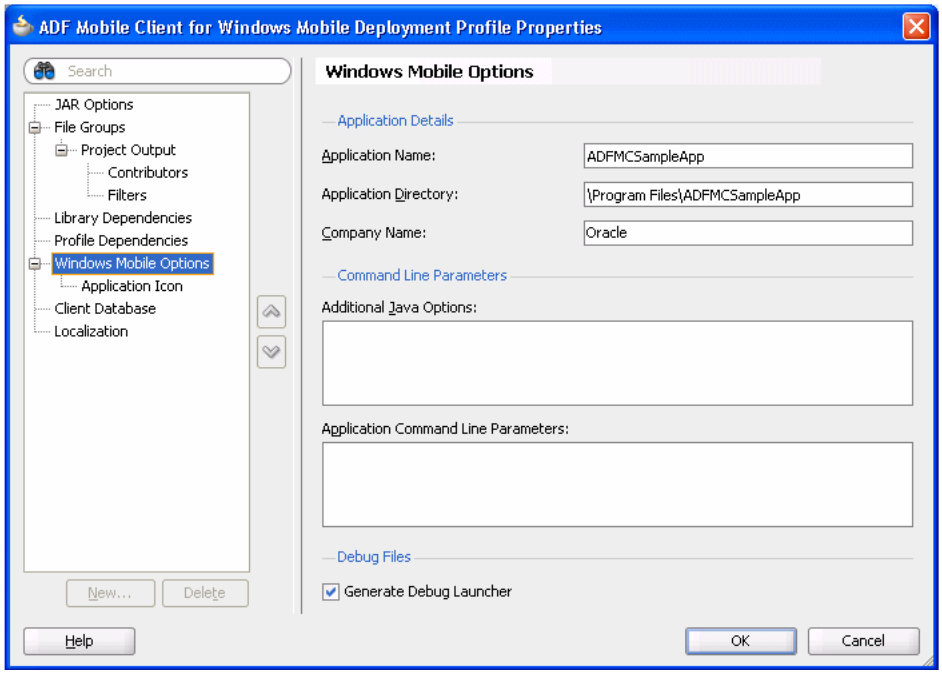
- Start Device Emulator Manager. This functionality is typically accessible through the execution of the `dvcemumanager.exe` file located in the `C:\Program Files\Microsoft Device Emulator\1.0` directory.
- Click **Refresh**, and then scroll through the list of emulators to find one with a green arrow icon beside it.
- Right-click this entry and select **Uncradle**.

Note: You must recradle the emulator when you are ready to redeploy your application from JDeveloper.

To deploy and install your application to the device or emulator:

1. In JDeveloper, create a Windows Mobile deployment profile for your application:
   - In Application Navigator, right-click the application name, and then select **Deploy > New Deployment Profile**.
   - In the **New Gallery** dialog, select **ADF Mobile Client for Windows Mobile** to create your deployment profile.
   - Accept all the default settings of subsequent screens by clicking **OK** until you reach the **ADF Mobile Client for Window Mobile Deployment Profile Properties** screen and make a selection of **Windows Mobile Options** from the tree on the left.
   - On the **ADF Mobile Client for Window Mobile Deployment Profile Properties > Windows Mobile Options** screen that Figure 10–1 shows, select **Generate Debug Launcher**, and then click **OK** (see Section 10.3.3, "What Happens When You Choose to Generate the Debug Launcher").

Figure 10–1 Generating Debug Launcher



   - Continue accepting the default settings on subsequent screens by clicking **OK** until the deployment profile is created.
2. In the Application Navigator, right-click the application name, and then select **Deploy > <DEPLOYMENT PROFILE NAME>**.
3. Click **Finish** to complete the deployment.

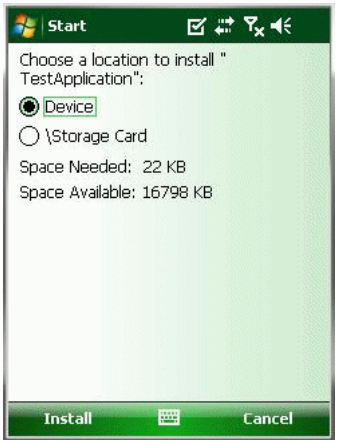
Upon successful deployment, you should see messages in the Deployment - log window in JDeveloper, similar to the following:

```
[10:50:12 AM] Window Mobile package successfully deployed.
```

```
[10:50:12 AM] If installing to a physical device, please refer to device to
complete installation.
[10:50:12 AM] Elapsed time for deployment:  1 second
[10:50:12 AM] ---  Deployment finished.  ---
```

In addition, you should see a screen that Figure 10–2 shows appear on the device or emulator, prompting you to select installation to **Storage Card** or **Device**, and select **Device**.

Figure 10–2 Installing Application to a Device



10.3.3 What Happens When You Choose to Generate the Debug Launcher

When you select **Generate Debug Launcher** in the Windows Mobile Options page (see Figure 10–1, "Generating Debug Launcher"), the ADF Mobile client runtime generates an executable debug file and a corresponding options file in addition to the application launcher and options files described in Section 8.4.2.1, "Setting the JAR File Options." Like those files, the debug files are named for the application and are also included in the CAB file. These files are designated as follows: `Debug_<Application Name>.exe` and `Debug_<Application Name>.options`. If the end user clicks `<Application Name>.exe`, the application launches in regular mode. If the end user clicks `Debug_<Application Name>.exe` the application starts in debug mode.

> **Note:** There must be a debugger attached to the application. Otherwise, the application remains in a wait state for the debugger and will appear to hang.

The contents of the debug options file are identical to those of the regular options file, with the exception of additional command line option which is used to start an application in debug mode:

```
-Xdebug "-agentlib:jdwp=transport=dt_socket,server=y,address=4041"
```

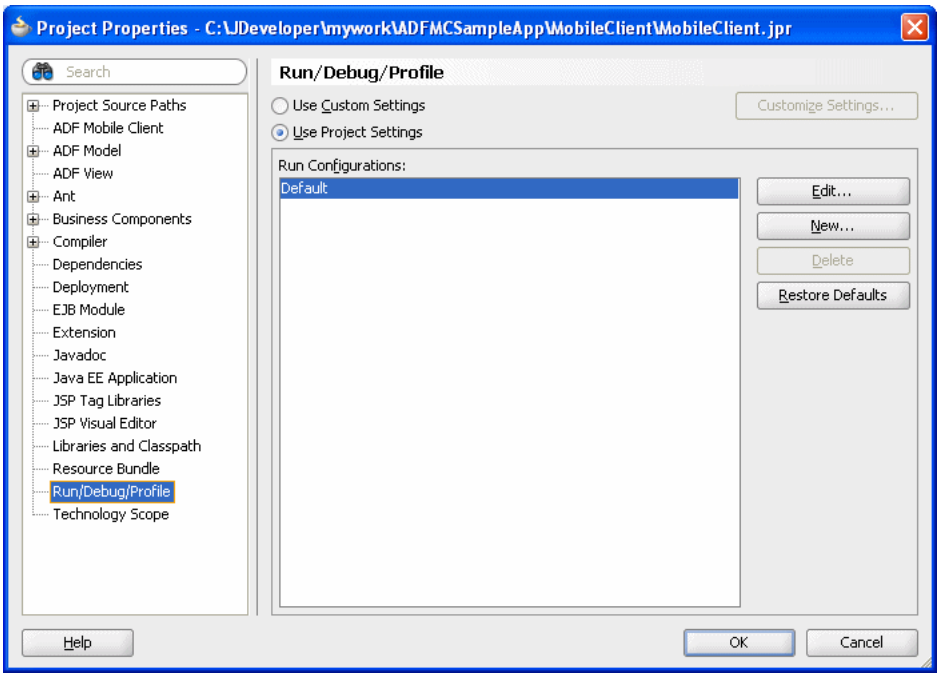
10.3.4 How to Debug the Application on the Windows Mobile Platform

After configuring the mobile device or emulator for debugging, and then deploying your mobile client application, you can start the debugging as follows:

1. In JDeveloper, open the Project Properties by selecting **Application > Project Properties** from the main menu.

2. On the **Project Properties** dialog that Figure 10–3 shows, select **Run/Debug/Profile**

Figure 10–3 Run Configurations Settings for Windows Mobile

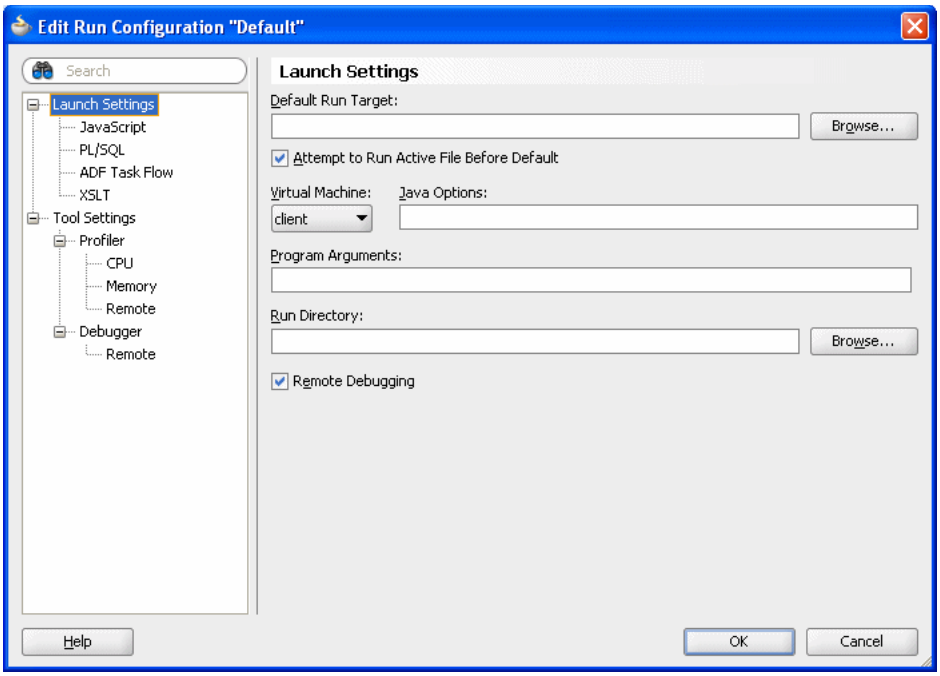


3. Click **Edit** to open the **Edit Run Configuration** dialog and to start editing the default configuration.

> **Note:** If no configurations are listed under **Run Configurations**, add a new configuration by clicking **New**.

4. On the **Edit Run Configuration** dialog that Figure 10–4 shows, select **Launch Settings** from the tree control on the left, and then select **Remote Debugging**.

Figure 10–4 Launch Settings Dialog



5. On the **Edit Run Configuration** dialog that Figure 10–5 shows, select **Debugger > Remote** from the tree on the left, and then enter the IP address of the device or emulator in the **Host** field.

   You can determine the IP address as follows:

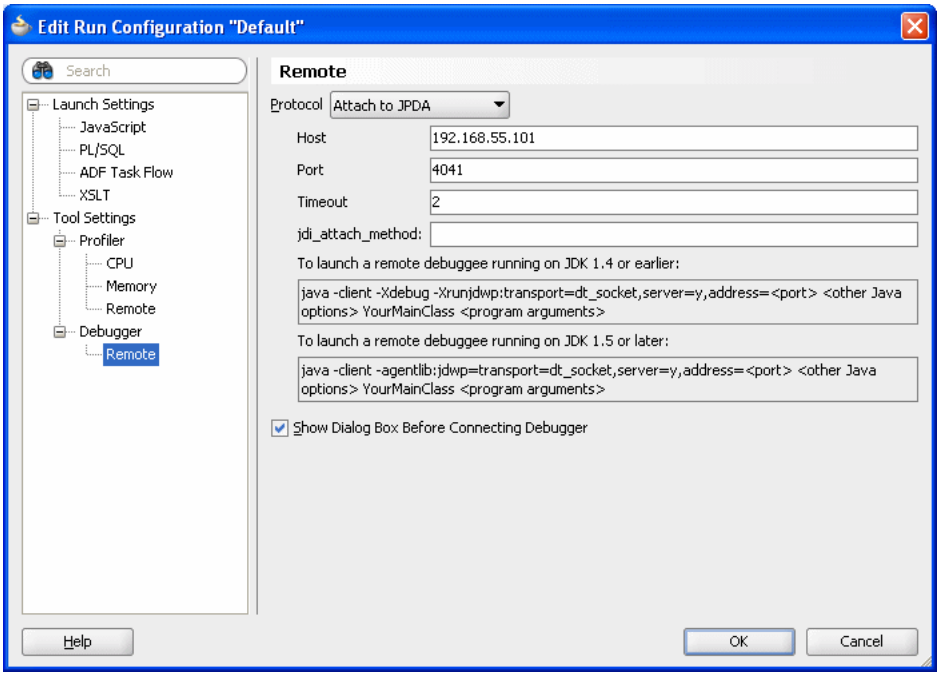
   - On the device or emulator screen, click **Start > Settings**.
   - Select the **Connections** tab, and then **Network Cards**.
   - Select **NE2000 Compatible Ethernet Driver** to display the IP address.

