Oracle® Mobile Application Framework

Installing Oracle Mobile Application Framework

2.2.2

E70799-01

January 2016

Documentation that describes how to install the Oracle Mobile Application Framework for use with Oracle JDeveloper to create mobile applications that run natively on devices.

ORACLE

Oracle Mobile Application Framework Installing Oracle Mobile Application Framework, 2.2.2

E70799-01

Primary Author: Walter Egan

Contents

Preface

Welcome to *Installing Oracle Mobile Application Framework*.

Audience

This manual is intended for developers who want to install the Oracle Mobile Application Framework for use with Oracle JDeveloper to create mobile applications that run natively on devices.

Related Documents

For more information, see *Developing Mobile Applications with Oracle Mobile Application Framework*.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements (for example, menus and menu items, buttons, tabs, dialog controls), including options that you select.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates language and syntax elements, directory and file names, URLs, text that appears on the screen, or text that you enter.

1

Installing Mobile Application Framework with JDeveloper

This chapter describes how to install JDeveloper and the Mobile Application Framework (MAF) extension for application development.

This chapter includes the following sections:

1.1 Introduction to Installing the MAF Extension with JDeveloper

The first step in starting with MAF application development is to install Oracle JDeveloper and the MAF extension.

In the current release, you must install JDeveloper using JDK 1.7, and then install the MAF extension in JDeveloper specifying JDK 1.8 in the dialog that appears after JDeveloper restarts after installation of the extension. This allows MAF applications to compile with JDK 1.8.

Following installation of the MAF extension in JDeveloper, configure additional development tools for the platforms where you intend to deploy your MAF application. For more information, see Setting Up the Development Environment .

Before you can create a MAF application using the MAF extension in JDeveloper, ensure that you have any third-party software required to develop applications for the platform on which you intend to deploy your MAF application.

Note:

You can deploy the same MAF application to all supported platforms without changing your application's code. You need the third-party software to test, debug, and deploy the MAF application on the target platform.

1.2 Installation Requirements for MAF Applications to be Deployed to the iOS Platform

Before you start creating a MAF application that you are planning to deploy to the iOS platform, ensure that you have the following available:

- A computer running Apple Mac OS X Version 10.9.5 or later.
- Oracle JDeveloper (see Setting Up JDeveloper).
- Oracle JDeveloper extension for MAF (see Installing the MAF Extension in JDeveloper).
- Xcode and iOS SDK (see How to Install Xcode and iOS SDK).
- The most recent version of JDK1.8.
- The most recent version of JDK1.7.

Before you start deploying your application to a development environment (see the "Getting Started with Mobile Application Development" chapter in *Developing Mobile Applications with Oracle Mobile Application Framework*), decide whether you would like to use a mobile device or its simulator: if you are to use a simulator, see How to Set Up an iPhone or iPad Simulator; if your goal is to deploy to a mobile device, ensure that, in addition to the components included in the preceding list, you have the following available:

- Various login credentials. For more information, see the "Deploying Mobile Applications" chapter in *Developing Mobile Applications with Oracle Mobile Application Framework*.
- iOS-powered device. For more information, see How to Set Up an iPhone or iPad.

1.3 Installation Requirements for MAF Applications to be Deployed to the Android Platform

Before you start creating a MAF application that you are planning to deploy to the Android, ensure that you have the following available:

- A computer running one of the following operating systems:
  - Microsoft Windows Vista
  - Microsoft Windows 7
  - Mac OS X
- The most recent version of JDK1.8
- The most recent version of JDK1.7
- Android SDK Manager (see Setting Up Development Tools for the Android Platform)
- Oracle JDeveloper (see Setting Up JDeveloper)

- Oracle JDeveloper extension for MAF (see Installing the MAF Extension in JDeveloper)

Before you start deploying your application to a development environment (see the "Getting Started with Mobile Application Development" chapter in *Developing Mobile Applications with Oracle Mobile Application Framework*), decide whether you would like to use a mobile device or its emulator: if you are to use an emulator, see Setting Up Development Tools for the iOS Platform; if your goal is to deploy to a mobile device, ensure that, in addition to the components included in the preceding list, you have the following available:

- Various login credentials. For more information, see the "Deploying Mobile Applications" chapter in *Developing Mobile Applications with Oracle Mobile Application Framework*.
- Android-powered device. For more information, see Setting Up Development Tools for the Android Platform.

1.4 Setting Up JDeveloper

Oracle JDeveloper and its MAF extension are essential tools used in developing MAF applications.

Before you begin:

- Download and install the latest version of JDK 1.7.

  This version of JDK is required by JDeveloper.

- Download and install the latest version of JDK 1.8.

  This version of JDK is required by the MAF extension.

- Download the 12.1.3.0.0 release of JDeveloper (Studio Edition) available at `http://www.oracle.com/technetwork/developer-tools/jdev/downloads/index.html`.

To install JDeveloper on a computer running the Windows platform:

1. In your file system, navigate to the directory that contains the JDeveloper executable file, then right-click that folder and select CMD Prompt Here As Administrator.
2. Run the following command to explicitly install JDeveloper using the required JDK 1.7:

   ```
   <fully_qualified_path_to_JDK7>\bin\java -jar
   <JDEV_12.1.3_jar>
   ```

For more information, see *Installing Oracle JDeveloper*.

To install JDeveloper on a computer running the Mac OS X platform:

1. Open a Terminal window.
2. Set the `JAVA_HOME` to Java 1.7 by running the following command:

   ```
   export JAVA_HOME=$(/usr/libexec/java_home -v1.7)
   ```

3. Verify that Java 1.7 is used by running the following command:

   ```
   java -version
   ```

4. Using the same Terminal window, install JDeveloper by executing the following:

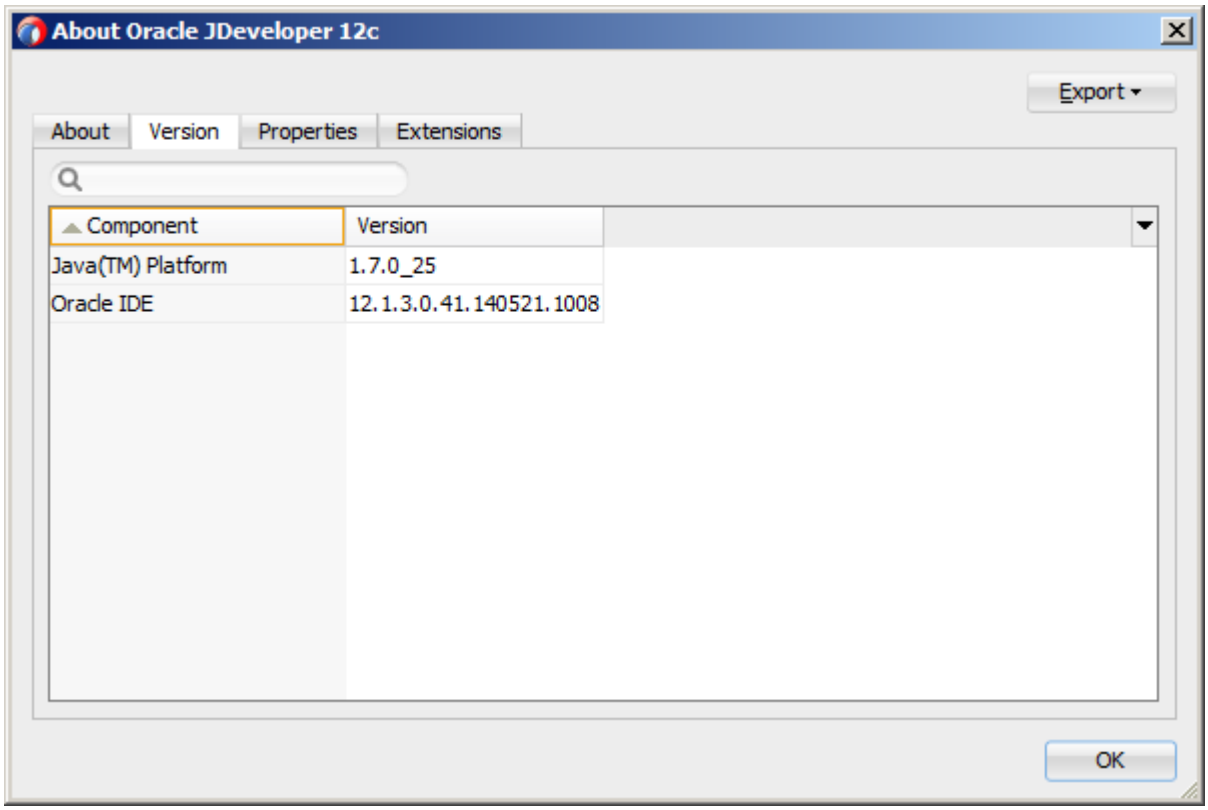
   ```
   java -jar <JDEV_12.1.3_jar>
   ```

For more information, see the section about using Oracle JDeveloper on the Mac OS X platform in *Installing Oracle JDeveloper*.

To verify the installation of JDeveloper:

1. Check the `<JDEV_HOME>\jdev\bin\jdev.conf` file and confirm that the `SetJavaHome` property points to JDK 1.7.
2. Start JDeveloper and select the Studio Developer (All Features) role when prompted.
3. From the main menu, select Help > About > Version and ensure that the Java platform 1.7 is used, as Figure 1-1 shows.

Figure 1-1 Verifying JDK Version



1.5 Installing the MAF Extension in JDeveloper

You download the MAF extension using the Check for Updates menu in JDeveloper.

Once you have installed the MAF extension, you need to configure additional development tools for the platforms where you intend to deploy your MAF application. For more information, see Setting Up the Development Environment .

To download and install the MAF extension:

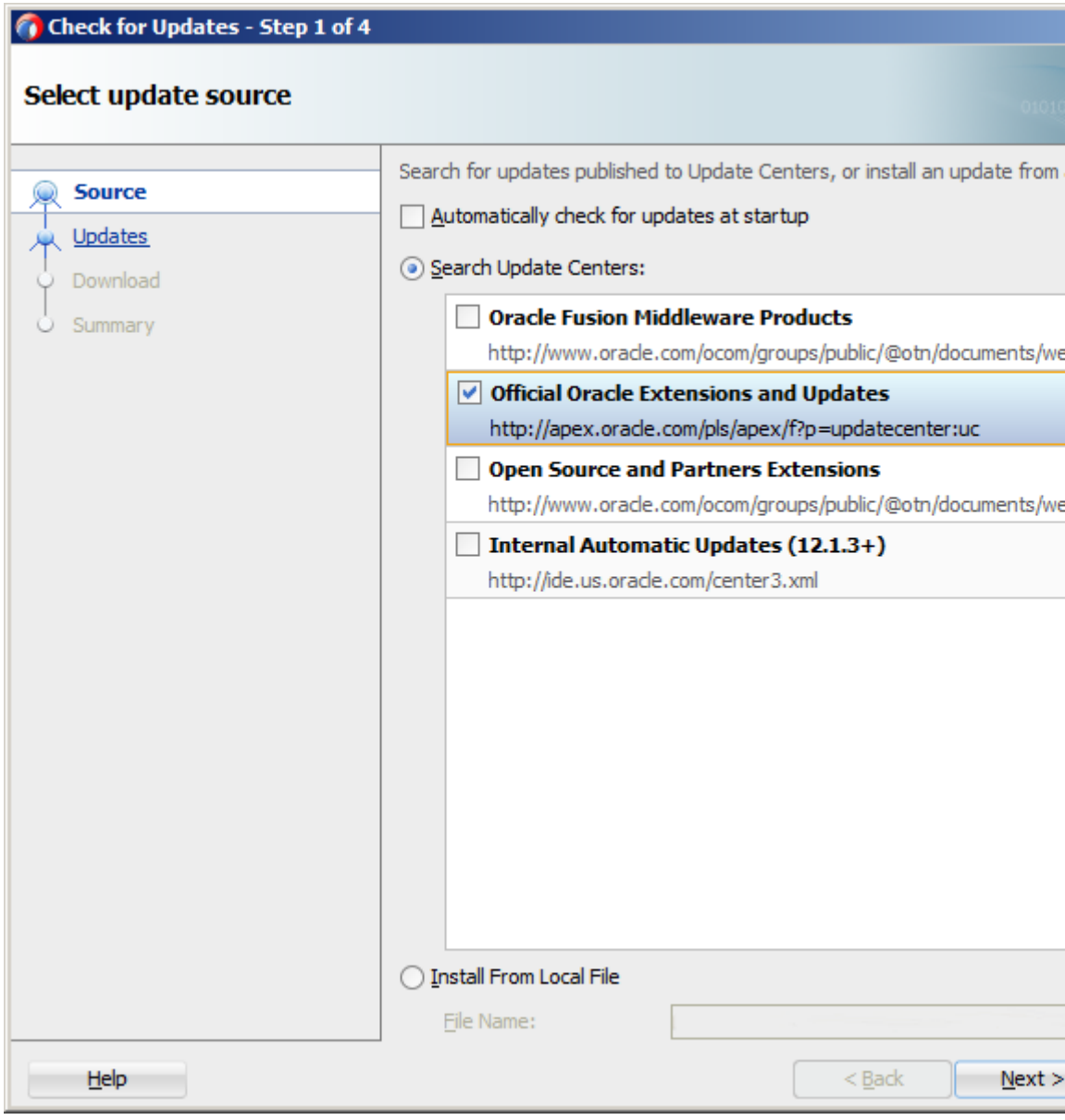
1. In JDeveloper, choose **Help** > **Check for Updates**.

Note:

You might need to configure proxy settings on your development computer: on Windows, select **Tools > Preferences** from the main menu, and then **Web Browser and Proxy** from the tree on the left of the **Preferences** dialog; on Mac OS X, this option is accessed from **JDeveloper > Preferences**.