6. Ping the device or emulator's IP address and ensure that the ping succeeds.

   Note: You might have to uncradle the device or emulator in order to ping it, and then recradle when deploying the application from JDeveloper.

7. On the **Remote** screen of the **Edit Run Configuration** dialog, enter the port number 4041 (see Figure 10–5), and then click **OK**.

Figure 10–5 Remote Settings Dialog



8. Open your application's **Deployment Profile Properties**, select **Windows Mobile Options**, and then ensure that **Generate Debug Launcher** is selected (see Figure 10–1 and Section 10.3.3, "What Happens When You Choose to Generate the Debug Launcher").

   ```
   -Xdebug "-agentlib:jdwp=transport=dt_socket,server=y,address=4041"
   ```

9. With the debug settings configured, deploy your application to the mobile device or emulator (see Section 10.3.2, "How to Deploy the Application to the Window Mobile Device or Emulator for Debugging").

10. Using the File Explorer application on the device or emulator, navigate to the application folder, which is typically located in the `\Program Files` directory, and then launch your application by clicking on it. At this point, the application is waiting for a debugger to attach before it will start running.

11. Connect JDeveloper's debugger to the application by selecting the Debug configuration that you created earlier.

12. Start debugging by selecting the edited run configuration from the debug icon drop-down list in JDeveloper.

Upon completion of the preceding steps, you should be able to set breakpoints, suspend execution, and inspect variables, just as you can with an ADF Faces application.

10.3.5 How to Enable Error Logging on a Window Mobile Device or Emulator

You can enable logging of errors while running the mobile client application on a device or emulator as follows:

1. In JDeveloper, in your application project, create a directory called `store` in a location such as `C:\JDeveloper\mywork\<application_name>`

2. In the `store` directory, create another directory called `logs` in which the device or emulator will record error log information and share it with your computer.

   You enable sharing of the `logs` directory as follows:

   a. On the emulator, select **File > Configure**.

   b. In the **Emulator Properties** screen, edit the location of the **Shared folder** field to point to the location on your computer:
   `C:\JDeveloper\mywork\<application_name>\store`

   c. Click **OK**.

You have enabled logging when you run the application.

Note that the `log` file is recreated each time you run the application.

10.4 Debugging ADF Mobile Client Applications for BlackBerry Platform

In the current release, you can debug your mobile client application on a BlackBerry smartphone simulator only. Once you configure the simulator, 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.

You can still deploy and test your application on a physical smartphone, but you will not be able to hit breakpoints and inspect variables, as you can on a simulator.

Note: Currently, you can only debug your Java code. Debugging of EL expressions or other declarative elements is not supported.

10.4.1 How to Configure a BlackBerry Smartphone Simulator for Debugging

Since debugging of Java applications on BlackBerry platform occurs over the standard Java Debug Wire Protocol (JDWP) using TCP sockets as the transport mechanism, both the simulator and the development computer must have network access and be visible to one another on the network.

Prior to debugging a mobile client application, you have to configure the smartphone simulator as follows:

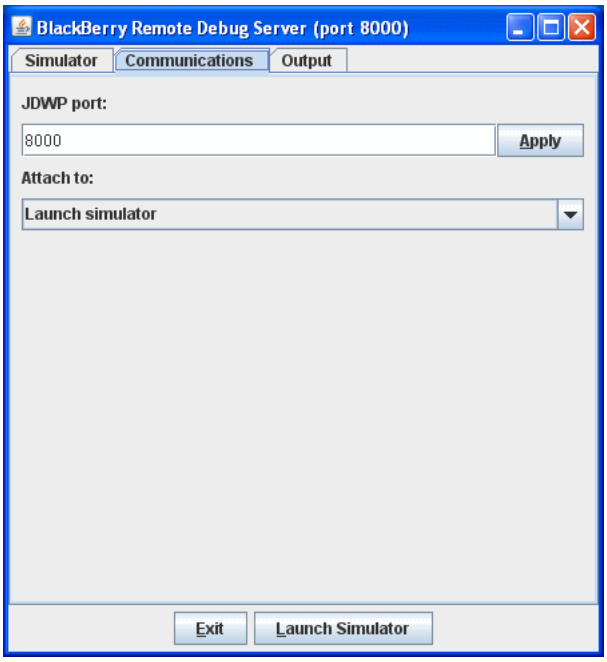
1. In the `<BLACKBERRY JDE PATH>\bin` directory, double-click the `JDWP.bat` file to start the BlackBerry Remote Debug Server.

   Note: JDWP is used for establishing a connection between JDeveloper and the simulator.

   ---

2. In the **BlackBerry Remote Debug Server** dialog, in the **Communications** tab (see Figure 10–6), make note of the JDWP port number. By default, this value is 8000, but you can change it if necessary.

Figure 10–6 BlackBerry Remote Debug Server



3. In the **Simulator** tab, ensure that **Launch simulator when debugger connects** is selected.

> **Note:** If you prefer to start the simulator manually, do not select this option, but instead on the **Communications** tab select **Attach to > Running simulator**. In this case, you have to ensure that a simulator instance has been started prior to attempting a connection.

10.4.2 How to Deploy the Application to the BlackBerry Simulator for Debugging

To deploy the mobile client application to a BlackBerry simulator for debugging, you start with establishing a connection between you development computer and the simulator. For more information, see Section 10.4.1, "How to Configure a BlackBerry Smartphone Simulator for Debugging."

To deploy and install your application to the simulator:

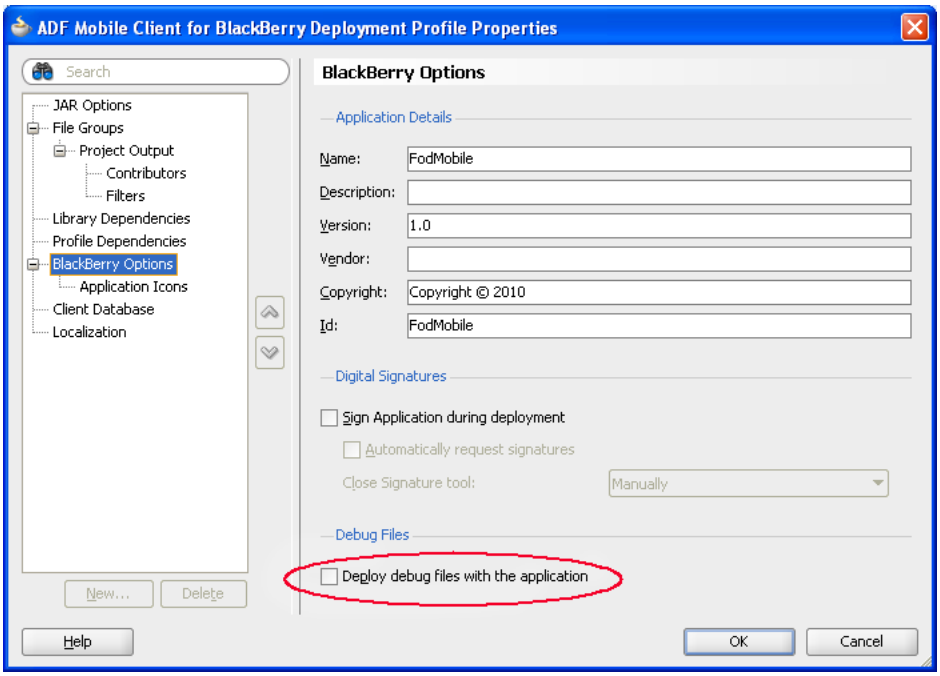
1. In JDeveloper, create a BlackBerry deployment profile for your application:
   - In the Application Navigator, right-click the application name, and then select **Deploy > New Deployment Profile**.
   - In the **New Gallery** dialog, select **ADF Mobile Client for BlackBerry** to create your deployment profile, and then accept all the default settings of subsequent screens by clicking **OK** until the deployment profile is created.
2. In the Application Navigator, right-click the application name, and then select **Deploy > <DEPLOYMENT PROFILE NAME>**.
3. Select the **Deploy Application to Simulator** deployment action.
4. Click **Finish** to complete the deployment.

Upon successful deployment, you should see messages in the Deployment-Log window in JDeveloper, similar to the following:

```
[04:15:55 PM] Deploying Mobile Client application to BlackBerry simulator located
at: C:\Program Files\Research In Motion\BlackBerry JDE 5.0.0\simulator
[04:15:55 PM] Mobile Client application successfully deployed to BlackBerry
simulator. Please start or restart the simulator before running the Mobile Client
application.
[04:15:55 PM] Elapsed time for deployment:  5 seconds
[04:15:55 PM] --  Deployment finished.  ----
```

If you want to hit breakpoints defined in any classes in your application, you can instruct JDeveloper to deploy the application's `.debug` file(s) to the simulator. To do so, select **Deploy debug files with the application** in the **BlackBerry Options** page of the **ADF Mobile Client for BlackBerry Deployment Profile Properties** dialog that shows. For more information, see Section 8.4.1.2, "Setting the BlackBerry Digital Signature Tool Options."

Figure 10–7 Deploying Debug Files With Application

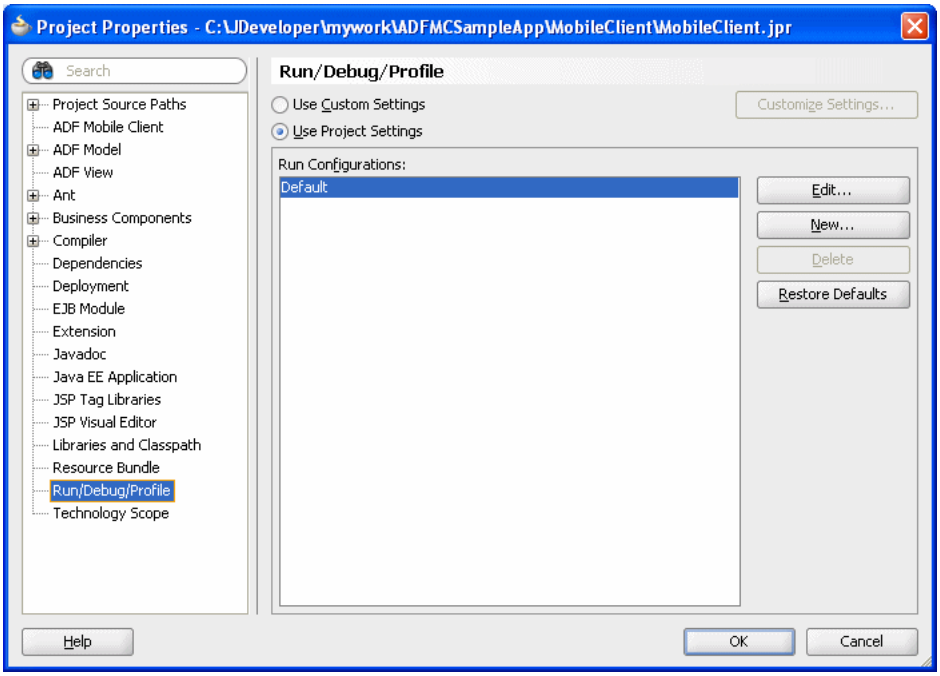


10.4.3 How to Debug the Application on BlackBerry Platform

After configuring the smartphone simulator for debugging, and then deploying your mobile client application, you can start debugging as follows:

1. In JDeveloper, open the Project Properties by selecting **Application > Project Properties** from the main menu.
2. On the **Project Properties** dialog that Figure 10–8 shows, select **Run/Debug/Profile**

Figure 10–8 Run Configurations Settings for BlackBerry

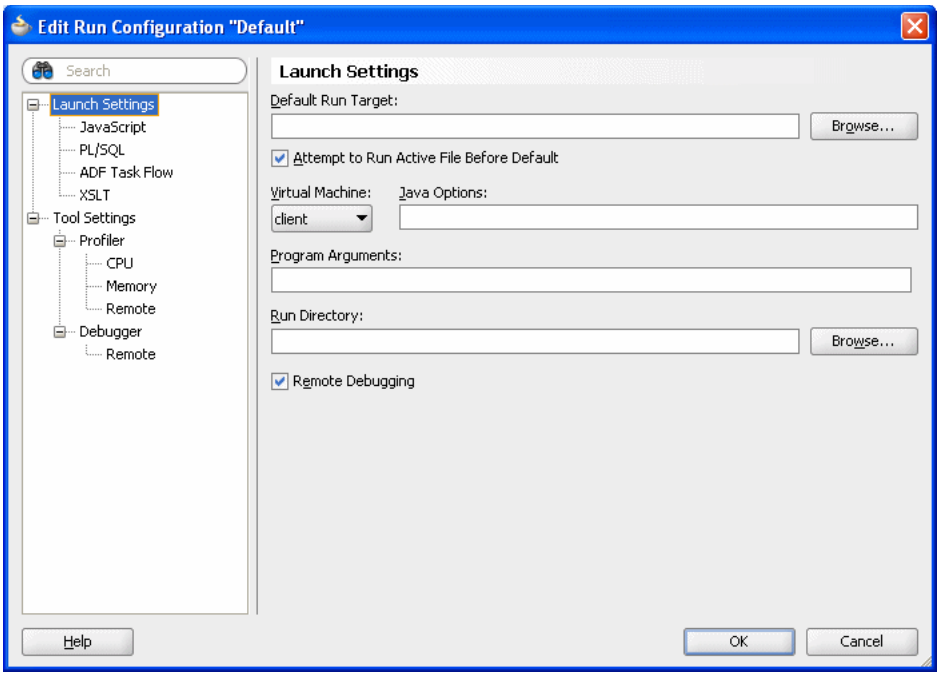


3. Click **Edit** to open the **Edit Run Configuration** dialog and to start editing the default configuration.

> **Note:** If no configurations are listed under **Run Configurations**, add a new configuration by clicking **New**.

4. On the **Edit Run Configuration** dialog that Figure 10–9 shows, select **Launch Settings** from the tree control on the left, and then select **Remote Debugging**.

Figure 10–9 Launch Settings Dialog



5. On the **Edit Run Configuration** dialog that Figure 10–8 shows, select **Debugger > Remote** from the tree on the left, and in the **Host** field enter the IP address of the computer on which the simulator is running, or 127.0.0.1

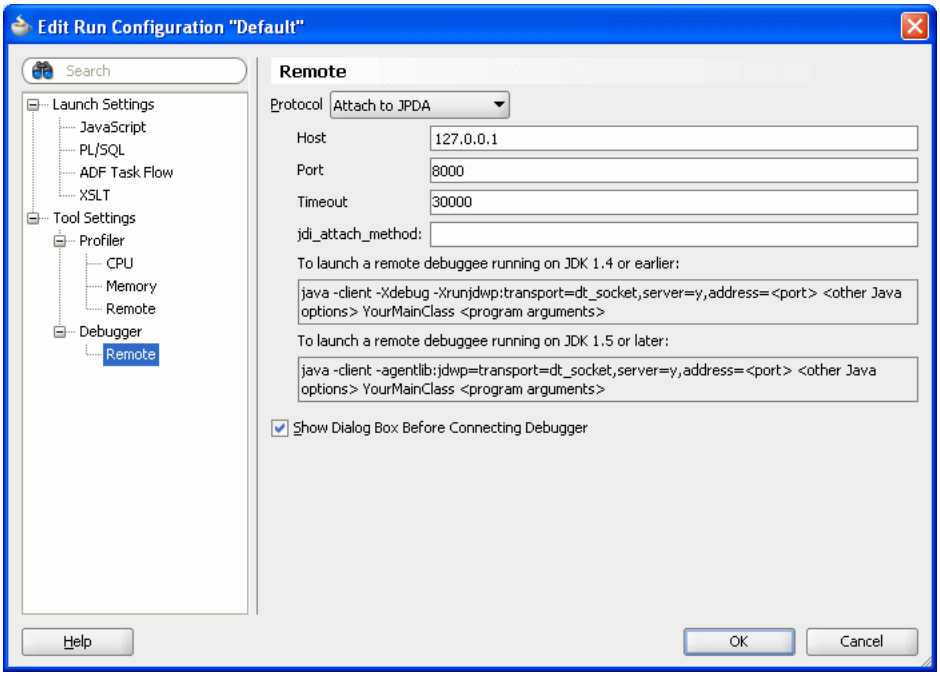
   Specify the same port as you provided in the **Communications** tab of the **BlackBerry Remote Debug Server** dialog (see Figure 10–6), which is 8000 by default.

   In addition, increase the time-out to 30000.

6. On the Remote screen of the **Edit Run Configuration** dialog, enter the same port as you provided in the **Communications** tab of the **BlackBerry Remote Debug Server** dialog (see Figure 10–10).

   In addition, increase the time-out to 30000, and then click **OK**.

Figure 10–10 Remote Settings Dialog



7. In your application, set breakpoints as needed.
8. Select the edited run configuration from the debug icon drop-down list in JDeveloper.
9. Start debugging by running your application in the simulator.

At this point, you should be able to suspend execution and inspect variables, just as you can with an ADF Faces application.

10.4.4 What You May Need to Know About Modifying the Deployment and Run Configurations

At certain times, you may find it necessary to modify the deployment and run configurations of your mobile client application.

You might need to modify your run configuration to debug a mobile device or its emulator, or a smartphone or its simulator that is not connected to the default location or port.

For information on how to modify the run configuration, see the following:

- Figure 10.4.3, "How to Debug the Application on BlackBerry Platform"
- Figure 10–8, "Run Configurations Settings for BlackBerry"

You might need to modify your deployment options for some of the following reasons:

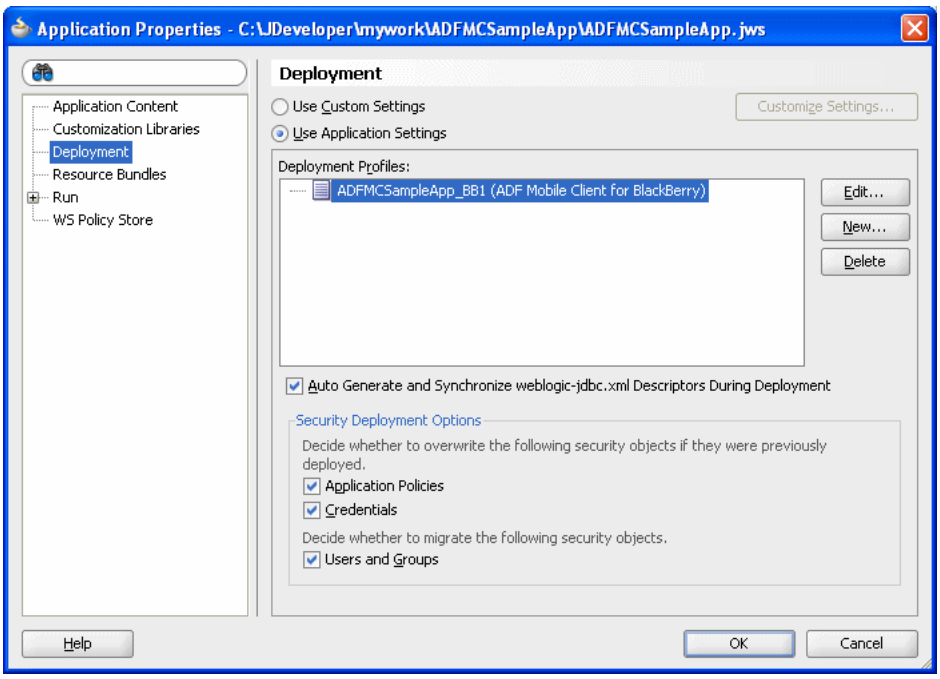
- To deploy the client application to a non-default location on the device.
- To configure the client to access custom code or libraries on the device.

You can change deployment options from the deployment profile as follows:

1. Open the **Application Properties** dialog and select **Deployment** from the tree on the left.

2. On the **Deployment** panel (see Figure 10–11), select the deployment profile that you want modify from the **Deployment Profile**s list, and then click **Edit** to open the **Deployment Profile Properties** dialog.

Figure 10–11 Modifying Deployment Options



3. On the **Deployment Profile Properties** dialog, perform the required modifications, and then click **OK**.

10.4.5 How to Enable Error Logging on a BlackBerry Simulator

You can enable logging of errors while running the mobile client application on the BlackBerry smartphone simulator as follows:

1. In JDeveloper, in your application project, create a directory (called `mystore`, for example) in a location such as `C:\JDeveloper\mywork\<application_name>`.
2. In the `mystore` directory, create another directory called `logs` in which the device or emulator will record error log information and share it with your computer.

   You enable sharing of the `logs` directory as follows:

   a. On the simulator, select **Simulate > Change SD Card**.

   b. In the **SD Card** screen, click **Add Directory**, and then select the location of the `logs` directory.

   c. Click **OK**.

Upon completion of the preceding procedure, when you run the application, the logging will be enabled. Note that the `log` file is recreated each time you run the application.

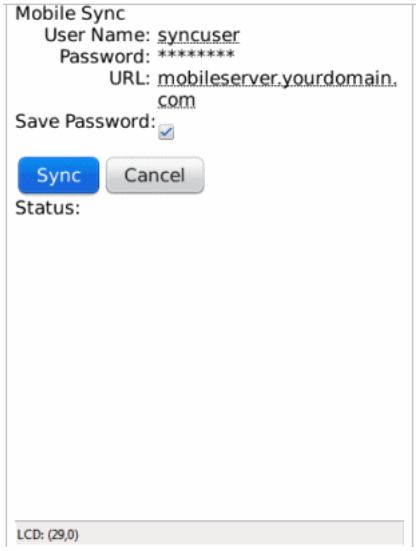
10.5 Testing Synchronization

If your mobile client application synchronizes with a back-end database, you should verify that this data synchronization functions as expected.

After deploying your application to the mobile device or smartphone and starting it for the first time, you may notice that data synchronization is automatically initiated when this data is required for the first time. If you have never synchronized data on your mobile device or smartphone, ADF Mobile client will prompt you to enter the required parameters, as Figure 10–12 and Figure 10–13 show.

Figure 10–12 Synchronization Parameters Screen on Windows Mobile Platform

Figure 10–13 Synchronization Parameters Screen on BlackBerry Platform



After entering the parameters, you start synchronization by clicking **Sync**.

If you previously synchronized data on your mobile device or smartphone and chose to save your password, by default the ADF mobile client attempts to reuse those settings and bypass the prompt. This behavior is useful when each device is owned by a single user. For information on how to modify the default behavior, see Chapter 9, "Synchronizing ADF Mobile Client Data and Transactions."

Once data synchronization has completed successfully, your application should continue running automatically. If an error occurs during synchronization, it must be resolved before your application can proceed. See Appendix C, "Troubleshooting" for information on solutions to common problems related to synchronization.