2. In the **Select update source** page that Figure 1-2 shows, select **Official Oracle Extensions and Updates** under the **Search Update Centers**, and then click **Next**.

Figure 1-2 Checking for Updates in JDeveloper



Alternatively, if network access is not available, you can select the **Install From Local File** option. In this case, you need to point to the MAF extension file that you already downloaded to a directory on your development computer.

3. In the **Select updates to install** dialog, select the **Mobile Application Framework** update.
4. In the **License Agreements** page, shown in Figure 1-3, review *The Oracle Technology Network License Terms for Oracle Mobile*.

Note:

You must comply with all of the license terms and conditions with respect to the Oracle Mobile Application Framework Program available at `http://www.oracle.com/technetwork/indexes/downloads/index.html`.

---

5. Click **I Agree**.

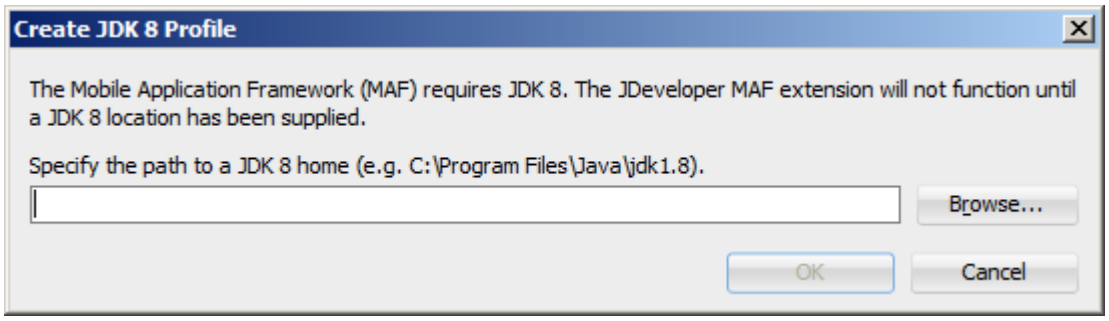
Figure 1-3 Licensing Agreements for Mobile Application Framework Program

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

7. Restart JDeveloper.

8. Use the **Create JDK 8 Profile** dialog that Figure 1-4 shows to specify the path to the directory on your computer that contains JDK 1.8.

Figure 1-4 Creating JDK 8 Profile



Note:

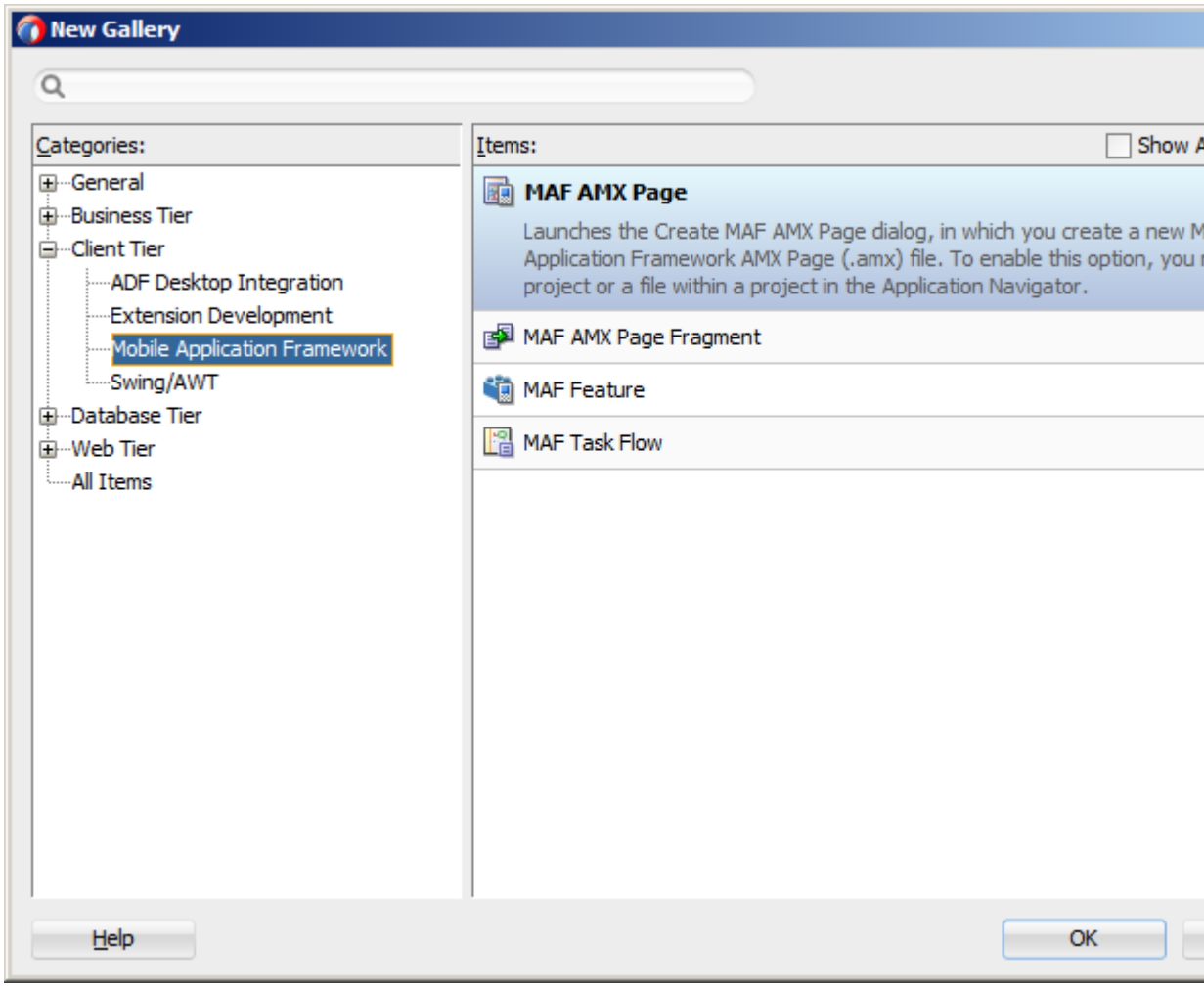
If you specify an invalid directory or directory that does not contain JDK 1.8, an error dialog is displayed.

You do not have to complete the Create JDK 8 Profile dialog the next time you use JDeveloper, unless you reinstall the MAF extension and choose not to preserve JDeveloper's system preferences.

9. Check whether or not MAF has been successfully added to JDeveloper:

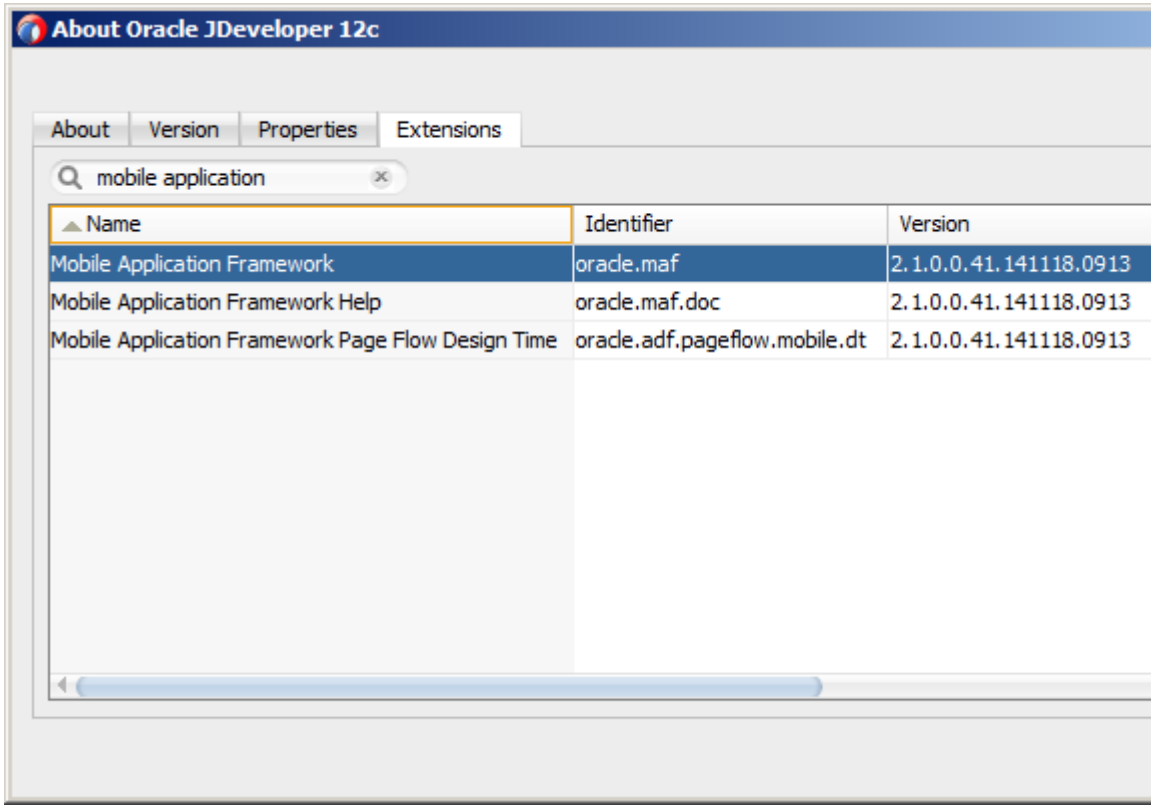
   - Select **File > New > From Gallery** from the main menu to open the **New Gallery** dialog.
   - In the **Categories** tree on the left, expand the **Client Tier** node and make sure it contains **Mobile Application Framework** (see Figure 1-5).

Figure 1-5 Verifying MAF Installation



In addition, verify that you installed the correct version of MAF. To do so, select **Help > About** from the main menu, then select the **Extensions** tab on the About Oracle JDeveloper dialog, and then examine the extension list entries by searching for **Mobile Application Framework**, as Figure 1-6 shows.

Figure 1-6 Verifying MAF Version



In addition to the preceding steps, your development environment must be configured for target platforms and form factors. For more information, see Setting Up the Development Environment .

2

Setting Up the Development Environment

This chapter provides information on setting up and configuring the MAF environment for application development and deployment.

This chapter includes the following sections:

2.1 Introduction to the MAF Development Environment

After you install JDeveloper and the MAF extension, as described in Installing Mobile Application Framework with JDeveloper , you may need to configure the development environment for the platforms to which you intend to deploy your MAF application. In addition, you may need to configure form factors if you intend to test or deploy on a particular mobile device. You may also need to install and configure third-party tools that allow you to package and deploy your MAF application on supported platforms.

For complete list of supported versions of development and runtime tools, see Oracle Mobile Application Framework Certification Matrix by following the Certification Information link on the MAF documentation page at `http://www.oracle.com/technetwork/developer-tools/maf/documentation/`.

2.2 Configuring the Development Environment for Target Platforms

For successful packaging and deployment of your application to platforms supported by MAF, JDeveloper must be provided with such information as the name of the platform and directories on your development computer that are to house the platform-specific tools and data. For convenience, MAF prepopulates JDeveloper Preferences with these settings. Depending on several factors related to the application signing, you may need to edit some of the fields.

Before you begin:

Download and install JDeveloper and the MAF extension, as described in Installing Mobile Application Framework with JDeveloper .

Depending on your target platform, download and configure either the Android SDK (see How to Install the Android SDK) or iOS SDK and Xcode (see How to Install Xcode and iOS SDK).

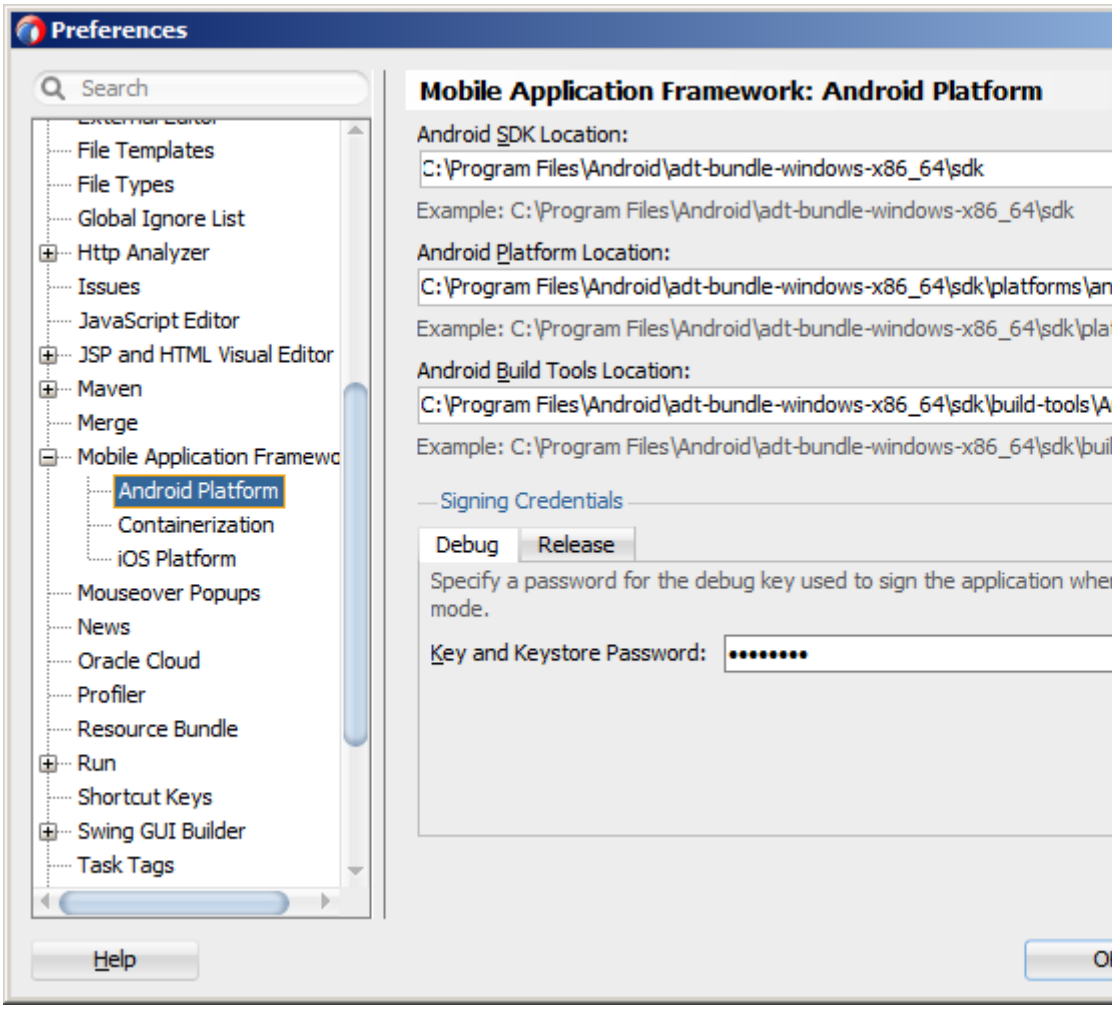
To configure your environment for target platforms:

1. Select **Tools > Preferences** from JDeveloper's main menu to open Preferences.
2. In the **Preferences** dialog, select either **Mobile Application Framework > Android Platform** or **Mobile Application Framework > iOS Platform** from the tree to open a page that contains the path and configuration parameters for the supported platforms, as Figure 2-1 and Figure 2-2 show.

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

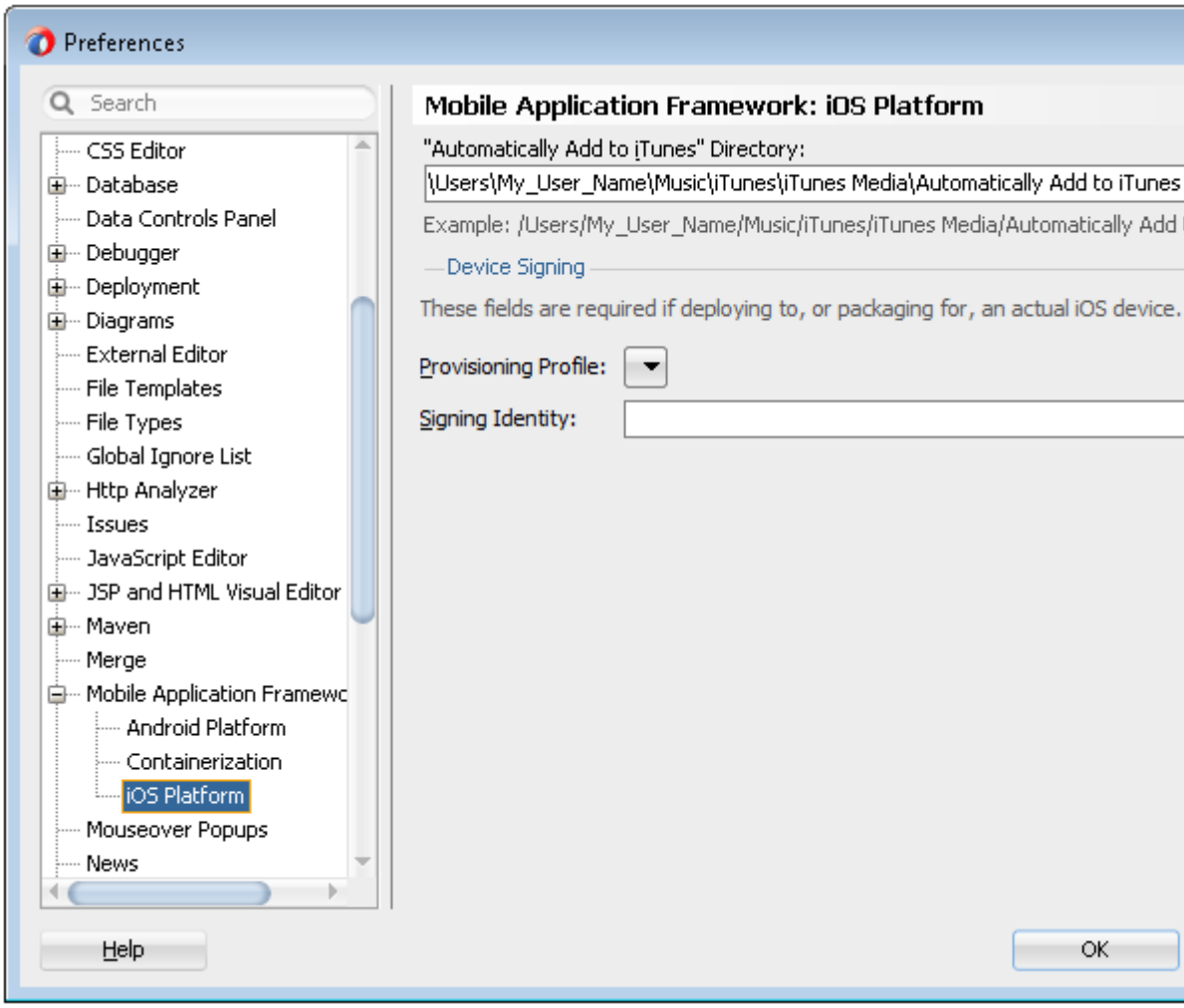
   - For the Android platform, specify the following:
     - The Android SDK location on your computer.
     - The local directory of your target Android platform.
     - The Android build tools location on your computer.
     - Information on the signing credentials.

Figure 2-1 Configuring Platform Preferences for Android



- For the iOS platform, specify the following:
  - Location of the iTunes media files, including the mobile applications that are synchronized to the iOS-powered device.
  - The iOS-powered device signing information (see the "Setting the Device Signing Options" section in *Developing Mobile Applications with Oracle Mobile Application Framework*).

Figure 2-2 Configuring Platform Preferences for iOS



2.3 Configuring the Development Environment for Form Factors

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

Since form factors defined in preferences are used in the MAF AMX page Preview tab (see the "Using the Preview" section in *Developing Mobile Applications with Oracle Mobile Application Framework*), you may choose to perform this configuration if you are planning to include a MAF AMX application feature as part of your MAF application and you do not want to accept the default settings. During development, you can select or switch between various form factors to see how a MAF AMX page is rendered. You can also see multiple form factors applied to the same page using the split screen view.

For more information, see the "About the maf-config.xml File" section in *Developing Mobile Applications with Oracle Mobile Application Framework*.

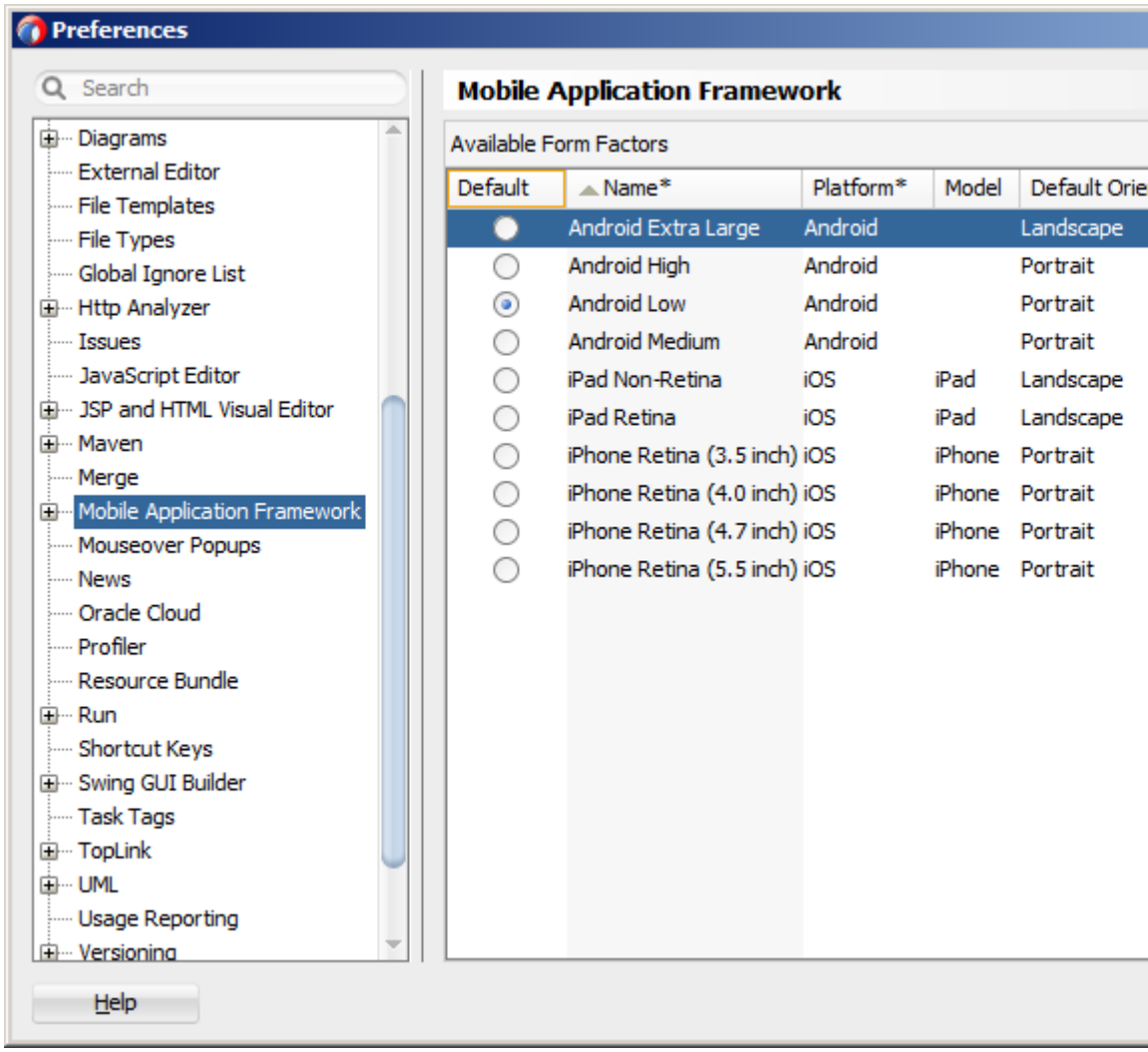
Before you begin:

Download and install JDeveloper and the MAF extension, as described in Installing Mobile Application Framework with JDeveloper .

To configure the form factors:

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

Figure 2-3 Defining Form Factors



The **Mobile Application Framework** page is populated with available form factors and the default is set to Android Low.

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

3. To create a new form factor, click the green plus sign (New), and then set the following:

- **Name**: a meaningful string that is used to identify the form factor.
- **Platform**: the platform of the mobile device.
- **Model**: the type of the mobile device.

- **Default Orientation**: the default device orientation used in the MAF AMX page Preview tab. It might be Portrait or Landscape. Select this setting from the drop-down list of values. The default value is Portrait and it is prepopulated during creation of the new form factor.
- **Width**: width, in pixels. This value must be a positive integer, and its input is validated.
- **Height**: height, in pixels. This value must be a positive integer, and its input is validated.
- **Scale Factor**: the display scale factor. This value must be either one of 1.0, 2.0, or 3.0.

Note:

If you do not set the name and resolution for your form, MAF will display an error message.

---

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

2.4 Setting Up Development Tools for the iOS Platform

In addition to general-purpose tools listed in Introduction to Installing the MAF Extension with JDeveloper, you might want to set up an iPhone or iPad when getting ready for development of a MAF application for the iOS platform (see How to Set Up an iPhone or iPad).

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

2.4.1 How to Install Xcode and iOS SDK

You download Xcode from `http://developer.apple.com/xcode/`. This download includes the iOS SDK.

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

Note:

Since older versions of Xcode and iOS SDK are not available from the Mac App Store, in order to download them you must obtain an Apple ID from `http://appleid.apple.com`, and then register this Apple ID with the Apple Developer Program to gain access to the Apple developer site at `http://developer.apple.com`.

2.4.2 How to Set Up an iPhone or iPad

In your MAF application development and deployment, you can use either the iPhone, iPad, or their simulators (see How to Set Up an iPhone or iPad Simulator). If you are planning to use an actual iPhone or iPad, which is preferable for testing (see the "Testing MAF Applications" section in *Developing Mobile Applications with Oracle Mobile Application Framework*), you need to connect it to your computer to establish a link between the two devices.

To deploy to an iOS-powered device, you need to have an iOS-powered device with a valid license, certificates, and distribution profiles. For more information, see the "Deploying Mobile Applications" chapter in *Developing Mobile Applications with Oracle Mobile Application Framework*.

Note:

Since Apple's licensing terms and conditions may change, ensure that you understand them, comply with them, and stay up to date with any changes.

2.4.3 How to Set Up an iPhone or iPad Simulator

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

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

Note:

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

If you are planning to use web services in your application and you are behind a corporate firewall, you might need to configure the external network access. You do so by modifying the network settings in the System Preferences on your development computer. For more information, see the "Configuring the Browser Proxy Information" section in *Developing Mobile Applications with Oracle Mobile Application Framework*.

2.5 Setting Up Development Tools for the Android Platform

In addition to the general-purpose tools listed in Introduction to Installing the MAF Extension with JDeveloper, you might want to set up an Android-powered device when getting ready for development of a MAF application for the Android platform (see How to Set Up an Android-Powered Device).

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

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

For more information, see the "Developer Tools" section of the Android Developers website at http://developer.android.com/tools/index.html.

2.5.1 How to Install the Android SDK

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

When choosing the platform, keep in mind that MAF supports Android 4.0.3 or later.