10.6 Using the ADF Mobile Client Settings Facility

ADF Mobile client makes use of a number of settings that alter the run-time behavior. These settings affect features such as logging and database initialization, and can also include arbitrary application-defined parameters. Settings are defined as key-value pairs in the following ways:

- Runtime defaults: When a value is missing for a given key, the Java code supplies a default value. These have lowest priority.
- `adf-config.xml` and `connections.xml`: If a value is specified in one of these files, it takes precedence over the Java-supplied default.
- Command-line arguments: These settings have the highest priority.

This multilayered approach makes it possible to provide values once for settings that are not expected to change frequently, but still override them easily when necessary.

To add application-specific settings:

1. Define a String constant in your Java code to represent the setting's key. For example:

```
public static String MYSETTING_KEY = "mysetting";
```

2. Read the value of the key by calling `Utility.getProperty`, optionally specifying a default value, as the following example shows:

```
// either
String myValue = Utility.getProperty(MYSETTING_KEY);
// or
String myValue = Utility.getProperty(MYSETTING_KEY, "defaultValue");
```

If `mysetting` is not specified, it will either be `null`, or take the value `defaultValue`, depending on which version of `getProperty` is called. You can change the value by adding the following entry to the `adf-config.xml` or `connections.xml` file:

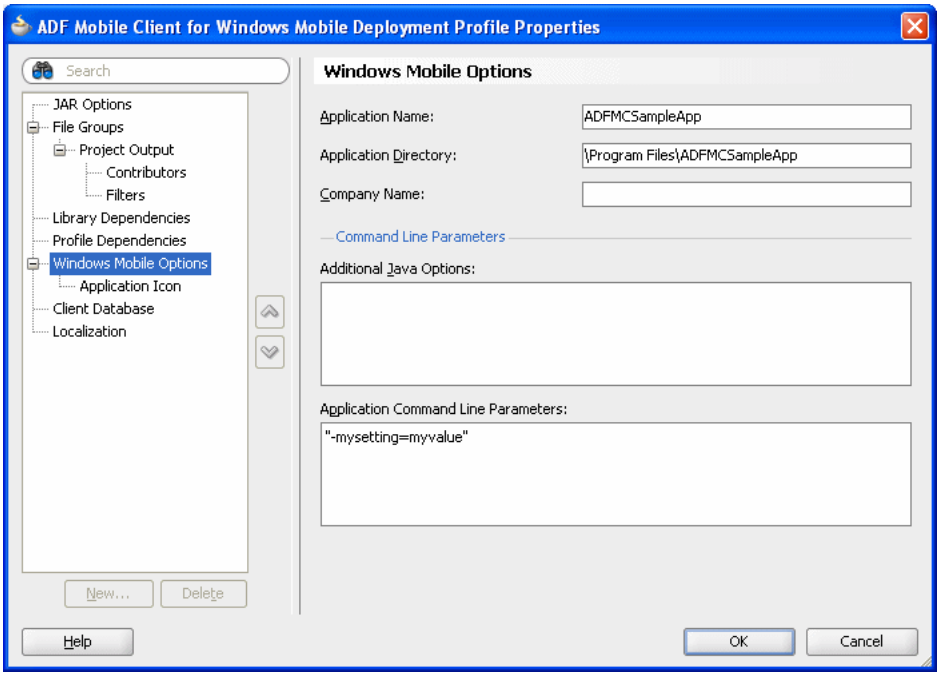
```
<amc:setting name="mysetting" value="myvalue"/>
```

Note: `adf-config.xml` and `connections.xml` are static files that can only be modified at design time.

To pass command-line parameters on Windows Mobile:

1. In JDeveloper, open your application deployment profile's properties.
2. Select **Windows Mobile Options** from the tree on the left, and then enter the command-line parameters in the **Application Command Line Parameters** field, as Figure 10–14 shows.

Figure 10–14 Passing Command-Line Parameters



These parameters are placed in the generated options file after the start-up class name.

To pass a command-line parameter on BlackBerry:

Since `adf-config.xml` and `connections.xml` are static files that can only be modified at design time, if you want to override the value without changing these files and rebuilding your application, you can specify a new value by passing a command-line parameter.

To do so, you must launch your application through another program, using code similar to the following:

```
int handle = CodeModuleManager.getModuleHandle(appName);
ApplicationDescriptor[] browserDescriptors =
                        CodeModuleManager.getApplicationDescriptors(handle);
ApplicationDescriptor descriptor =
                        new ApplicationDescriptor(browserDescriptors[0],
                        null, new String[]{"-mysetting=myvalue"});
ApplicationManager.getApplicationManager().runApplication(descriptor);
```

Table 10–1 lists ADF Mobile client framework settings, which are specific to the ADF Mobile client logging facility that Table 10–4 lists. For more information on the logging facility, see Section 10.6.1, "How to Use the ADF Mobile Client Logging Facility."

Table 10–1 ADF Mobile Client Framework Settings

Setting	Description
`<amc:setting name="root-task-flow" value="<path/to/MobileClient-task-flow.xml>"/>`	Defines the path to the root task flow. Default value: `" "`

Table 10–1 (Cont.) ADF Mobile Client Framework Settings

Setting	Description
`<amc:setting name="app-name" value="NameOfYourApplication"/>`	Sets the name of the application. This is normally prepopulated by JDeveloper. Do not edit this directly; if you need to change it, use the JDeveloper design-time features to do so. Default value: `""`

Table 10–2 lists ADF Mobile client framework database-related settings, which are specific to the ADF Mobile client logging facility that Table 10–4 lists. For more information on the logging facility, see Section 10.6.1, "How to Use the ADF Mobile Client Logging Facility."

Table 10–2 ADF Mobile Client Framework Database Settings

Setting	Description
`<amc:setting name="db-connection" value="connName"/>`	Defines the name of the database connection to use. `connName` is the name of a connection in the `connections.xml` file.
`<amc:setting name="local-init-script" value="<path to sql script>"/>`	If the database needs initialization, it will be initialized by executing the SQL statements in the script referenced by this setting. This script must be included as a resource in the application. Default value: `""`
`<amc:setting name="local-run-init-script" value="never|ifnodb|always"/>`	If set to `"always"`, the application will execute the SQL script in `"init-script"` during application startup to (re)create the application database. If the database exists, all tables will be dropped before the script is executed. If set to `"ifnodb"`, the script will only be run if the database does not exist. If set to `"never"`, the script will not be executed during startup and the database will not be modified in any way. Default value: `"never"`
`<amc:setting name="sync-credentials-mode" value="prompt|provided|lastuser"/>`	Controls how synchronization parameters are provided to the framework. For more information, see Section 9.4, "Enabling Data Synchronization at Application Startup." Default value: `"prompt"`

10.6.1 How to Use the ADF Mobile Client Logging Facility

ADF Mobile client makes extensive use of log messages throughout an application's execution for informational and diagnostic purposes. These messages can be

selectively enabled or disabled by different priority levels across the framework or for specific areas of functionality within the framework for both performance and easier processing of log output.

> **Note:** By default, only messages indicating severe issues are enabled.

You can add log messages to your own code, using logging channels and logging output mechanisms defined within ADF Mobile client, or creating your own.

The ADF Mobile client logging facility is a lightweight adaptation of the Java platform's core logging facility implemented in the `java.util.logging` package.

The following are the key elements used in ADF Mobile client logging:

- Logger: A configurable logical channel with which different groups of log messages are associated. Loggers are named and organized hierarchically, where sub-loggers can inherit or override settings of their parent loggers. A logger directs messages to one or more associated handlers, which control how the messages are output.

  All ADF Mobile client framework classes log their messages through the following named loggers according to their various functional areas within the framework:

  > **Note:** You can configure these loggers declaratively through the ADF Mobile client settings facility as well as add your own.

  - `adfnmc.util`: Logs messages from low-level Utility classes.
  - `adfnmc.model`: Logs messages associated with the ADF Mobile client Model layer.
  - `adfnmc.bindings`: Logs messages associated with the ADF Mobile client data-binding layer.
  - `adfnmc.ui`: Logs messages associated with the ADF Mobile client UI layer.
  - `adfnmc.test`: A channel that can be used for test messages and test output.
  - `adfnmc.profiling`: Logs messages used for outputting performance metrics.
  - `adfnmc`: The parent logger for all ADF Mobile client loggers.
- Handler: A configurable object responsible for exporting messages to various destinations including memory, output streams, consoles, files, and sockets. The subclass of the handler determines the destination type that it supports.

  ADF Mobile client includes the following Handler types:

  - `oracle.adfnmc.java.util.logging.FileHandler`: Writes messages to a file.
  - `oracle.adfnmc.java.util.logging.ConsoleHandler`: Writes messages to the standard output stream.
  - `oracle.adfnmc.java.util.logging.StreamHandler`: Writes messages to an `OutputStream`.
  - `oracle.adfnmc.java.util.logging.DevNullHandler`: Messages sent to this handler are not exported.

- Level: Loggers and handlers can be configured with priority levels to show or suppress messages. All messages are assigned a priority level when they are logged. If the level for a given message is lower than the level assigned to the logger or handler the message is sent to, it will be suppressed.

  The levels in descending order, along with their uses in the different areas of the Mobile client framework are:

  - `SEVERE` (highest value): Logs errors.
  - `WARNING`: Logs warnings; provides information that may indicate a problem in the framework or the application.
  - `INFO`: Logs user actions in the UI area; logs PageDef loads in the bindings area; logs view objects' read and writes, as well as EA/VL movements in the Model area; helps in resolving of objects at load time, as well as checkpoints.
  - `CONFIG`: This level is not supported by ADF Mobile client.
  - `FINE`: Logs EL evaluations in the binding area. Used for loading and parsing, sets and gets.

  Child loggers can override the level threshold for their messages to a higher or lower level than that of their parent loggers. Note, however, that the highest level threshold in a logger-handler chain will be applied. For example, if a logger is configured with a level of `INFO` and has two handlers, configured with levels of `SEVERE` and `FINE`, respectively, only `SEVERE` messages will be output from the first handler, while `SEVERE`, `WARNING`, and `INFO` messages will be output from the second handler. Neither will output `FINE` messages.

- Formatter: Handlers can be configured with different formatting options to apply to messages.

  ADF Mobile client includes the following Formatter types:

  - `oracle.adfnmc.java.util.logging.SimpleFormatter`: Writes brief "human-readable" summaries of log message records.
  - `oracle.adfnmc.java.util.logging.NmcPatternFormatter`: Writes messages modified according to a supplied format string. Note that this formatter does not fully parse format strings. See Table 10–3 for the fixed formats that are supported.

Table 10–3 Formats Supported by NmcPatternFormatter

Pattern String	Message Format
`"[%p - %c - %C{1} - %M] %m%n"`	`"[messageLevel - LoggerName - sourceClassname - sourceMethodName] message"`
`"[%p - %c - %C{1} - %M - %d{ABSOLUTE}] %m%n"`	`"[messageLevel - LoggerName - sourceClassname - sourceMethodName - timeStamp] message"`
`"[%c] %m%n"`	`"[LoggerName] message"`

Table 10–4 ADF Mobile Client Logging Settings

Setting	Description
`<amc:setting name="declare-logger-<loggerIdentifier>" value="<loggerName>"/>`	Declares a named logger. `loggerName` is the actual name of the logger to be used in the logging framework. A logger's name determines its position in the logger hierarchy. A '`.`' is used in the logger name to delimit each level in the hierarchy, from left to right. For example, `"adfnmc"` declares a top-level logger, while `"adfnmc.bindings"` declares a child logger that inherits settings applied to the logger name `"adfnmc"`. `loggerIdentifier` is an identifier used to reference the logger in subsequent setting declarations. loggerIdentifier needs to be the same as loggerName, with any "`.`" delimiters replaced by a "-".
`<amc:setting name="<loggerIdentifier>-level" value="<level>"/>`	Declares the logging level to be used with the named logger.
`<amc:setting name="<loggerIdentifier>-handlers" value="<handlerIdentifiers>"/>`	Declares the named handler or handlers (separated by spaces) to be used with the named logger.
`<amc:setting name="<loggerIdentifier>-useParentHandlers" value="<true or false>"/>`	Declares whether or not the named child logger should use the handler or handlers associated with its parent loggers. Default value: `"true"`.
`<amc:setting name="declare-handler-<handlerIdentifier>" value="<handlerClass>"/>`	Declares a named handler to be used by the logging framework. `handlerIdentifier` is an identifier used to reference the handler in subsequent setting declarations. Cannot contain spaces. `handlerClass` is a fully qualified class name for the handler to use.
`<amc:setting name="<handlerIdentifier>-level" value="<level>"/>`	Declares the logging level to use with the named handler.
`<amc:setting name="<handlerIdentifier>-formatter" value="<formatterClass>"/>`	Declares the formatter class to use with the named handler.
`<amc:setting name="<handlerIdentifier>-formatterPattern" value="<patternString>"/>`	Declares a pattern string to use with a pattern-based formatter. Valid for use with `oracle.adfnmc.java.util.logging.NmcPatternFormatter` only.