Before you begin:

Ensure that your environment meets the operating system, JDK version, and hardware requirements listed in the "Get the Android SDK" section of the Android Developers website at http://developer.android.com/sdk/index.html.

Note:

Ant and Linux requirements are not applicable to the MAF development environment; Eclipse might be applicable depending on your IDE of choice.

To install the Android SDK:

1. Download the Android SDK starter package from http://developer.android.com/sdk/index.html.
2. Complete the installation by following the instructions provided in the "Setting Up an Existing IDE" section of the Android Developers website at http://developer.android.com/sdk/installing.html.

Note:

If you are not planning to use Eclipse, skip step 3 in the Android SDK installation instructions.

2.5.2 How to Set Up an Android-Powered Device

In your MAF application development and deployment, you can use either the Android device itself, which is preferable for testing (see the "Testing MAF Applications" section in *Developing Mobile Applications with Oracle Mobile Application Framework*), or an emulator (see How to Set Up an Android Emulator.).

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

Note:

You might experience issues when using USB connectivity for the device-based debugging. For more information, see the "Testing and Debugging MAF Applications" chapter in *Developing Mobile Applications with Oracle Mobile Application Framework*.

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

1. Find the correct values for your device.
2. Update the `[Google.NXx86]` and `[Google.NTamd64]` sections of the `android_winusb.inf` file.

For more information, see the "Google USB Driver" section of the Android Developers website at `http://developer.android.com/sdk/win-usb.html`.

2.5.3 How to Set Up an Android Emulator

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

For information on how to create an emulator configuration called Android Virtual Device (AVD), follow the instructions from the "Managing Virtual Devices" section of the Android Developers website at `http://developer.android.com/tools/devices/index.html`. When creating an AVD through the Create New Android Virtual Device dialog (see "Managing AVDs with AVD Manager" at `http://developer.android.com/tools/devices/managing-avds.html`), review all the settings to ensure that configuration matches what you are planning to emulate. In particular, you should verify the following:

- The Target field should define the desired Android platform level for proper emulation.
- The CPU/ABI field should reflect the ARM or Intel Atom system image (see Configuring AVD for Intel HAXM).
- The SD card field should be defined based on whether the application uploads files or files install themselves to the SD card.
- Default settings for the Hardware field (see the "Hardware Options" table at `http://developer.android.com/tools/devices/managing-avds.html#hardwareopts`) should be acceptable for a typical MAF application. For additional hardware capabilities you may want to use in your application, such as cameras or geolocation services, create new properties.

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

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

2.5.3.1 Configuring the Android Emulator

After the basic Android emulator setup is complete, you may choose to perform the following configurations:

- Save the emulator state (see Saving the Emulator State)
- Create, save, and reuse the SD card (see Creating_ Saving_ and Reusing the SD Card)

- Configure the network (see Configuring the Network)
- Configure the network proxy (see Configuring the Network Proxy)

2.5.3.1.1 Saving the Emulator State

You can reduce the emulator's load time by saving the emulator state or reusing the saved state. To do so, you manipulate the `avd` files or folders that are located in the `C:\Users\`*`username`*`\.android\avd` directory (on a Windows computer). Each `avd` folder contains several files, such as `userdata.img`, `userdata.qemu.img`, and `cache.img`. You can copy the `cache.img` file to another emulator's `avd` folder to use that state with another emulator.

Alternatively, you can use the command line to run relevant commands, such as, for example, `-snapshot-list`, `-no-snapstorage`, and so on. You can access these commands through `emulator -help` command.

Caution:

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

2.5.3.1.2 Creating, Saving, and Reusing the SD Card

The "SD Card Emulation" section of the Android Developers website at `http://developer.android.com/tools/devices/emulator.html#sdcard` lists reasons for creating, saving, and reusing the SD card. You can perform these operations by executing the following commands:

- To create an SD card:

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

- To list existing AVDs:

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

  This produces a listing similar to the following:

  ```
  Name:    AndroidEmulator1
  Device:  Nexus S (Google)
  Path:    C:\Users\username\.android\avd\AndroidEmulator1.avd
  Target:  Android 4.2.2 (API level 17)
  Tag/ABI: default/x86
  Skin:    480x800
  ------------
  Name:    AndroidEmulator2
  Device:  Nexus S (Google)
  Path:    C:\Users\username\.android\avd\AndroidEmulator2.avd
  Target:  Android 4.2.2 (API level 17)
  Tag/ABI: default/armeabi-v7a
  Skin:    480x800
  Sdcard:  500M
  ```

- To start the AndroidEmulator2 with the SD card that has just been created:

  ```
  C:\Android\android sdk directory\tools>emulator -avd AndroidEmulator2 -sdcard C:
  \Android\sd500m.img
  ```

- To list the running Android emulator instances:

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

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

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

For more information, see the Android Tools Help at http://developer.android.com/tools/help/index.html.

2.5.3.1.3 Configuring the Network

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

To forward socket connections, execute

`adb forward local remote`

using the following forward specifications:

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

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

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

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

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

2.5.3.1.4 Configuring the Network Proxy

If your development computer is behind a corporate firewall, you might need to configure a proxy by using one of the following techniques:

1. Execute this command to start the emulator and initiate its connection with the browser:

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

2. Start the emulator and then use its Settings utility as follows:

   a. Select Wireless & Networks

   b. Select Mobile Networks > Access Point Names

c. Select the appropriate internet option

d. Set the proxy, port, username, and password using the Edit access point list

2.5.3.2 Speeding Up the Android Emulator

The Intel Hardware Accelerated Execution Manager (Intel HAXM) is designed to accelerate the Android-powered device emulator by making use of Intel drivers.

The Intel HAXM is available for computers running Microsoft Windows, Mac OS X, and a separate kernel-based virtual machine option (KRM) for Linux. See `http://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager` to access installation guides and detailed descriptions of system requirements for each operating system.

Regardless of which operating system your development computer is running on, it must have the following:

- Version 17 or later of the Android SDK installed (see How to Install the Android SDK).

Note:

Currently, the recommended version for MAF development is 21.

- Intel processor with support for Intel VT-x, EM64T and Execute Disable (XD) Bit functionality at the BIOS level.
- At least 1 GB of available RAM.

To download the Intel HAXM, either use the Android SDK Manager (see *Speeding Up the Android Emulator on Intel Architecture*) or use the following Intel locations:

- Download for Microsoft Windows
- Download for Mac OS X
- Download for Linux

To install the Intel HAXM, follow the steps described in the "Speeding Up the Android Emulator on Intel Architecture" article available at `http://software.intel.com/en-us/android/articles/speeding-up-the-android-emulator-on-intel-architecture`. It is particularly important to configure AVD (see Configuring AVD for Intel HAXM).

If your development computer is running either Microsoft Windows 8.*n* or later, or Mac OS X 10.9.*n* or later, you have to apply a Hotfix provided by Intel before using the emulator with the Intel HAXM.

Note:

If you do not apply the Hotfix, your computer will freeze and you will lose your work.

To download the Hotfix, use the following locations:

- Download for Microsoft Windows

- Download for Mac OS X

For more information, see the following:

- *Installation Guide and System Requirements - Windows*
- *Installation Guide and System Requirements - Mac OS X*
- *Installation Guide and System Requirements - Linux*

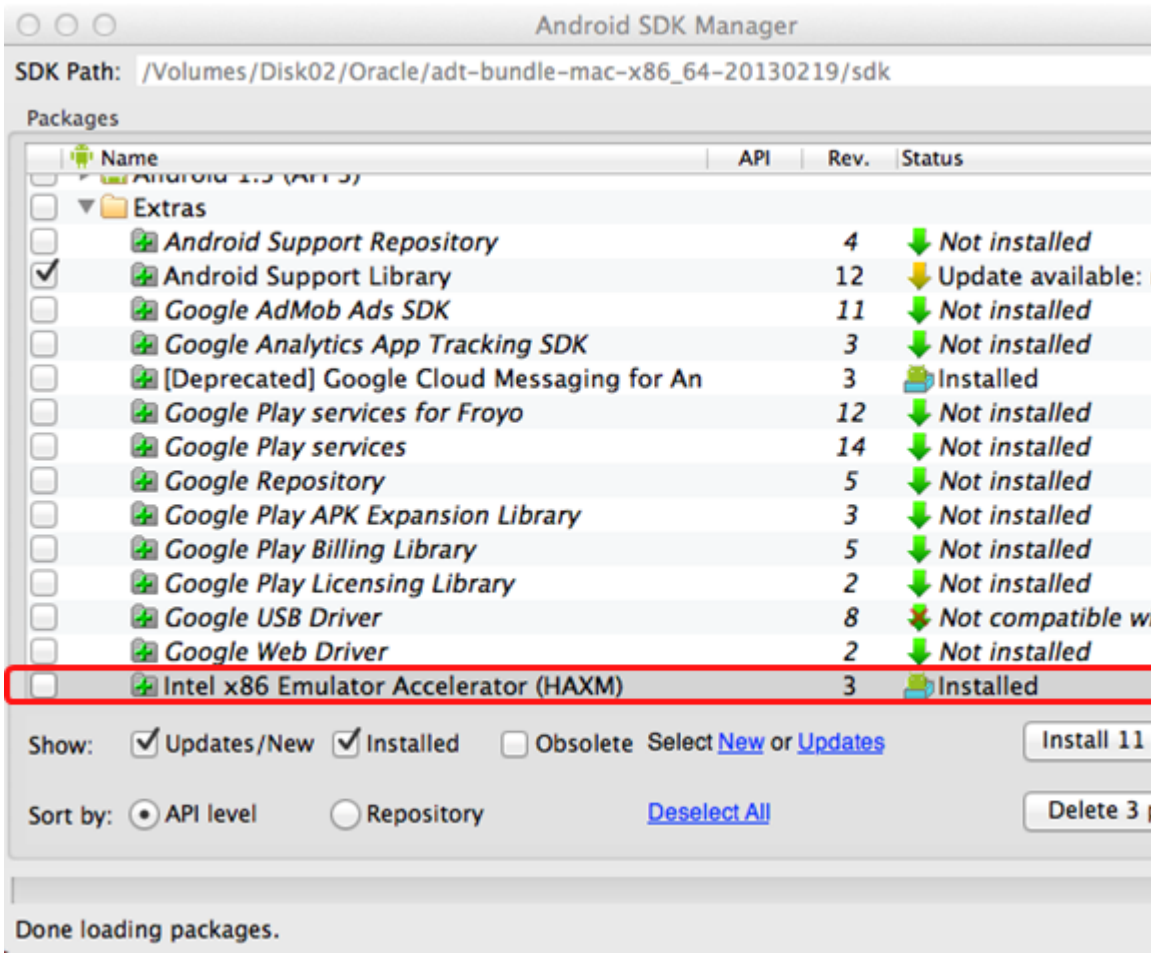
2.5.3.2.1 Configuring AVD for Intel HAXM

When enabling the Intel HAXM, ensure that you download the Intel system image for the Android API level using the Android SDK Manager (see Figure 2-4). The following steps described in *Speeding Up the Android Emulator on Intel Architecture* guide you through the configuration process:

- After you have installed the Android SDK, open the SDK Manager and then find the Intel HAXM in the extras section.
- Select **Intel x86 Emulator Accelerator (HAXM)** and click **Install packages**.

  Once you have installed the package, the status changes to Installed, which is not accurate: the SDK only copies the Intel HAXM executable on your computer; you have to manually install the executable.

Figure 2-4 Downloading Intel System Image in Android SDK Manager

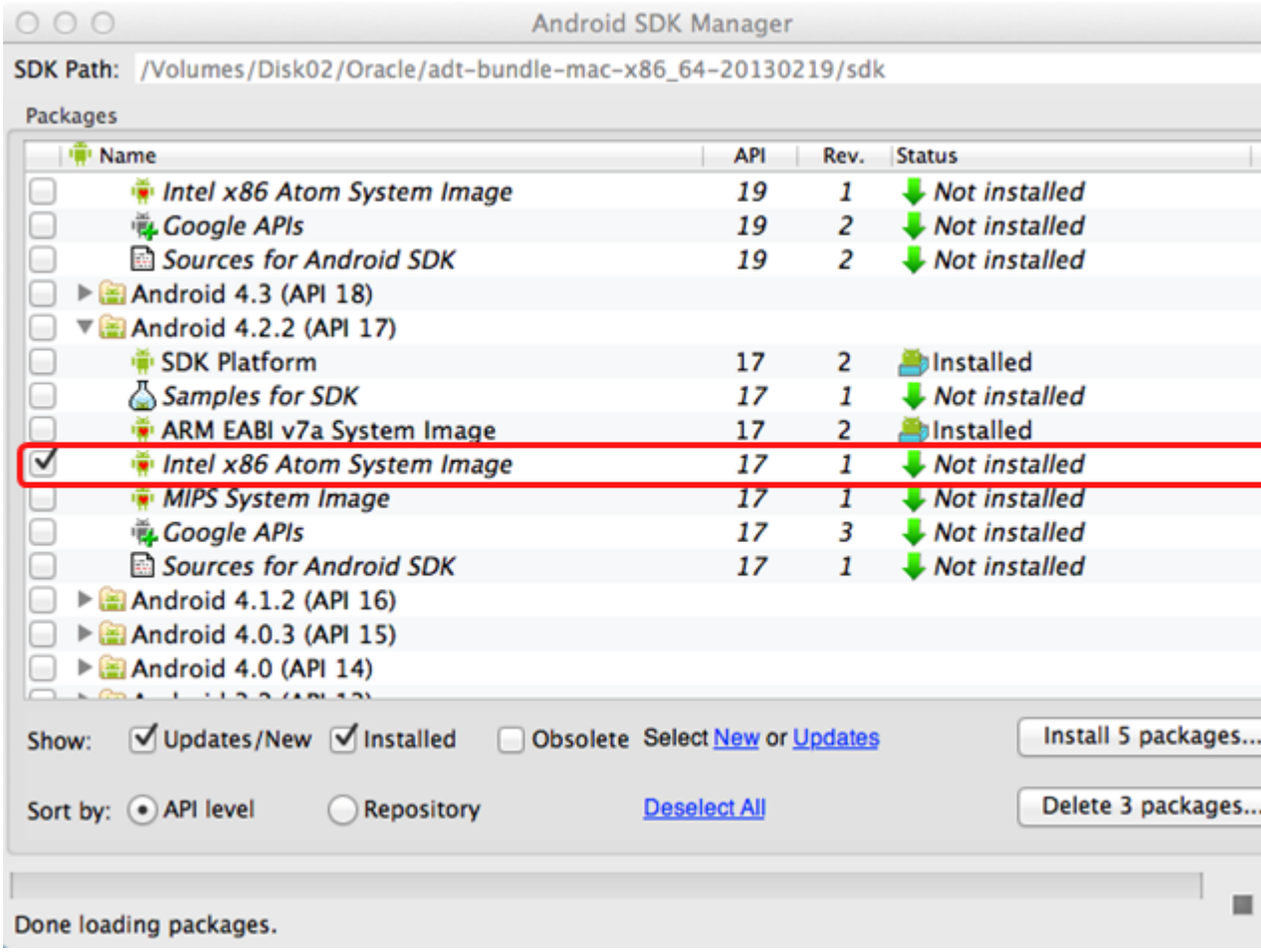


- To install the Intel HAXM executable, depending on your development platform search your hard drive for one of the following:
  - On Windows, search for `IntelHaxm.exe`
  - On Mac OS X, search for `IntelHaxm.dmg`

  If you accepted default settings, the executable should be located at `C:\Program Files\Android\android-sdk\extras\Intel\Hardware_Accelerated_Execution_Manager\IntelHaxm.exe` on Windows.

The Intel HAXM only functions in combination with one of the Intel Atom processor x86 system images, which are available for Android 2.3.3 (API 10), 4.0.3 (API 15), 4.1.2 (API 16), 4.2.2 (API 17), 4.4 (API 19), 4.4W (API 20), 5.0 (API 21). These system images can be installed exactly like the ARM-based images through the Android SDK Manager.

Figure 2-5 Installing Intel Atom System Image

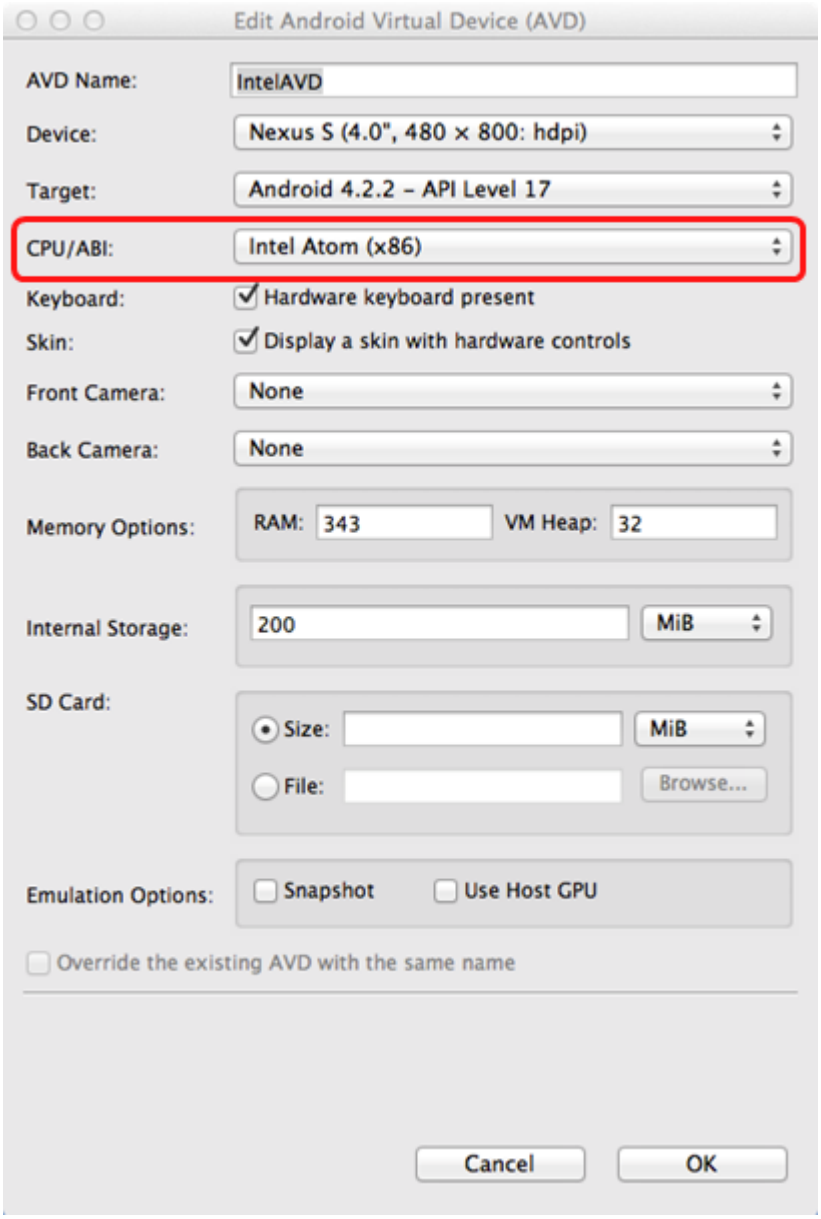


To complete the process, use the AVD Manager to create a new virtual device that has hardware-accelerated emulation by selecting **Intel Atom (x86)** as the CPU/ABI, (see Figure 2-6).

Note:

This option appears in the list only if you have the Intel x86 system image installed.

Figure 2-6 Creating Accelerated AVD



2.6 Testing the Environment Setup

You can test your environment setup as follows:

1. In JDeveloper, open the HelloWorld sample application by selecting the `HelloWorld.jws` file (see the "Mobile Application Framework Sample Applications" appendix in *Developing Mobile Applications with Oracle Mobile Application Framework*).

2. Select Application > Deploy from the main menu.

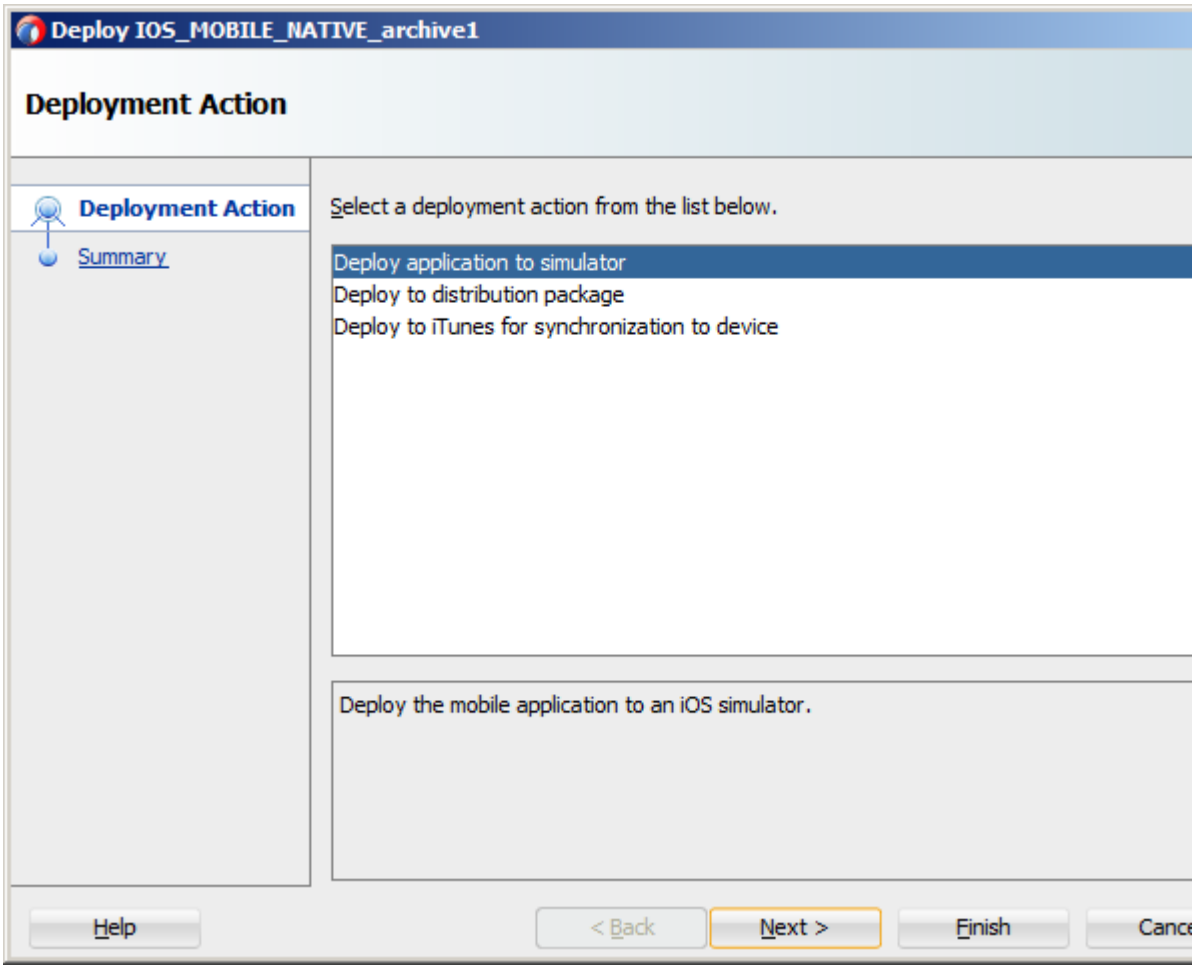
   For more information, see the "Deploying Mobile Applications" chapter in *Developing Mobile Applications with Oracle Mobile Application Framework*.

3. From the drop-down menu, select the deployment profile for the platform to which you wish to deploy the application.

4. Since using an iOS-powered device simulator or Android-powered device emulator to test the environment setup is preferable because it does not require signing of the application, you should select one of the following deployment actions using the **Deploy** dialog:

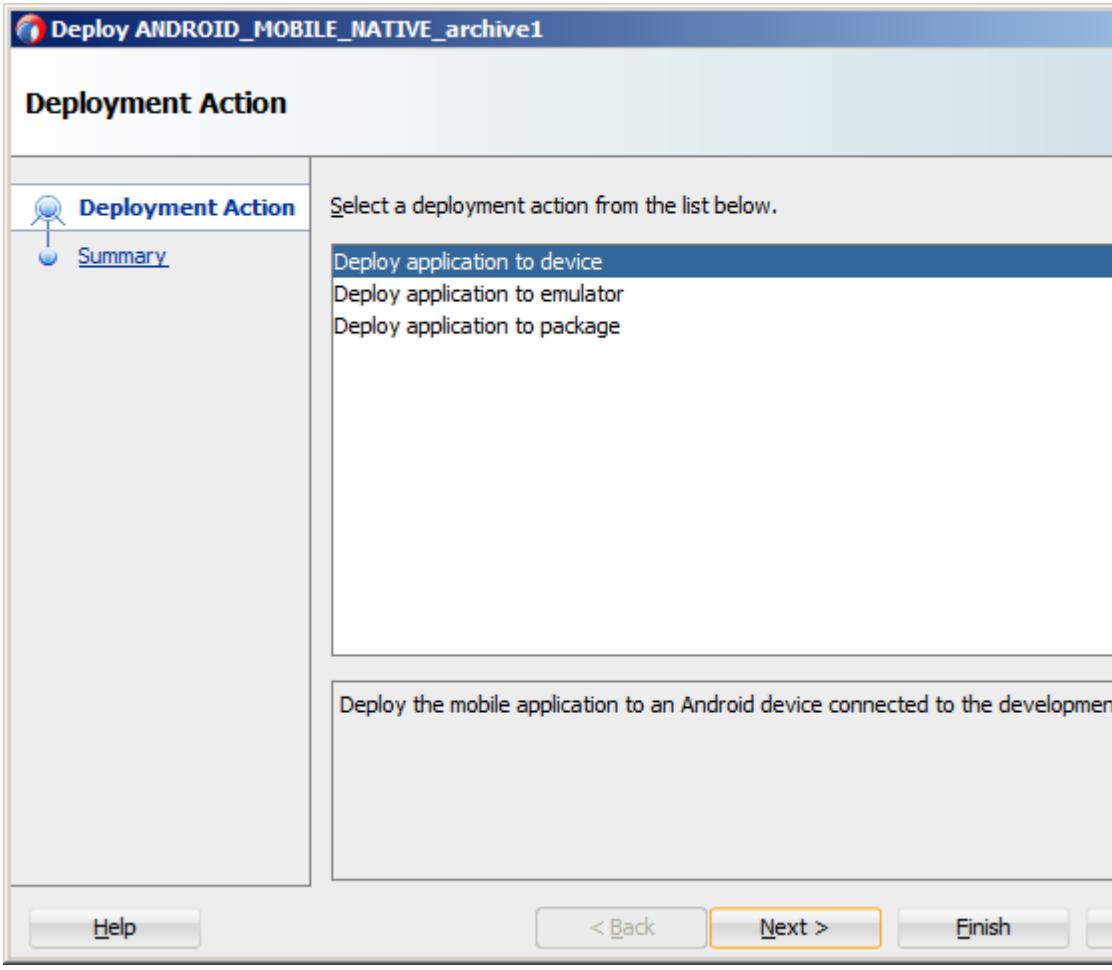
   - For iOS, select **Deploy application to simulator**, as Figure 2-7 shows.