Table 10–4 (Cont.) ADF Mobile Client Logging Settings

Setting	Description
`<amc:setting name="<handlerIdentifier>-pattern" value="<path to log file>"/>`	Declares a path to the file to use for log output. Valid for use with `oracle.adfnmc.java.util.logging.FileHandler` only.

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

10.6.2 How to Configure Logging Using the Settings Facility

Most elements in the ADF Mobile client logging facility can be configured declaratively through the ADF Mobile client settings facility. See Table 10–1, " ADF Mobile Client Framework Settings" for a complete list of the settings and their descriptions.

Consider the following examples:

- To change the root ADF Mobile client framework logger level to `FINE`:

```
<amc:setting name="declare-logger-adfnmc" value="adfnmc"/>
<amc:setting name="adfnmc-level" value="FINE"/>
```

- To change the Model layer logger level to `WARNING`:

```
<amc:setting name="declare-logger-adfnmc-model" value="adfnmc.model"/>
<amc:setting name="adfnmc-model-level" value="WARNING"/>
```

- To specify a `FileHandler` and set its level and file path (per platform):

```
<amc:setting name="declare-handler-FILE1"
             value="oracle.adfnmc.java.util.logging.FileHandler"/>
<amc:setting name="FILE1-level" value="FINEST"/>
<amc:setting name="FILE1-pattern" platform="wm"
             value="\Storage Card\Logs\logfile.txt"/>
<amc:setting name="FILE1-pattern" platform="bb"
             value="/SDCard/Logs/logfile.txt"/>
```

- To associate a handler with a logger:

```
<amc:setting name="adfnmc-handlers" value="FILE1"/>

<!-- use a space delimiter to specify multiple handlers -->
<amc:setting name="adfnmc-handlers" value="FILE1 FILE2"/>
```

10.6.3 How to Enable Logging in Java Code

ADF Mobile client includes a utility class, `oracle.adfnmc.java.util.logging.Trace`, which wraps most of the details that you might need to use the logging facility.

For example, the following code logs an informational message:

```
Trace.log(Trace.TEST_LOGNAME, Level.INFO,
          this.getClass(),"someMethod", "my message");
```

The message can also be formatted with parameters, as in the following example:

```
Trace.log(Trace.TEST_LOGNAME, Level.INFO, this.getClass(),
```

```
"someMethod", "my message with param: {0} and another param:
{1}", new Object[] {param0, param1});
```

In the preceding methods, the first parameter is the name of the logger through which to send the message, and the second parameter is the level to apply to the message.

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

11

Working Directly with the Database

This document includes the following sections:

11.1 About Using a Client Database

As stated in Section 8.5, "Specifying the Client Database Location for an Application," you can enable an application to bypass synchronization with the backend server using Oracle Database Lite Mobile Server in favor of a SQLite client database by selecting the **Standalone Database on Client** option in the Create Database Connection dialog of the deployment profiles dialog and then defining the location of database in the **Device Database File** field using one of the platform-appropriate formats listed in Table 11–1.

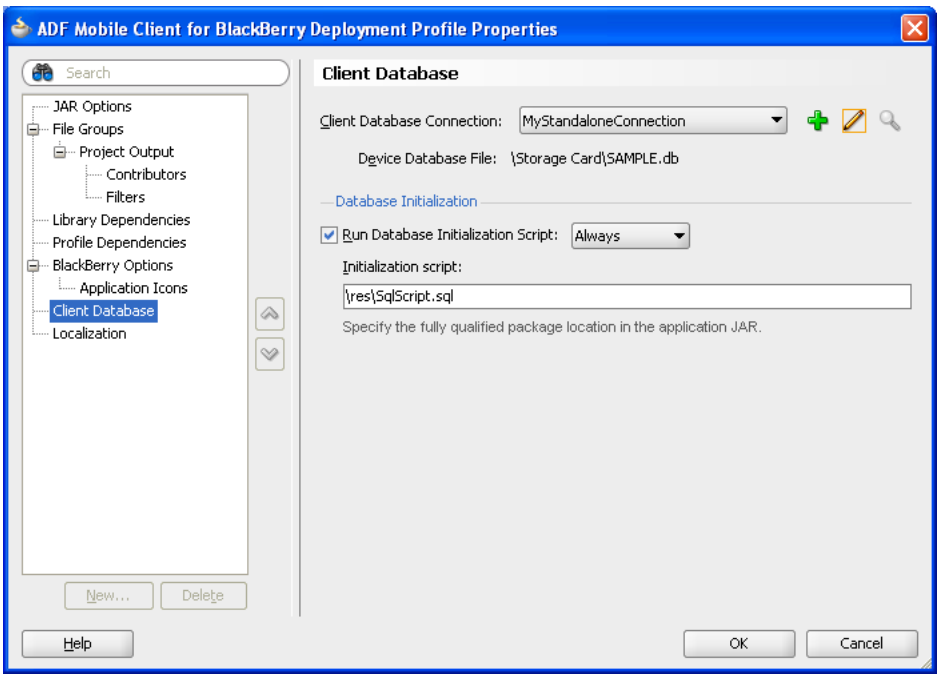
Table 11–1 Fully Qualified Paths to SQLite Databases on BlackBerry and Windows Mobile

Usage Scenario	Path Format in Device Database File Field
Windows Mobile device with a database on an internal file system	`\SAMPLE.db`
Windows Mobile device with a database on an external storage card	`\Storage Card\SAMPLE.db` You can only create SQL databases in flash memory on Windows Mobile emulators, not on the storage card. Windows Mobile devices, however, do not have this limitation.
BlackBerry with a database on an internal file system	`/store/home/user/SAMPLE.db` In general, SQLite databases on BlackBerry smartphones can only be created on an SD card. While some BlackBerry smartphones permit databases on internal flash memory, you should always specify a database that resides on an SD card to ensure maximum compatibility.
BlackBerry, with a database on an external SD card	`/SDCard/SAMPLE.db`

Because SQL databases are binary-compatible across platforms, you can use the same database file on either Windows Mobile or BlackBerry by entering the location in the appropriate format.

Using the **Run Database Initialization Script** options, as shown in Figure 11–1, you can enable the application to use a SQL script to initialize the database each time the application starts.

Figure 11–1 Enabling the Use of a SQL Script



11.2 Enabling Applications to Use SQL Initialization Scripts

If you do not want an application to synchronize data with Oracle Database Lite Mobile Server, you can enable it to use a client database through a SQL initialization script. Although this simple script supports a subset of SQL syntax, it is robust enough to populate a database with some default values.

Example 11–1 illustrates a SQL initialization script. This example shows some of the supported SQL syntax (described in Section 11.2.3, "SQL Syntax") through its use of the `DROP TABLE`, `CREATE TABLE`, and `INSERT` commands and the `NUMBER` and `VARCHAR2` data types. For more information, see Section 11.2.1, "Supported Column Data Type Declarations," and Section 11.2.2, "Literal Format for Date Types."

Example 11–1 A SQL Initialization Script

```
DROP TABLE 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');
```

11.2.1 Supported Column Data Type Declarations

Table 11–2 lists the data types used in column declarations. The types in *italic* font get mapped.

Table 11–2 Data Types

Declared Type	Mapped Type
BIGINT	BIGINT
BINARY	BINARY
BINARY_DOUBLE	DOUBLE
BINARY_FLOAT	FLOAT
BLOB	BLOB
CHAR	CHAR
CLOB	CLOB
DATE	DATE
DECIMAL	DECIMAL
FLOAT	FLOAT
INT	INT
LONG	BIGINT
LONG VARCHAR	LONG VARCHAR
NCHAR	NCHAR
NUMBER	DECIMAL
NUMERIC	DECIMAL
TEXT	VARCHAR
TIME	TIME
TIMESTAMP	TIMESTAMP
VARCHAR	VARCHAR

11.2.2 Literal Format for Date Types

Table 11–3 lists the literal formats for date types that are allowed in the SQL script.

Table 11–3 Literal Formats for Date Types

Declared Type	Allowed Format
DATE	'yyyy-mm-dd'
TIME	'hh:mm:ss'

Table 11–3 (Cont.) Literal Formats for Date Types

Declared Type	Allowed Format
TIMESTAMP	'yyyy-mm-dd hh:mm:ss.fffffffff'

11.2.3 SQL Syntax

The SQL script supports a subset of the SQL data types and commands.

Declaring Data Types

Example 11–2 shows the syntax for declaring data types.

Example 11–2 Declaring Data Types

```
CHAR( n ) | NCHAR( n ) | VARCHAR( n ) | LONG VARCHAR( n ) | INT |
DECIMAL( p,s ) | FLOAT | DOUBLE | DATE | TIME | TIMESTAMP| BLOB | CLOB | BINARY
```

Using Commands

Example 11–3 lists the supported commands.

Table 11–4 Supported Commands

Command	Type	Description	BNF Notation
`DROP TABLE`	DDL	Removes existing objects from the database.	`DROP TABLE name...`
`CREATE TABLE`	DDL	Creates a new table.	`CREATE TABLE table ( column type [ NULL | NOT NULL ] [ UNIQUE ] [ DEFAULT value ] [column_constraint_clause | PRIMARY KEY ] [, ... ] ] [, ...] )`
`INSERT`	DML	Inserts new rows into table.	`INSERT INTO table ( column [, ...] ) VALUES ( expression [, ...] )`

11.2.4 Inserting Multiple Rows into a Table

Example 11–3 shows adding rows to a table one at a time, using separate `INSERT` statements. Alternatively, you can use a single `INSERT` statement in the following form to add multiple rows into the table, as illustrated in Example 11–4:

```
INSERT INTO table (column1, column2,...) VALUES(?,?,?,?);
```

Example 11–3 Inserting Rows into a Table

```
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES ( 100,
'Steven', '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');
```

Example 11–3 shows how a single `INSERT` statement adds the same rows to the PERSONS table as did the separate statements used in Example 11–4.

Example 11–4 Inserting Multiple Rows into a Table

```
INSERT INTO PERSONS (PERSON_ID, FIRST_NAME, LAST_NAME, EMAIL) VALUES(?,?,?,?);
{
100, 'Steven', 'King', 'steven@king.net'
101, 'Neena', 'Kochhar', 'neena@kochhar.net'
102, 'Lex', 'De Haan', 'lex@dehaan.net'
103, 'Alexander', 'Hunold', 'alexander@hunold.net'
104, 'Bruce', 'Ernst', 'bruce@ernst.net'
};
```

11.2.5 Commit Handling

`Commit` statements are ignored when encountered. Each statement is committed as it is read from the SQL script.

11.3 Adding the SQL Script as a Resource to the ADF Mobile Client Application

After you write the script, you add it as a resource to the ADF Mobile client application by selecting the required Run database initialization script option and entering its location in the **Initialization script** field in the Client Database page. For more information, see Section 8.5, "Specifying the Client Database Location for an Application."

12

Using Web Services in ADF Mobile Client Applications

This chapter describes how to integrate a third-party Web service into ADF Mobile client applications.