   Figure 2-7 Selecting Deployment Action for iOS

   - For Android, select **Deploy application to emulator**, as Figure 2-8 shows. Ensure that the emulator is running before you start the deployment.

Figure 2-8 Selecting Deployment Action for Android



5. Click **Next** on the Deploy dialog to verify the Summary page, and then click **Finish**.

   For more information, see one of the following sections in *Developing Mobile Applications with Oracle Mobile Application Framework*:

   - "How to Deploy an iOS Application to an iOS Simulator"
   - "How to Deploy an Android Application to an Android Emulator"

For more information on deployment, see the "Deploying Mobile Applications" chapter in *Developing Mobile Applications with Oracle Mobile Application Framework*

After a successful deployment (which might take a few minutes), your iOS-powered device simulator or Android-powered device emulator will display the HelloWorld application icon that you have to activate to launch the application.

3

Migrating Your Application to MAF 2.2.2

This chapter provides information that you may need to know if you migrate an application created using an earlier release of MAF to MAF 2.2.2.

This chapter includes the following sections:

3.1 Migrating an Application to MAF 2.2.2

MAF enables App Transport Security (ATS) by default for applications that you migrate to this release. For more information, see **Security Changes in Release 2.2.1 and Later of MAF**. If your migrated application uses URL schemes to invoke other applications, configure the migrated application as described in **Migrating MAF Applications that Use Customer URL Schemes to Invoke Other Applications**.

The MAF 2.1.0 release introduced significant changes described in this chapter. Use the information in this chapter if you migrate an application created in a pre-MAF 2.1.0 release to MAF 2.2.2.

MAF 2.1.0 used newer versions of Apache Cordova and Java. It also changed the way that JDeveloper registered plugins in your MAF application. For SSL, it delivered a `cacerts` file that contains new CA root certificates.

If you migrate an application to MAF 2.2.2 that was created in MAF 2.1.0 or previously migrated to MAF 2.1.0, MAF will have made already made the changes required by migration to JDK 8, management of Cordova plugins, and a new `cacerts` file.

Read the subsequent sections in this chapter that describe how these changes impact the migration of your MAF application to MAF 2.1.0 or later.

Finally, MAF 2.1.0 delivered an updated SQLite database and JDBC driver. Review, and migrate as necessary, any code in your migrated MAF application that connects to the SQLite database. For more information about how to connect to the SQLite database, see the "Using the Local SQLite Database" section in the *Developing Mobile Applications with Oracle Mobile Application Framework*.

After you migrate your application to this release, invoke JDeveloper's Clean All command. This cleans your application of build artifacts from builds prior to migrating to this release. To do this, click **Build** > **Clean All** from JDeveloper's main menu.

3.2 Security Changes in Release 2.2.1 and Later of MAF

Starting with MAF 2.2.1, use of HTTPS with TLS 1.2 for all connections to the server from MAF applications on iOS is required. Any MAF application that uses non-HTTPS connections and an SSL version lower than TLS1.2 will fail to run on iOS. MAF enforces this behavior to meet Apple iOS 9's requirement to use App Transport Security (ATS) that requires use of HTTPS with TLS 1.2. You can disable use of ATS, as described below.

MAF applications also adhere to the default behavior enforced by Java 8's JVM to use the latest SSL version and cipher suites. While we encourage you to upgrade your servers to use these later versions, you can configure your MAF application to work around SSL errors you may encounter by using servers with older SSL versions, as described below.

Disabling App Transport Security for MAF Applications on iOS Devices

MAF applications that you migrate to this release of MAF enable ATS by default. You can disable ATS in your MAF application as follows:

1. In JDeveloper, choose **Application** > **Application Properties** > **Deployment**.
2. In the Deployment page, double-click the iOS deployment profile.
3. Choose **iOS Options**.
4. Select **Disable Application Transport Security** and click **OK**.

SSL Configuration Changes

Customers who use SSL versions lower than TLS 1.2, deprecated cipher suites or deprecated encryption algorithms will see SSL errors like "`invalid cipher suite`", "`close notify`", "`TLS error`", and so on. Java 8 enforces use of the latest SSL version and cipher suites. It disables use of insecure SSL versions by default. We encourage you to update your servers to use the later SSL version. If this is not possible, you can use the following configuration to work around the SSL errors just described:

1. Update `maf.properties` file with the version of SSL that you want to use. For example, add the following entry to the `maf.properties` file to use TLS 1:

   ```
   java.commandline.argument=-Dhttps.protocols=TLSv1
   ```

2. Update `maf.properties` file with the full list of cipher suites required by the application. For the list of cipher suites that Java supports, see the Cipher Suites section on this page.

   For example, to enable `SSL_RSA_WITH_RC4_128_MD5`, add the following:

   ```
   java.commandline.argument=-D SSL_RSA_WITH_RC4_128_MD5
   ```

3. Update the `java.security` file to enable deprecated algorithms. Existing MAF applications will not have this file so create a new empty MAF application and copy the `java.security` file created in the new MAF application's `/resources/security` to the same directory in the existing application.

   For example, the RC4 algorithm is disabled by default per the following entry in the `java.security` file:

   ```
   jdk.tls.disabledAlgorithms=SSLv3, RC4, DH keySize < 768
   ```

   If you use a cipher suite that requires the RC4 algorithm, such as `SSL_RSA_WITH_RC4_128_MD5`, an error is thrown at runtime while establishing the SSL connection. To work around this, change the `java.security` entry as follows to enable the RC4 algorithm:

   ```
   jdk.tls.disabledAlgorithms=SSLv3, DH keySize < 768
   ```

3.3 Maintaining Separate Xcode Installations for MAF 2.2.2 and MAF 2.2.0

You can create two MAF development environments where you install two different versions of Xcode and two instances of JDeveloper on the same machine. Post installation, you manually activate the version of Xcode you want to use. JDeveloper uses the currently-active instance of Xcode.

MAF 2.2.2 requires Xcode 7.x. If you want to maintain one development environment only (for MAF 2.2.2), install or upgrade to Xcode 7.x, as described in How to Install Xcode and iOS SDK. Once you install or upgrade to Xcode 7.x, make sure to start it so that you accept the license agreements. Failure to do this may cause deployment errors when JDeveloper attempts to deploy a MAF 2.2.2 application to iOS. With this installation, Xcode 7.x replaces Xcode 6.x. No other changes are required, since JDeveloper will now use the active Xcode installation.

If you want to maintain separate development environments for MAF 2.2.2 (using Xcode 7.x) and MAF 2.2 (using Xcode 6.x), you can install both Xcode 6.x and Xcode 7.x on the same machine where you install separate JDeveloper environments for MAF 2.2 and MAF 2.2.2. See the following procedures for information about how you can accomplish this task.

For a complete list of supported versions of development and runtime tools, see Oracle Mobile Application Framework Certification Matrix by following the Certification Information link on the MAF documentation page at http://www.oracle.com/technetwork/developer-tools/maf/documentation/.

How To Maintain Separate JDeveloper Environments for MAF 2.2.2 and 2.2.0

To maintain separate JDeveloper environments for MAF 2.2.0 and 2.2.2:

1. Install a new instance of JDeveloper for MAF 2.2.2, so that you have one instance of JDeveloper for each version of MAF. That is, your preexisting JDeveloper installation for MAF 2.2.0 and a new installation of JDeveloper for MAF 2.2.2, as shown in the following example:

   ```
   /usrdir/Oracle/Middleware/Oracle_Home/maf222/jdeveloper
   ```

   In the Installation Complete screen of the Installation wizard, select the **Finish Installation without Starting JDeveloper** radio button under the Next Steps field. You need to configure the `jdev.conf` file before you start JDeveloper, as described in the next step.

2. Before you start the JDeveloper instance for MAF 2.2.2 after installation, review its `jdev.conf` file to make sure that the system folder for the new installation points to a different location to that used for the preexisting JDeveloper installation for MAF 2.2.0. Also verify that the `SetJavaHome` variable in the `jdev.conf` file references JDK 7. This release of MAF requires JDK 8 to compile, but you install JDeveloper using JDK 7, as described in Introduction to Installing the MAF Extension with JDeveloper.

   The `jdev.conf` file is in the `/usrdir/Oracle/Middleware/Oracle_Home/maf222/jdeveloper/jdev/bin` directory

   The following example shows entries for a JDeveloper installation used for MAF 2.2.2 development. Review the `jdev.conf` file in the JDeveloper installation that you use for MAF 2.2.0 development to make sure that you use a different value for the system folder in the MAF 2.2.2 installation.

   ```
   # Point -Dide.system.dir variable to the folder where JDeveloper extracts its
   system folder.
   AddVMOption -Dide.system.dir=/usrdir/Oracle/Middleware/Oracle_Home/maf222/
   jdeveloper

   # SetJavaHome variable must point to the JDK 1.7 Home
   SetJavaHome /Library/Java/JavaVirtualMachines/jdk1.7.0_79.jdk/Contents/Home
   ```

   These entries make sure that the system folders of both JDeveloper installations are in different locations and, as a result, you do not inadvertently overwrite content in one environment while using another.

How To Maintain Separate Xcode 7.x and Xcode 6.x Installations

To maintain separate Xcode 7.x and Xcode 6.x installations:

1. Rename the preexisting Xcode.app installation for Xcode 6.x (For example, `Xcode6.app`.)
2. Install Xcode 7.x from the Apple App Store, as described in How to Install Xcode and iOS SDK. Make sure that you install, not update, Xcode from the Apple App Store.
3. Once you install Xcode 7.x, make sure to start it so that you accept the license agreements.

   After installation, verify that you have the following Xcode installations in your Applications location:

   ```
   Xcode 7.x installation:
   /Applications/Xcode.app

   Xcode 6.x installation:
   /Applications/Xcode6.app
   ```

4. Once the two versions of Xcode have been installed, you must manually control which Xcode installation is active at any given time. Use the `xcode-select` command in a terminal window to perform this procedure, as shown in the following examples:

   ```
   //To make Xcode 7.x active:
   sudo xcode-select -s /Applications/Xcode.app

   //To make Xcode 6 active:
   ```

```
sudo xcode-select -s /Applications/Xcode6.app

//To determine which instance of Xcode is currently active:
xcode-select --print-path
```

3.4 Migrating MAF Applications that Use Customer URL Schemes to Invoke Other Applications

If the application you migrate to MAF 2.2.2 uses a custom URL scheme to invoke another application, add the scheme(s) to the **Allowed Scheme** list in the Security page of the `maf-application.xml` file's overview editor.

This change addresses iOS 9's requirement that applications declare any URL schemes they use to invoke other applications. Click the **Add** icon in the Allow Schemes section of the Security page to add the custom URL scheme, as shown in Figure 3-1.

Figure 3-1 Registering a Custom URL Scheme that a MAF Applications Use to Invoke Another Application

3.5 Migrating to JDK 8 in MAF 2.2.2

MAF applications that you create in MAF 2.1.0 and later use JDK 8. You specify the location of your JDK 8 installation the first time you start JDeveloper after installing the MAF extension, as described in Installing the MAF Extension in JDeveloper.

If you migrate a MAF application that compiled with an earlier version of Java, note that MAF 2.1.0 and later requires JDK 8 and compiles applications using the Java SE Embedded 8 compact2 profile. When you open an application that you migrated from a pre-MAF 2.1.0 release in MAF 2.2.2 for the first time, JDeveloper makes the following changes:

- Renames the configuration file that specifies the startup parameters of the JVM from `cvm.properties` to `maf.properties`. For more information about the `maf.properties` file, see the "How to Enable Debugging of Java Code and JavaScript" section in *Developing Mobile Applications with Oracle Mobile Application Framework*.

- Replaces instances (if any) of the following import statement in the application's Java source files:

  ```
  com.sun.util.logging
  ```

  With:

  ```
  java.util.logging
  ```

- Replaces the following entries in the application's `logging.properties` file

  ```
  .handlers=com.sun.util.logging.ConsoleHandler
  .formatter=com.sun.util.logging.SimpleFormatter
  ```

  With:

```
.handlers=java.util.logging.ConsoleHandler
.formatter=java.util.logging.SimpleFormatter
```

For more information about the `logging.properties` file, see the "How to Configure Logging Using the Properties File" section in *Developing Mobile Applications with Oracle Mobile Application Framework*.

3.6 Migrating Cordova Plugins from Earlier Releases to MAF 2.2.2

MAF applications developed using earlier releases of MAF (prior to MAF 2.1.0) registered plugins in the `maf-application` file. Release MAF 2.1.0 and later registers plugins in the `maf-plugins.xml` file. JDeveloper makes the following changes to an application from an earlier release that uses plugins when you migrate the application:

- Comments out entries in the `maf-application.xml` file that referenced plugins. For example, JDeveloper comments out entries such as the following:

```
<!--<adfmf:cordovaPlugins>
    <adfmf:plugin fullyQualifiedName="BarcodeScanner"
                  implementationClass="com.phonegap.plugins.
                       barcodescanner.BarcodeScanner" platform="Android"
                                                       name="BarcodeScanner">
  .....
</adfmf:cordovaPlugins>-->
```

- Registers the plugin in the `maf-plugins.xml` file, as shown in the following example:

```
<cordova-plugins>
    ...
    <cordova-plugin id="c3" pluginId="org.apache.cordova.barcodeScanner">
      <platform id="p3" name="ios" enabled="true"/>
      <platform id="p4" name="android" enabled="false"/>
    </cordova-plugin>
  </cordova-plugins>
```

To complete the migration and make sure that your migrated MAF application can use the plugins it used previously, verify that the:

- Version of the plugin is supported by MAF.

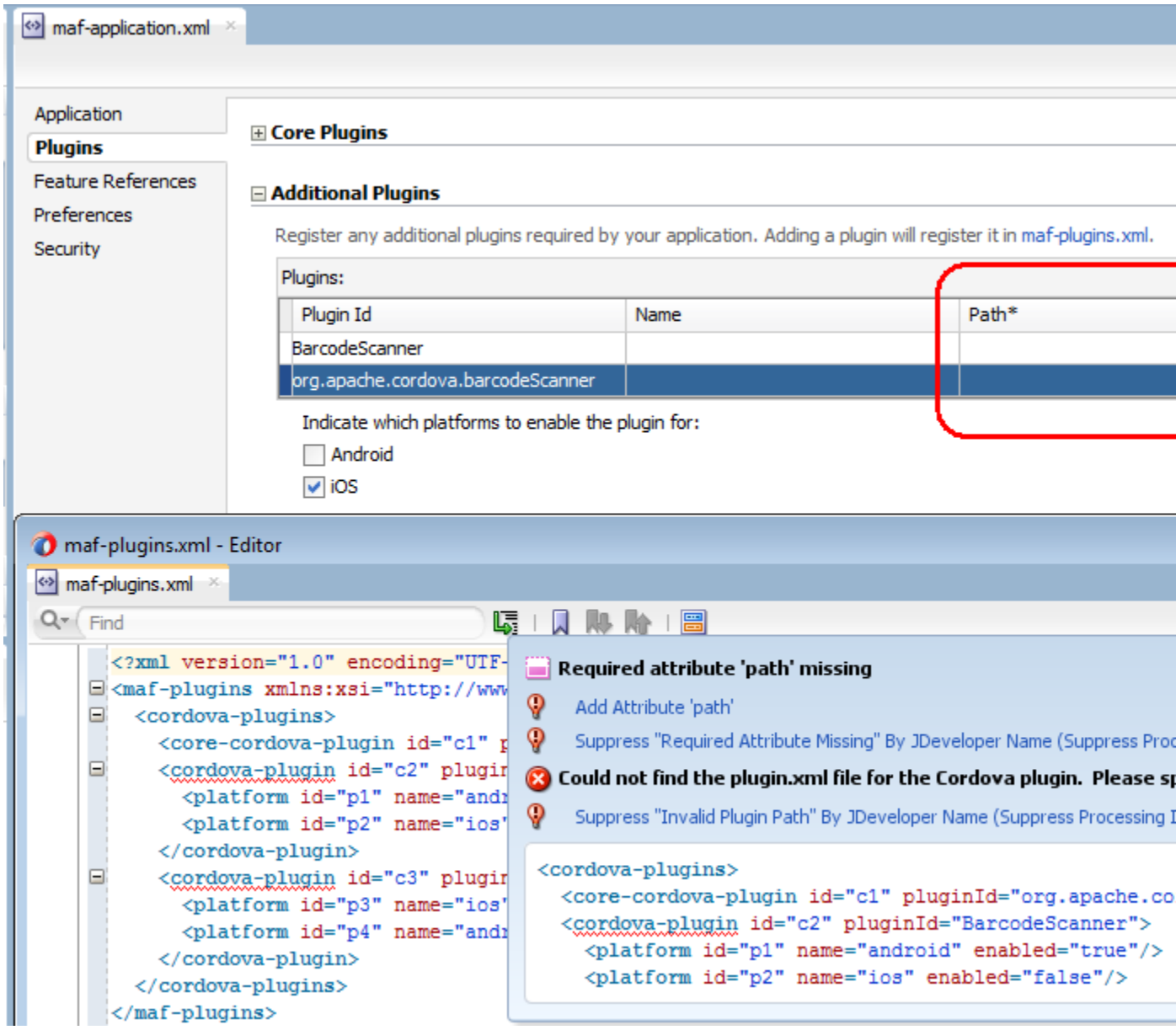
  MAF applications in 2.2.2 use Cordova 3.7.2 on Android and Cordova 3.8.0 on iOS.

  Obtain a newer version of the plugin if the plugin was created using an earlier release of Cordova than that used by the current release of MAF.

- Set the relative path to the plugin so that the MAF application's `maf-plugins.xml` file correctly references the plugin. For more information, see the "Registering Additional Plugins in Your MAF Application" section in *Developing Mobile Applications with Oracle Mobile Application Framework*.

  If the `maf-plugins.xml` file does not correctly reference a plugin using a relative path, the overview editor for the `maf-application.xml` file's **Path*** field which requires a value is empty and the `maf-plugins.xml` displays a validation failure, as shown in Figure 3-2.

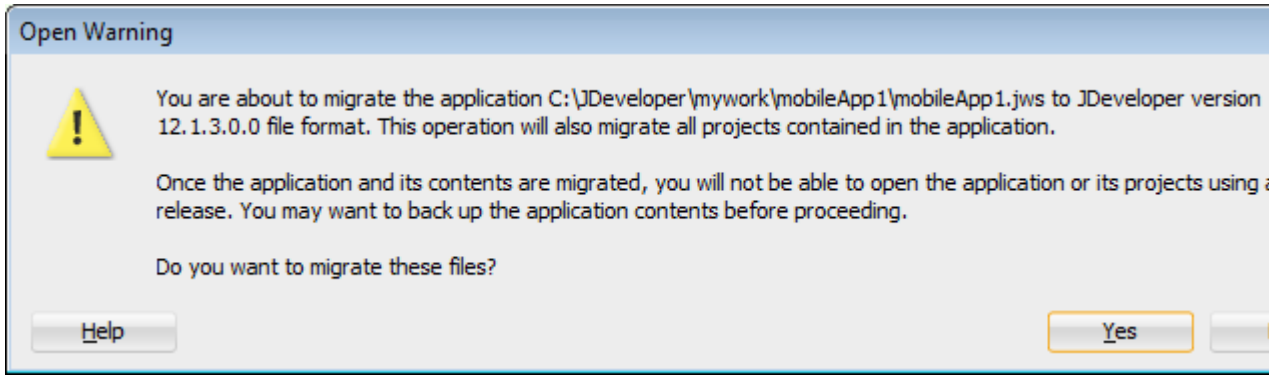
Figure 3-2 MAF Application that Does Not Specify Path to Plugin



3.7 Migrating ADF Mobile Applications

MAF automatically migrates the configuration of applications written in Versions 11.1.2.3.0 and 11.1.2.4.0 of ADF Mobile. After you open the workspace (`.jws`) file of an ADF Mobile application, MAF alerts you that the application is not the current version by presenting the Open Warning dialog (illustrated in Figure 3-3), that prompts you to continue with the migration, or dismiss the dialog and close the file.

Figure 3-3 Open Warning Dialog

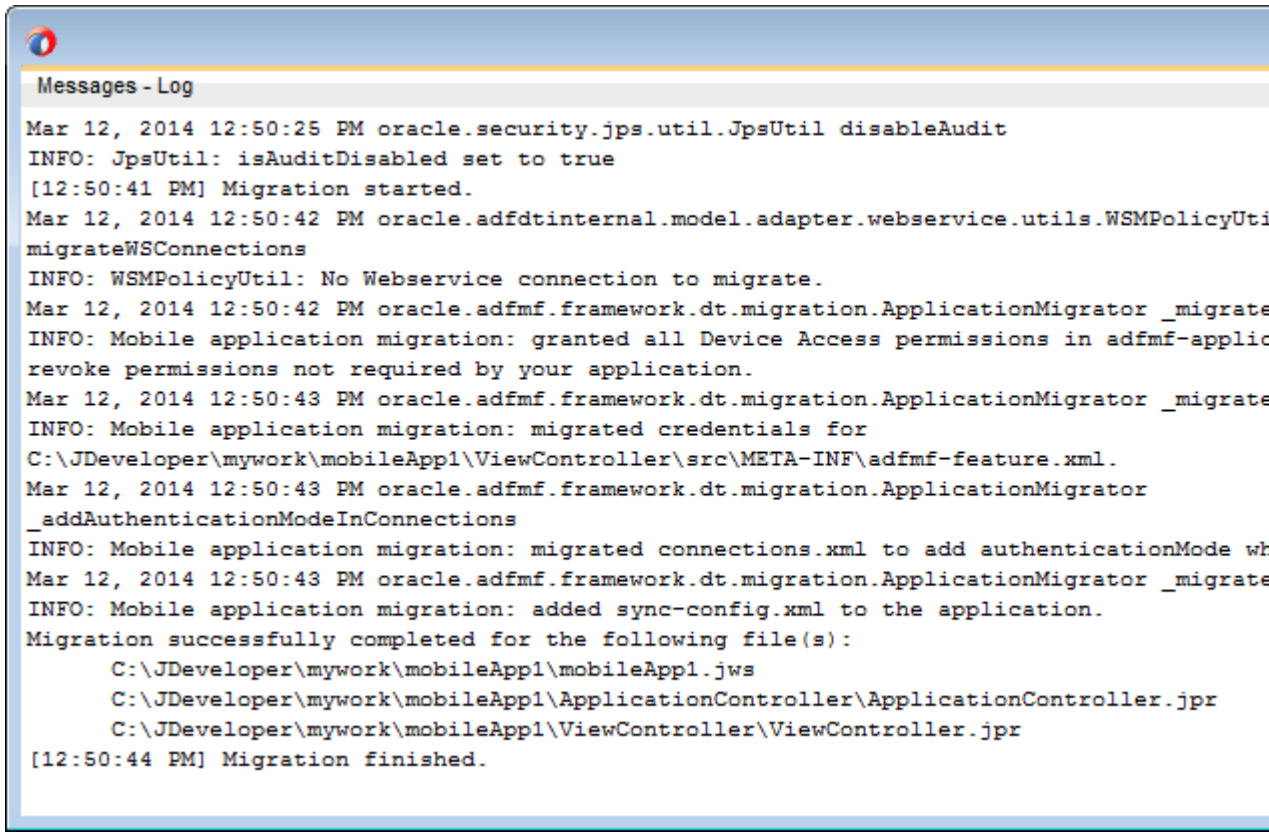


MAF writes the status of the migration to the Log window, as illustrated by Figure 3-4. The migration process also logs the following warning if it detects that the application to migrate uses the old configuration service API.

```
The MAF 2.0 Configuration Service API is not backwards compatible with previous
versions and cannot be migrated automatically. Refer to Section 9.3 "Migrating
the Configuration Service API" in Oracle Fusion Middleware Developing Mobile
Applications with Oracle Mobile Application Framework 2.0. for information on
migrating to the new API.
```

For more information, see the "Migrating the Configuration Service" section in *Developing Mobile Applications with Oracle Mobile Application Framework*.

Figure 3-4 Migration Log



3.7.1 What Happens When You Migrate an ADF Mobile Application

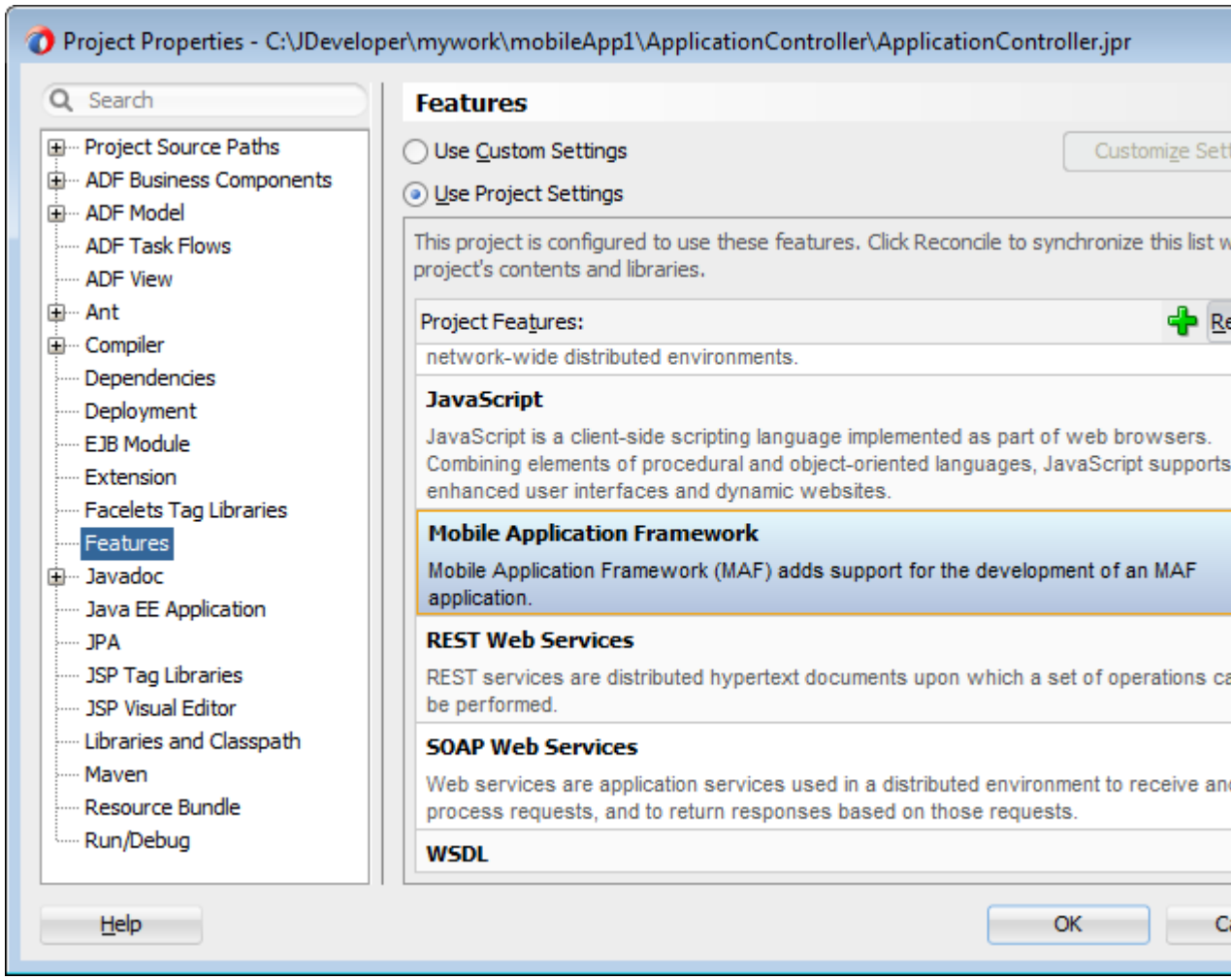
Table 3-1 describes how migration affects ADF Mobile artifacts.