This chapter includes the following sections:

- Section 12.1, "Introduction to Web Services in ADF Mobile Client Applications"
- Section 12.2, "Creating and Using Web Service Data Controls"
- Section 12.3, "Securing Web Service Data Controls"

12.1 Introduction to Web Services in ADF Mobile Client Applications

Web services let you expose business functionality irrespective of the platform or language of the originating application. For more information, see "Introduction to Web Services in Fusion Web Applications" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

12.2 Creating and Using Web Service Data Controls

The most common way of using Web services in an application developed with Oracle ADF Mobile client 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 the mobile client 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*
- "Data Controls in Oracle ADF Fusion Web Applications" appendix in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*
- Section 6.3.2.3, "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*

12.2.1 How to Create a Web Service Data Control

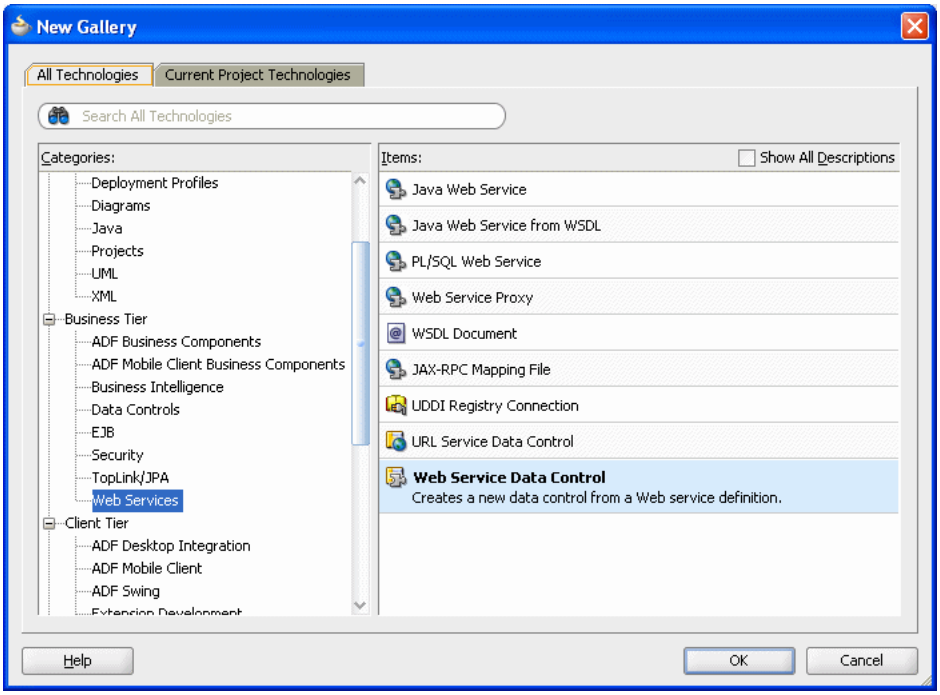
JDeveloper lets you create a data control for an existing Web service using only the Web Services Description Language (WSDL) file for the service. You can either browse to a WSDL file on the local file system, locate one in a Universal Description, Discovery and Integration (UDDI) registry, or enter 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 "Setting Browser Proxy Information" section in *Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework*.

To create a Web service data control:

1. In the Application Navigator, right-click the application name, and then select **New**.
2. In the **New Gallery** dialog, select the **All Technologies** tab, and then expand the **Business Tier** node on the left. Select **Web Services**, and then **Web Service Data Control** from the **Items** list on the right (see Figure 12–1), and click **OK**.

Figure 12–1 Creating a New Web Service



3. Follow the wizard instructions to complete creation of the data control.

Note: The mobile client supports the following encoding styles for both SOAP 1.1 and 1.2 versions:

- Document/literal
- Document/wrapped
- RPC

12.2.2 How to Adjust the Endpoint for a Web Service Data Control

After creating a Web service data control, you can modify the endpoint. This is useful in such cases as when you migrate the application from a test to production environment.

You can modify the endpoint by editing the `connections.xml` file.

Alternatively, to change the endpoint for a Web service data control:

1. In the Application Navigator, select the `.dcx` file for the Web service data control.
2. In the Structure view, right-click the Web service data control and choose **Edit Web Service Connection** from the context menu to open the **Edit Web Service Connection** dialog.
3. In the **Edit Web Service Connection** dialog, make the necessary changes to the endpoint URL and port name.
4. Click **OK**.

12.2.3 How to Create 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 12.2.1, "How to Create a Web Service Data Control").

You modify the connection settings by editing the `connections.xml` file.

12.3 Securing Web Service Data Controls

Web services allow applications to exchange data and information through defined application programming interfaces. The mobile client provides the following means of securing Web service data controls:

- Secure Sockets Layer (SSL): provides secure data transfer over unreliable networks.
- Java Key Store (JKS): provides message-level security for which Web service data controls can be configured.

Note: Currently, the mobile client only supports basic authentication (`BASIC_AUTH`) over HTTP and HTTPS.

For more information, see the following:

- Sections about SSL configuration and managing key stores in *Oracle Database Administrator's Guide*
- Section about configuring policies in *Oracle Fusion Middleware Security and Administrator's Guide for Web Services*

A

Language Abbreviations

This document provides a list of the languages, countries, and their abbreviations used in resource bundle file names. For more information, see Section 5.3.12, "Working with Resource Bundles" and Section 8.6, "Deploying a Multi-Language ADF Mobile Client Application."

Table A–1 Country Abbreviations in Resource Bundle File Names

Country Abbreviation	Country Name
AD, AND	Andorra, Principality of
AE, ARE	United Arab Emirates
AF, AFG	Afghanistan
AG, ATG	Antigua and Barbuda
AI, AIA	Anguilla
AL, ALB	Albania, People's Socialist Republic of
AM, ARM	Armenia
AN, ANT	Netherlands Antilles
AO, AGO	Angola, Republic of
AQ, ATA	Antarctica (the territory South of 60 deg S)
AR, ARG	Argentina, Argentine Republic
AS, ASM	American Samoa
AT, AUT	Austria, Republic of
AU, AUS	Australia, Commonwealth of
AW, ABW	Aruba
AX, ALA	Aaland Islands
AZ, AZE	Azerbaijan, Republic of
BA, BIH	Bosnia and Herzegovina
BB, BRB	Barbados
BD, BGD	Bangladesh, People's Republic of
BE, BEL	Belgium, Kingdom of
BF, BFA	Burkina Faso
BG, BGR	Bulgaria, People's Republic of
BH, BHR	Bahrain, Kingdom of

Table A–1 (Cont.) Country Abbreviations in Resource Bundle File Names

Country Abbreviation	Country Name
BI, BDI	Burundi, Republic of
BJ, BEN	Benin, People's Republic of
BL, BLM	Saint Barth, Saint Barthelemy
BM, BMU	Bermuda
BN, BRN	Brunei Darussalam
BO, BOL	Bolivia, Republic of
BR, BRA	Brazil, Federative Republic of
BS, BHS	Bahamas, Commonwealth of the
BT, BTN	Bhutan, Kingdom of
BV, BVT	Bouvet Island (Bouvetoya)
BW, BWA	Botswana, Republic of
BY, BLR	Belarus
BZ, BLZ	Belize
CA, CAN	Canada
CC, CCK	Cocos (Keeling) Islands
CD, COD	Congo, Democratic Republic of
CF, CAF	Central African Republic
CG, COG	Congo, People's Republic of
CH, CHE	Switzerland, Swiss Confederation
CI, CIV	Cote D'Ivoire, Ivory Coast, Republic of the
CK, COK	Cook Islands
CL, CHL	Chile, Republic of
CM, CMR	Cameroon, United Republic of
CN, CHN	China, People's Republic of
CO, COL	Colombia, Republic of
CR, CRI	Costa Rica, Republic of
CS, SCG	Serbia and Montenegro
CU, CUB	Cuba, Republic of
CV, CPV	Cape Verde, Republic of
CX, CXR	Christmas Island
CY, CYP	Cyprus, Republic of
CZ, CZE	Czech Republic
DE, DEU	Germany
DJ, DJI	Djibouti, Republic of
DK, DNK	Denmark, Kingdom of
DM, DMA	Dominica, Commonwealth of
DO, DOM	Dominican Republic

Table A–1 (Cont.) Country Abbreviations in Resource Bundle File Names

Country Abbreviation	Country Name
DZ, DZA	Algeria, People's Democratic Republic of
EC, ECU	Ecuador, Republic of
EE, EST	Estonia
EG, EGY	Egypt, Arab Republic of
EH, ESH	Western Sahara
ER, ERI	Eritrea
ES, ESP	Spain, Spanish State
ET, ETH	Ethiopia
FI, FIN	Finland, Republic of
FJ, FJI	Fiji, Republic of the Fiji Islands
FK, FLK	Falkland Islands (Malvinas)
FM, FSM	Micronesia, Federated States of
FO, FRO	Faeroe Islands
FR, FRA	France, French Republic
GA, GAB	Gabon, Gabonese Republic
GB, GBR	United Kingdom of Great Britain and North Ireland
GD, GRD	Grenada
GE, GEO	Georgia
GF, GUF	French Guiana
GH, GHA	Ghana, Republic of
GI, GIB	Gibraltar
GL, GRL	Greenland
GM, GMB	Gambia, Republic of the
GN, GIN	Guinea, Revolutionary People's Rep'c of
GP, GLP	Guadaloupe
GQ, GNQ	Equatorial Guinea, Republic of
GR, GRC	Greece, Hellenic Republic
GS, SGS	South Georgia and the South Sandwich Islands
GT, GTM	Guatemala, Republic of
GU, GUM	Guam
GW, GNB	Guinea-Bissau, Republic of
GY, GUY	Guyana, Republic of
HK, HKG	Hong Kong, Special Administrative Region of China
HM, HMD	Heard and McDonald Islands
HN, HND	Honduras, Republic of
HR, HRV	Hrvatska (Croatia)
HT, HTI	Haiti, Republic of

Table A–1 (Cont.) Country Abbreviations in Resource Bundle File Names

Country Abbreviation	Country Name
HU, HUN	Hungary, Hungarian People's Republic
ID, IDN	Indonesia, Republic of
IE, IRL	Ireland
IL, ISR	Israel, State of
IN, IND	India, Republic of
IO, IOT	British Indian Ocean Territory (Chagos Archipelago)
IQ, IRQ	Iraq, Republic of
IR, IRN	Iran, Islamic Republic of
IS, ISL	Iceland, Republic of
IT, ITA	Italy, Italian Republic
JM, JAM	Jamaica
JO, JOR	Jordan, Hashemite Kingdom of
JP, JPN	Japan
KE, KEN	Kenya, Republic of
KG, KGZ	Kyrgyz Republic
KH, KHM	Cambodia, Kingdom of
KI, KIR	Kiribati, Republic of
KM, COM	Comoros, Union of the
KN, KNA	St. Kitts and Nevis
KP, PRK	Korea, Democratic People's Republic of
KR, KOR	Korea, Republic of
KW, KWT	Kuwait, State of
KY, CYM	Cayman Islands
KZ, KAZ	Kazakhstan, Republic of
LA, LAO	Lao People's Democratic Republic
LB, LBN	Lebanon, Lebanese Republic
LC, LCA	St. Lucia
LI, LIE	Liechtenstein, Principality of
LK, LKA	Sri Lanka, Democratic Socialist Republic of
LR, LBR	Liberia, Republic of
LS, LSO	Lesotho, Kingdom of
LT, LTU	Lithuania
LU, LUX	Luxembourg, Grand Duchy of
LV, LVA	Latvia
LY, LBY	Libyan Arab Jamahiriya
MA, MAR	Morocco, Kingdom of
MC, MCO	Monaco, Principality of

Table A–1 (Cont.) Country Abbreviations in Resource Bundle File Names

Country Abbreviation	Country Name
MD, MDA	Moldova, Republic of
ME, MNE	Montenegro, Republic of
MG, MDG	Madagascar, Republic of
MH, MHL	Marshall Islands
MK, MKD	Macedonia, the former Yugoslav Republic of
ML, MLI	Mali, Republic of
MM, MMR	Myanmar
MN, MNG	Mongolia, Mongolian People's Republic
MO, MAC	Macao, Special Administrative Region of China
MP, MNP	Northern Mariana Islands
MQ, MTQ	Martinique
MR, MRT	Mauritania, Islamic Republic of
MS, MSR	Montserrat
MT, MLT	Malta, Republic of
MU, MUS	Mauritius
MV, MDV	Maldives, Republic of
MW, MWI	Malawi, Republic of
MX, MEX	Mexico, United Mexican States
MY, MYS	Malaysia
MZ, MOZ	Mozambique, People's Republic of
NA, NAM	Namibia
NC, NCL	New Caledonia
NE, NER	Niger, Republic of the
NF, NFK	Norfolk Island
NG, NGA	Nigeria, Federal Republic of
NI, NIC	Nicaragua, Republic of
NL, NLD	Netherlands, Kingdom of the
NO, NOR	Norway, Kingdom of
NP, NPL	Nepal, Kingdom of
NR, NRU	Nauru, Republic of
NU, NIU	Niue, Republic of
NZ, NZL	New Zealand
OM, OMN	Oman, Sultanate of
PA, PAN	Panama, Republic of
PE, PER	Peru, Republic of
PF, PYF	French Polynesia
PG, PNG	Papua New Guinea

Table A–1 (Cont.) Country Abbreviations in Resource Bundle File Names

Country Abbreviation	Country Name
PH, PHL	Philippines, Republic of the
PK, PAK	Pakistan, Islamic Republic of
PL, POL	Poland, Polish People's Republic
PM, SPM	St. Pierre and Miquelon
PN, PCN	Pitcairn Island
PR, PRI	Puerto Rico
PS, PSE	Palestinian Territory, Occupied
PT, PRT	Portugal, Portuguese Republic
PW, PLW	Palau
PY, PRY	Paraguay, Republic of
QA, QAT	Qatar, State of
RE, REU	Reunion
RO, ROU	Romania, Socialist Republic of
RS, SRB	Serbia, Republic of
RU, RUS	Russian Federation
RW, RWA	Rwanda, Rwandese Republic
SA, SAU	Saudi Arabia, Kingdom of
SB, SLB	Solomon Islands
SC, SYC	Seychelles, Republic of
SD, SDN	Sudan, Democratic Republic of the
SE, SWE	Sweden, Kingdom of
SG, SGP	Singapore, Republic of
SH, SHN	St. Helena
SI, SVN	Slovenia
SJ, SJM	Svalbard and Jan Mayen Islands
SK, SVK	Slovakia (Slovak Republic)
SL, SLE	Sierra Leone, Republic of
SM, SMR	San Marino, Republic of
SN, SEN	Senegal, Republic of
SO, SOM	Somalia, Somali Republic
SR, SUR	Suriname, Republic of
ST, STP	Sao Tome and Principe, Democratic Republic of
SV, SLV	El Salvador, Republic of
SY, SYR	Syrian Arab Republic
SZ, SWZ	Swaziland, Kingdom of
TC, TCA	Turks and Caicos Islands
TD, TCD	Chad, Republic of

Table A–1 (Cont.) Country Abbreviations in Resource Bundle File Names

Country Abbreviation	Country Name
TF, ATF	French Southern Territories
TG, TGO	Togo, Togolese Republic
TH, THA	Thailand, Kingdom of
TJ, TJK	Tajikistan
TK, TKL	Tokelau (Tokelau Islands)
TL, TLS	Timor-Leste, Democratic Republic of
TM, TKM	Turkmenistan
TN, TUN	Tunisia, Republic of
TO, TON	Tonga, Kingdom of
TR, TUR	Turkey, Republic of
TT, TTO	Trinidad and Tobago, Republic of
TV, TUV	Tuvalu
TW, TWN	Taiwan, Province of China
TZ, TZA	Tanzania, United Republic of
UA, UKR	Ukraine
UG, UGA	Uganda, Republic of
UM, UMI	United States Minor Outlying Islands
US, USA	United States of America
UY, URY	Uruguay, Eastern Republic of
UZ, UZB	Uzbekistan
VA, VAT	Holy See (Vatican City State)
VC, VCT	St. Vincent and the Grenadines
VE, VEN	Venezuela, Bolivarian Republic of
VG, VGB	British Virgin Islands
VI, VIR	U.S . Virgin Islands
VN, VNM	Viet Nam, Socialist Republic of
VU, VUT	Vanuatu
WF, WLF	Wallis and Futuna Islands
WS, WSM	Samoa, Independent State of
YE, YEM	Yemen
YT, MYT	Mayotte
ZA, ZAF	South Africa, Republic of
ZM, ZMB	Zambia, Republic of
ZW, ZWE	Zimbabwe

Table A–2 Languages and Abrreviations Used in Resource Bundle File Names

Language Abbreviation	Language Name
aa, aar	Afar
ab, abk	Abkhazian
ae, ave	Avestan
af, afr	Afrikaans
ak, aka	Akan
am, amh	Amharic
an, arg	Aragonese
ar, ara	Arabic
as, asm	Assamese
av, ava	Avaric
ay, aym	Aymara
az, aze	Azerbaijani
ba, bak	Bashkir
be, bel	Belarusian
bg, bul	Bulgarian
bh, bih	Bihari
bi, bis	Bislama
bm, bam	Bambara
bn, ben	Bengali
bo, bod	Tibetan
br, bre	Breton
bs, bos	Bosnian
ca, cat	Catalan
ce, che	Chechen
ch, cha	Chamorro
co, cos	Corsican
cr, cre	Cree
cs, ces	Czech
cu, chu	Church Slavic
cv, chv	Chuvash
cy, cym	Welsh
da, dan	Danish
de, deu	German
dv, div	Divehi
dz, dzo	Dzongkha
ee, ewe	Ewe
el, ell	Greek

Table A–2 (Cont.) Languages and Abrreviations Used in Resource Bundle File Names

Language Abbreviation	Language Name
en, eng	English
eo, epo	Esperanto
es, spa	Spanish
et, est	Estonian
eu, eus	Basque
fa, fas	Persian
ff, ful	Fulah
fi, fin	Finnish
fj, fij	Fijian
fo, fao	Faroese
fr, fra	French
fy, fry	Frisian
ga, gle	Irish
gd, gla	Scottish Gaelic
gl, glg	Gallegan
gn, grn	Guarani
gu, guj	Gujarati
gv, glv	Manx
ha, hau	Hausa
he, heb	Hebrew
hi, hin	Hindi
ho, hmo	Hiri Motu
hr, hrv	Croatian
ht, hat	Haitian
hu, hun	Hungarian
hy, hye	Armenian
hz, her	Herero
ia, ina	Interlingua
id, ind	Indonesian
ie, ile	Interlingue
ig, ibo	Igbo
ii, iii	Sichuan Yi
ik, ipk	Inupiaq
in, ind	Indonesian (old)
io, ido	Ido
is, isl	Icelandic
it, ita	Italian

Table A–2 (Cont.) Languages and Abrreviations Used in Resource Bundle File Names

Language Abbreviation	Language Name
iu, iku	Inuktitut
iw, heb	Hebrew (old)
ja, jpn	Japanese
ji, yid	Yiddish (old)
jv, jav	Javanese
ka, kat	Georgian
kg, kon	Kongo
ki, kik	Kikuyu
kj, kua	Kwanyama
kk, kaz	Kazakh
kl, kal	Greenlandic
km, khm	Khmer
kn, kan	Kannada
ko, kor	Korean
kr, kau	Kanuri
ks, kas	Kashmiri
ku, kur	Kurdish
kv, kom	Komi
kw, cor	Cornish
ky, kir	Kirghiz
la, lat	Latin
lb, ltz	Luxembourgish
lg, lug	Ganda
li, lim	Limburgish
ln, lin	Lingala
lo, lao	Lao
lt, lit	Lithuanian
lu, lub	Luba-Katanga
lv, lav	Latvian
mg, mlg	Malagasy
mh, mah	Marshallese
mi, mri	Maori
mk, mkd	Macedonian
ml, mal	Malayalam
mn, mon	Mongolian
mo, mol	Moldavian
mr, mar	Marathi

Table A–2 (Cont.) Languages and Abrreviations Used in Resource Bundle File Names

Language Abbreviation	Language Name
ms, msa	Malay
mt, mlt	Maltese
my, mya	Burmese
na, nau	Nauru
nb, nob	Norwegian Bokm?l
nd, nde	North Ndebele
ne, nep	Nepali
ng, ndo	Ndonga
nl, nld	Dutch
nn, nno	Norwegian Nynorsk
no, nor	Norwegian
nr, nbl	South Ndebele
nv, nav	Navajo
ny, nya	Nyanja
oc, oci	Occitan
oj, oji	Ojibwa
om, orm	Oromo
or, ori	Oriya
os, oss	Ossetian
pa, pan	Panjabi
pi, pli	Pali
pl, pol	Polish
ps, pus	Pushto
pt, por	Portuguese
qu, que	Quechua
rm, roh	Raeto-Romance
rn, run	Rundi
ro, ron	Romanian
ru, rus	Russian
rw, kin	Kinyarwanda
sa, san	Sanskrit
sc, srd	Sardinian
sd, snd	Sindhi
se, sme	Northern Sami
sg, sag	Sango
si, sin	Sinhalese
sk, slk	Slovak

Table A–2 (Cont.) Languages and Abrreviations Used in Resource Bundle File Names

Language Abbreviation	Language Name
sl, slv	Slovenian
sm, smo	Samoan
sn, sna	Shona
so, som	Somali
sq, sqi	Albanian
sr, srp	Serbian
ss, ssw	Swati
st, sot	Southern Sotho
su, sun	Sundanese
sv, swe	Swedish
sw, swa	Swahili
ta, tam	Tamil
te, tel	Telugu
tg, tgk	Tajik
th, tha	Thai
ti, tir	Tigrinya
tk, tuk	Turkmen
tl, tgl	Tagalog
tn, tsn	Tswana
to, ton	Tonga
tr, tur	Turkish
ts, tso	Tsonga
tt, tat	Tatar
tw, twi	Twi
ty, tah	Tahitian
ug, uig	Uighur
uk, ukr	Ukrainian
ur, urd	Urdu
uz, uzb	Uzbek
ve, ven	Venda
vi, vie	Vietnamese
vo, vol	Volapuk
wa, wln	Walloon
wo, wol	Wolof
xh, xho	Xhosa
yi, yid	Yiddish
yo, yor	Yoruba

Table A–2 (Cont.) Languages and Abrreviations Used in Resource Bundle File Names

Language Abbreviation	Language Name
za, zha	Zhuang
zh, zho	Chinese
zu, zul	Zulu

B

Advanced Topics

This document includes the following sections:

B.1 Adding Devices in the Page Designer

The `adfnmc-config.xml` file (Example B–1) contains information about all of the supported platforms and form factors within its `platforms` and `form-factors` elements.

> **Tip:** This file is usually located at `C:\Documents and Settings\<USERNAME>\Application Data\JDeveloper\<Build Number>\o.adfnmc.dt\adfnmc-config.xml`.

Using this XML file, you can create new platforms and form factors and map the rendering kits (using in the `rendering-kit-factory-mappings` element). Only update this file with devices supported by the ADF Mobile client runtime. This file enables you to change the list of platforms and form factors without any changes to the JDeveloper mobile client extension.

> **Note:** JDeveloper does not automatically generate `adfmnc-config.xml` when you create an ADF Mobile client project. Instead, you must create this file within the `META-INF` folder of the MobileClient project.

Example B–1 The adfnmc-config.xml File

```
<?xml version="1.0"?>
<adfnmc-config>
  <platforms>
      <platform id="WM" display-value="Windows Mobile" default="true"/>
      <platform id="RIM" display-value="RIM BlackBerry"/>
  </platforms>
  <form-factors>
      <form-factor width="240" height="320" default="true"/>
      <form-factor width="240" height="260"/>
      <form-factor width="240" height="240"/>
      <form-factor width="320" height="240"/>
  </form-factors>
  <rendering-kit-factory-mappings>
      <rendering-kit-factory-mapping idref="WM">
          <rendering-kit-factory-class>
```

```
                                        oracle.adfnmc.imagerenderer.WMRenderingKitF
actory
          </rendering-kit-factory-class>
      </rendering-kit-factory-mapping>
      <rendering-kit-factory-mapping idref="RIM">
         <rendering-kit-factory-class>
                                        oracle.adfnmc.imagerenderer.RIMRenderingKit
Factory
         </rendering-kit-factory-class>
      </rendering-kit-factory-mapping>
  </rendering-kit-factory-mappings>
  <image-repository-mappings>
      <image-repository-mapping idref="WM">
          <image-repository>c:\adf-nmc\controls\images\wm</image-repository>
      </image-repository-mapping >
      <image-repository-mapping idref="RIM">
          <image-repository>c:\adf-nmc\controls\images\rim</image-repository>
      </image-repository-mapping >
  </image-repository-mappings>
</adfnmc-config>
```

C

Troubleshooting

This document includes the following sections:

C.1 Recovering from an mSync Failure

To recover from an mSync failure:

1. Delete the Oracle folder within the Device Memory folder and from the media card.
2. Restart Mobile Server
3. Re-assign, or re-simulate a new SD card.
4. Restart the device and then re-start Mobile Server.

For issues at runtime, such as unable to login or invalid login credentials, perform the following:

1. Verify that the IDE connection with the *mobileadmin* user exists in JDeveloper and is a valid connection.

> **Note:** The Mobile Server user name and credentials are specified in the Application Data Publication page of the deployment profile.

2. Verify that the database exists on the device's SD card.
3. Verify that the SD card is connected and mounted on the device or simulator.

C.2 Errors When Testing Value Binding Queries

Issue

JDeveloper generates the following error when you test a value binding query:

```
SQL Query Error Message: Missing IN or OUT parameter at index:: 1
```

Solution

This error occurred because you used the incorrect binding style. You can only use the JDBC Positional binding style, which substitutes question mark symbols (?) for the bind parameters names. See also Section 5.3.11, "Adding Bind Variables to View Objects."

C.3 Receiving ActiveSync Connection Error Message on Deployment Log

Issue

The following message is written to the JDeveloper deployment log when:

- The ActiveSync process was killed manually through the Windows Task Manager and then restarted and reconnected.
- ActiveSync crashes, hangs, or encounters other problems.

```
Connection with ActiveSync could not be established. Ensure that PC-to-device/
emulator connectivity is working correctly. Restarting JDeveloper may restore
communications. See the ADF Mobile client documentation on Troubleshooting if
problem persists.
```

Solution

Try the following to address this issue:

1. Restart the Windows Mobile emulator, Device Emulator Manager, and ActiveSync.
2. If you are deploying to a Windows Mobile device, disconnect the device and then reconnect it.
3. If the problem persists, restart JDeveloper.
4. Stop and restart ActiveSync using the `asreboot.exe` utility.
5. If the problem continues after you restart JDeveloper, then restart the system.

C.4 Windows Mobile 6.0 Limitations

Because of a limit on the amount of memory available to all dynamic link libraries (DLLs) on Windows Mobile 6.0, some components of ADF Mobile client or Oracle Database Lite may fail to load when other DLLs have consumed that memory. If an application fails to start, or a sync action fails with an error message pertaining to a plugin library, it could indicate that this situation has occurred. You can determine whether this has occurred by looking for the following messages in the application logfile:

- `oseException(-12044): Could not find plugin library "ospsqlite".`
- `oracle.adfnmc.java.sql.SQLException: java.sql.SQLException:no SQLite library found`

- `java.lang.UnsatisfiedLinkError: no eswt-converged.dll in java.library.path`

If one of these errors has occurred, first verify that the ADF Mobile client runtime is installed and that Oracle Lite is installed (if you are using it). This situation is indistinguishable from a missing DLL, and these error messages reflect that.

Once the installation is verified, try to free up the memory available to DLLs by stopping other running applications. Use the Running Programs List to stop them (accessed through **Settings** > **System tab** > **Memory** > **Running Programs tab**) then try the application again.

If the application continues to fail with one of these errors, restart the device and try the application again. If it continues to fail, check the Running Programs list to stop any applications that may have been launched at start-up, and try again.

If it continues to fail, it is possible that applications or utilities are being launched at start-up that do not show themselves in the Running Programs list, and these programs are consuming the memory available to DLLs.

Check the contents of the `\Windows\StartUp` directory. Nearly all Windows Mobile devices will have a shortcut named `poutlook` which is used by the built-in Inbox application. Others may be applications installed by cell phone carriers, or utilities that make use of the features of industrial devices. Do not move or delete any of the files here, but if you recognize the application that any of them belong to, and the application is not necessary, you can try uninstalling the application using **Settings** > **System tab** > **Remove Programs**. The device will restart (or you may restart it manually). Then try running the ADF Mobile client application.

If the failure continues to occur, the remaining work-around requires careful attention. You can move items in the StartUp folder to other directories to prevent their execution at start-up, which could adversely affect applications or device functionality that depends on those programs running, but it may free enough memory for the ADF Mobile client application to run.

You can also upgrade the operating system of the device if an upgrade has been made available by the device manufacturer. Windows Mobile 6.1 and later releases do not have this problem, thanks to a large increase in the amount of memory made available to DLLs.

For more information, see the Microsoft Knowledge Base article at:
`http://support.microsoft.com/kb/326163`

C.5 Sync Agent Issues

If you face issues with the sync agent, a hard reset of the device may be required.

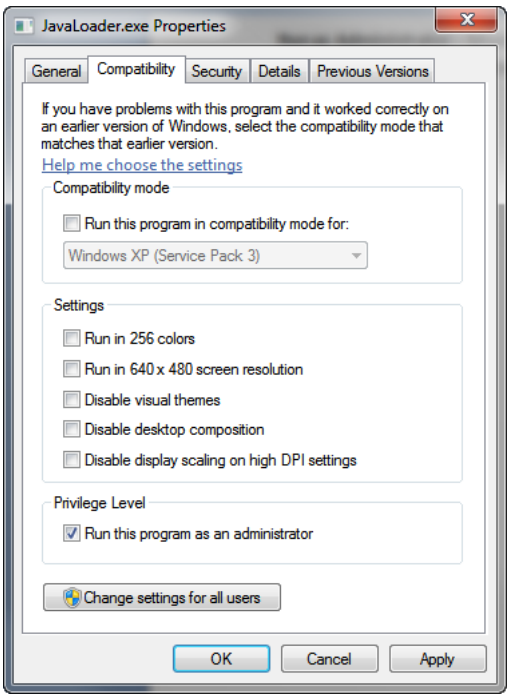
C.6 Windows 7 Workarounds

Some workarounds may be required when Developing ADF Mobile client applications in a Windows 7 development environment may require some workarounds:

- **'\Program Files (x86)'**—On 64-bit editions of Windows 7, 32-bit programs are installed to `\Program Files (x86)` by default rather than the `\Program Files` directory. Most components required for ADF Mobile client application development are 32-bit applications, so ensure you use the correct folder when specifying settings and searching for programs to run.

- **Run as Administrator**— Most common problems can be solved by right-clicking the executable, selecting the **Compatibility** tab (shown in Figure C–1), and checking **Run this program as an administrator.**

Figure C–1 Compatibility Tab



This may be necessary for many development components, including JDeveloper. If the executable is a batch file, it may be necessary to open a command prompt as an administrator and run the batch file directly from the command line.

- **BlackBerry Remote Debug Server**—On 64-bit editions of Windows 7, this application may display an error dialog as shown in Figure C–2 about missing components.

Figure C–2 Missing Components Error

This error occurs when the wrong `javaw` executable appears earlier in your `PATH` than the `javaw` executable from your installed JDK. You can fix this by adjusting your `PATH` so that the `bin` subdirectory of your JDK installation appears first in your PATH. Another solution is to install BlackBerry JDE 6 to a non-standard location, outside of `Program Files (x86)`.

- **BlackBerry Simulator Not Rendering Correctly**—On some configurations, the BlackBerry simulator may not display correctly. Ensure the you are using the latest drivers for your video card. If this does not address the problem, disable or reduce the level of graphics acceleration in the simulator. To do this, from the BlackBerry

simulator menu, select **View** then **Graphics Acceleration**, and choose a different level than the default as shown in Figure C–3.

Figure C–3 Adjusting the Graphics Acceleration Level

- **Installing Oracle Lite Mobile Development Kit**—It may be necessary to set the environment variable `OPATCH_PLATFORM_ID=912` before running the installer.

C.7 SQLite Limitations

SQLite does not allow you to publish an application's data if the application's entity objects have different primary keys but belong to the same table. All entity objects belonging to the same table must have the same primary key (or virtual primary key) structure.

To enable the publication with SQLite:

1. In the Mobile Workbench, remove the existing publication items based on the table in question.
2. In JDeveloper, republish the data.
3. On the device, navigate to the location of the client database (such as `\OraCE\sqlite_db\<name>\<name>.db`) and then delete this client database file.
4. Re-synchronize.

C.8 Font Usage Limitations

When defining properties for ADF Mobile client UI components, consider the following limitations of font usage:

- If the `fontStyle` attribute of a component is set to `italic`, the last character in a text will not be visible. This limitation is present in both Windows Mobile and BlackBerry platforms, and affects all components that display text (for example, Command Button and Output Text).

  Setting the `fontFamily` attribute to one of the following fonts enhances readability of the italicized text:

  - Verdana
  - Century
  - Times New Roman

D

Sample Code

This document shows examples of the `OperationProvider` and `OperationDelegate` which provide an older means of simulating dynamic method bindings. These interfaces have been superseded and are included in this document for backward compatibility. Currently, the ADF Mobile runtime uses `MethodDispatch`, as described Chapter 7, "Extending ADF Mobile Client Applications with Java" instead.

This document includes the following sections:

- Section D.1, "Using the OperationProvider and OperationDelegate Interfaces"

D.1 Using the OperationProvider and OperationDelegate Interfaces

Example D–2 illustrates a sample implementation of an `OperationProvider` and `OperationDelegater` to call custom methods. Example D–1 illustrates the corresponding metadata in the page definition file.

Example D–1 The bindings Metadata

```
</bindings>
       ...
      <methodAction id="Foo" InstanceName="AppModuleDataControl.dataProvider"
                                DataControl="AppModuleDataControl" RequiresUpdateModel="true"
                                Action="invokeMethod" MethodName="Foo"
                                IsViewObjectMethod="false">
         <NamedData NDName="text" NDValue="testString" NDType="java.lang.String"/>
      </methodAction>
</bindings>
```

Example D–2 Using OperationProvider and OperationDelegate

```
package oracle.apps.sales.mobile.view;

import oracle.adfnmc.bindings.DataControl;
import oracle.adfnmc.bindings.OperationDelegate;
import oracle.adfnmc.bindings.dbf.OperationProvider;
import oracle.adfnmc.java.util.List;

import oracle.apps.sales.mobile.model.entities.LeadEOImpl;
import oracle.apps.sales.mobile.uiModel.views.LeadVOImpl;

import oracle.jbo.server.ApplicationModuleImpl;

public class CustomOperationProvider implements OperationProvider
```

```
{
  public CustomOperationProvider()
  {
  }

  public OperationDelegate getCustomOperation(Object module, String instanceName, String
methodName)
  {
    if (instanceName.equals("AppModuleDataControl.dataProvider"))
    {
      if (methodName.equals("testMethod"))
      {
        return new testMethodDelegate();
      }
      else
      {
        // etc...
      }
    }
    else if (instanceName.equals("AppModuleDataControl.LeadVO1"))
    {
      if (methodName.equals("convertLeadToOpty"))
      {
        return new convertLeadToOptyDelegate();
      }
      else
      {
        // etc...
      }
    }
    else if (instanceName.equals("AppModuleDataControl.OptyVO1"))
    {
      // etc...
    }

    return null;
  }

  private static final class convertLeadToOptyDelegate implements OperationDelegate
  {
    public Object execute(Object instance, List params)
    {
      ((LeadVOImpl)instance).convertLeadToOpty();
      return null;
    }
  }

  private static final class testMethodDelegate implements OperationDelegate
  {
    public Object execute(Object instance, List params)
    {
      String arg0 = (String)params.get(0);
      return ((ApplicationModuleImpl)instance).testMethod(arg0);
    }
  }
}
```

Index

A

B

C

D

E

F

G

I

J

L

M

O

P

R

S

T

V

W

Z