Table 3-1 Migration of ADF Mobile Artifacts and Configuration

File Name	Change
`adfmf-feature.xml`	The migration makes the following changes: • Renames the file as `maf-feature.xml`. • Replaces the `credentials` attribute with `securityEnabled=true`. • Transcribes the `credentials` attribute definition (defined as either `local` or `remote`) as a hybrid connection definition (`<authenticationMode value="hybrid"/>`) in the `connections.xml` file.
`adfmf-application.xml`	The migration renames the file as `maf-application.xml`.
`connections.xml`	The migration removes the secure SOAP web service connections defined by the `<policy-references>` element from the `connections.xml` file. These definitions are populated to the `wsm-assembly.xml` file. The migration creates stub `connections.xml` and `wsm-assembly.xml` files if the ADF Mobile application does not include a `connections.xml` file. If the ADF Mobile application includes a `connections.xml` that has no web services policy definitions, then the migration creates a stub `wsm-assembly` file.
`adfmf-config.xml`	The migration renames the file as `maf-config.xml`. It also adds the default skin version for the skin family if the skin family is the default skin family and the skin version is not specified. For example, the `maf-config.xml` may be modified to include the following values: `<skin-family>mobileAlta</skin-family> <skin-version>v1.1</skin-version>`
`adfmf-skins.xml`	The migration renames the file as `maf-skins.xml`.

The application migrates from the ADF Mobile Framework technology to use the Mobile Application Framework technology as a project feature. Figure 3-5 shows the Features page for an application controller project that uses the Mobile Application Framework technology. Choose **Project Properties > Features** to view this dialog.

Figure 3-5 Mobile Application Framework Project Feature



MAF does not override the icon, splash screen, or navigation bar images created for the ADF Mobile application; the image files within the application controller's `resources` file are retained. Likewise, any images used for application features are also retained.

3.7.1.1 About Migrating Web Service Policy Definitions

MAF stores web service policy definitions in the `wsm-assembly.xml` file. ADF Mobile applications store this information in the `connections.xml` file. Example 3-1 illustrates `oracle/wss_username_token_client_policy` by the `<policy-references>` element in the `connections.xml` file.

Example 3-2 illustrates the policy defined in the `wsm-assembly.xml` file.

Example 3-1 The connections.xml File

```
<policy-references xmlns="http://oracle.com/adf">policy-reference category="security"
                   uri="oracle/wss_username_token_client_policy"
                   enabled="true"
                   id="oracle/wss_username_token_client_policy" xmlns=""/>
</policy-references>
```

Example 3-2 The wsm-assembly.xml File

```
<wsp:PolicyReference xmlns:wsp="http://www.w3.org/ns/ws-policy"
                     DigestAlgorithm="http://www.w3.org/ns/ws-policy/Sha1Exc"
                     URI="oracle/wss_username_token_client_policy"
                     orawsp:status="enabled"
                     orawsp:id="2"/>
```

3.7.2 What You May Need to Know About FARs in Migrated Applications

MAF does not migrate the `adfmf-feature.xml` file packaged within a Feature Archive (FAR) file. You replace the ADF Mobile FARs used by a migrated application to make sure that the `credentials` attribute has been replaced by `securityEnabled=true` in the FAR's `maf-feature.xml` file.

After you migrate the application:

1. Choose **Application Properties > Libraries and Classpath**.
2. Select the FAR and click **Remove**.
3. Import the FAR containing the migrated view controller.
4. Migrate the ADF Mobile application that contains the view controller project that was packaged as a FAR.

Note:

A FAR cannot include both an `adfmf-feature.xml` file and a `maf-feature.xml` file.

---

   a. Deploy the view controller project as a FAR.
   b. Import the FAR into the migrated application.

For more information about how to import a FAR into an application, see the "How to Use FAR Content in a MAF Application" section of *Developing Mobile Applications with Oracle Mobile Application Framework.*

3.8 Configuring your Migrated MAF Application to Use the Full Screen on iOS Devices

MAF applications that you create using the MAF 2.2.0 release and later use the full screen by default on devices running iOS 7 or later.

This means that the iOS device's status bar appears on top of the content rendered by the MAF application. Content from the MAF application appears overlaid by the status icons of the status bar, as shown in Figure 3-6. This happens because the iOS device's status bar's background is transparent. In Figure 3-6, a MAF application's yellow panel header component appears overlaid by the status bar's information about network, time, and battery.

The status bar that renders in an iOS device supports two styles: light and dark. MAF provides APIs to get and set the status bar style on the iOS device so that it renders appropriately when the MAF application renders in the background. Apply the light style to the status bar when the status bar renders on a MAF application with a dark background. Apply the dark style to the status bar when the status bar renders on a light background.

MAF provides the following JavaScript methods to get and set the style of your MAF application on an iOS device:

```
adf.mf.api.getStatusBarStyle = function(callback)
adf.mf.api.setStatusBarStyle = function(style, callback)
```

For more information about these methods, see *JSDoc Reference for Oracle Mobile Application Framework*.

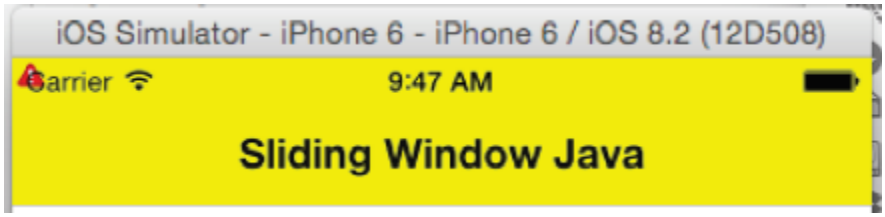
MAF also provides the following Java methods in `oracle.adfmf.framework.api.AdfmfContainerUtilities` that you can use to set the status bar style from a managed bean or lifecycle listener in your MAF application.

```
getStatusBarStyle()
setStatusBarStyle(AdfmfContainerUtilities.STATUS_BAR_STYLE color)
```

For more information about these methods, see *Java API Reference for Oracle Mobile Application Framework*.

The MAF application ignores these methods on non-iOS devices. For more information about using Java and JavaScript APIs in your MAF application, see the "Local HTML and Application Container APIs" appendix in *Developing Mobile Applications with Oracle Mobile Application Framework*.

Figure 3-6 MAF Application Using the Full Screen on an iOS Device



MAF applications migrated to MAF 2.2.2 do not exhibit the just-described behavior. Instead, the iOS device's status bar appears above the MAF application. You can configure a MAF application that you migrate to MAF 2.2.2 to use the full screen on devices running iOS 8 or later.

3.8.1 How to Configure your Migrated MAF Application to Use an iOS Device's Full Screen

You configure a MAF application that you migrate to MAF 2.2.0 or later to use the full screen on a device running iOS 8 or later by setting the `<fullscreenLayout>` element in the `maf-config.xml` file.

To configure a migrated MAF application to use the full screen on an iOS device:

1. In the Applications window, expand the Application Resources panel.
2. In the Application Resources panel, expand **Descriptors** and then **ADF META-INF**.
3. Double-click the **maf-config.xml** file
4. In the Structure window, right-click the **adfmf-config** node and choose **Go to Properties**.

5. In the Properties window, choose `fullscreen` from the **fullscreenLayout** dropdown menu.

3.8.2 What Happens When You Configure your Migrated MAF Application to Use an iOS Device's Full Screen

JDeveloper writes the entry shown in the following example to the `maf-config.xml` file of your migrated MAF application.

Example 3-3 Configuration in maf-config.xml to Render a Migrated MAF Application on the Full Screen of an iOS Device

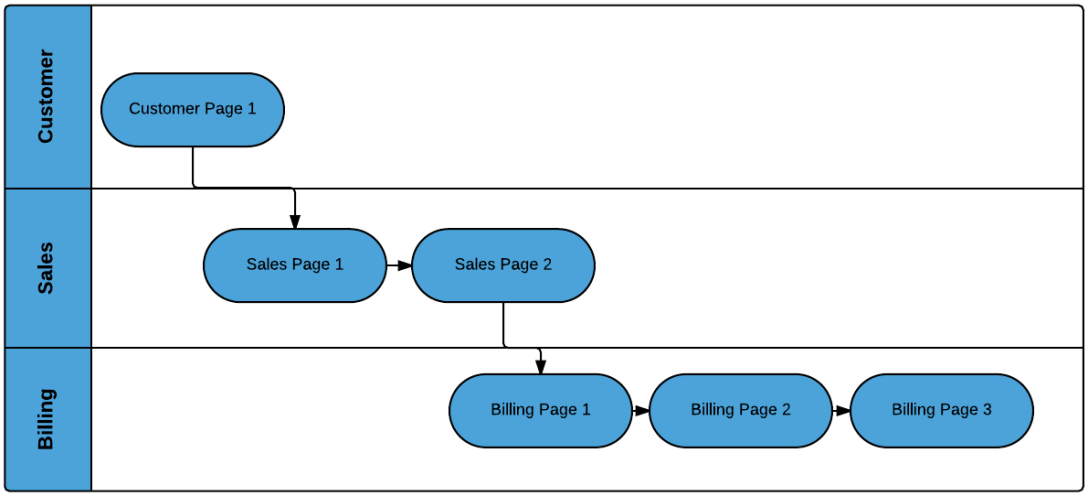
```
<?xml version="1.0" encoding="UTF-8" ?>
  <adfmf-config xmlns="http://xmlns.oracle.com/adf/mf/config">
   ...
  <fullscreenLayout>fullscreen</fullscreenLayout>
</adfmf-config>
```

3.9 Retaining Legacy Behavior When Navigating a MAF Application Using Android's Back Button

MAF 2.2.0 introduces a change in the way that MAF applications created using this release respond to usage of the Android system´s Back button. A MAF application that you created in a previous release and migrate to MAF 2.2.0 or later uses the new behavior.

Figure 3-7 shows a navigation flow on a MAF application where an end user has navigated between three application features (Customer, Sales, and Billing) to the Billing Page 3 page of the Billing application feature.

Figure 3-7 Navigation Flow Between Application Features and Pages in a MAF Application



Prior to Release MAF 2.2.0, the default MAF application behavior in response to an end user tapping Android's system Back button on:

- Billing Page 3 was to navigate to the Sales application feature
- Sales application feature was to navigate to the Customers application feature
- Customer application feature was to close the MAF application

In MAF 2.2.0 and later, the default MAF application behavior in response to an end user tapping Android's system Back button on:

- Billing Page 3 is to navigate to Billing Page 2
- Billing Page 2 is to navigate to Billing Page 1
- Billing Page 1 is to hibernate the MAF application

You can customize how your MAF application responds to an end user´s tap of the Android system´s Back button, as described in the "Navigating a MAF Application Using Android's Back Button" section of the *Developing Mobile Applications with Oracle Mobile Application Framework*.

You can also configure your MAF application to exhibit the pre-MAF 2.2.0 application behavior (navigate between application features) by setting a property in the `maf-config.xml`, as described in How to Retain Pre-MAF 2.2.0 Application Behavior in Response to Usage of Android´s Back Button.

3.9.1 How to Retain Pre-MAF 2.2.0 Application Behavior in Response to Usage of Android´s Back Button

You configure the `legacyBack` element in the `maf-config.xml` file to make your MAF application exhibit pre-MAF 2.2.0 behavior when an end user taps Android´s Back button.

To Retain Pre-MAF 2.2.0 Application Behavior in Response to Usage of Android´s Back Button:

1. In the Applications window, double-click the **maf-config.xml** file.

   By default, this is in the Application Resources pane under the Descriptors and ADF META-INF nodes.

2. In the `maf-config.xml` file, set the value of the `legacyBack` element to `true`, as shown in Example 3-4.

Example 3-4 legacyBack element to Retain Pre-MAF 2.2.0 Application Behavior for Usage of Android Back Button

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

3.10 Migrating to New cacerts File for SSL in MAF 2.2.2

MAF 2.1.0 delivered a new `cacerts` file for use in MAF applications. Make sure that the `cacerts` file packaged in the MAF application that you publish for your end users to install contains the same CA root certificates as the HTTPS server that end users connect to when they use your MAF application.

You may need to import new certificates to your MAF application's `cacerts` file if the HTTPS server contains certificates not present in your MAF application's `cacerts` file. Similarly, system administrators for the HTTPS servers that your MAF application connects to may need to import new certificates if your MAF application uses a certificate not present on the HTTPS server.

Use JDK 8's `keytool` utility to view and manage the certificates in your MAF application's `cacerts` file. The following example demonstrates how you might use JDK 8's `keytool` utility to display the list of certificates in a `cacerts` file:

```
JDK8install/bin/keytool -list -v -keystore dirPathToCacertsFile/
cacerts –storepass changeit | grep "Issuer:"
```

For more information about using the JDK 8's `keytool` utility to manage certificates, see http://docs.oracle.com/javase/8/docs/technotes/tools/#security. For example, to use the `keytool` utility on Windows, see http://docs.oracle.com/javase/8/docs/technotes/tools/windows/keytool.html. For UNIX-based operating systems, see http://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html.

For more information about the `cacerts` file and using SSL to secure your MAF application, see the "Supporting SSL" section in *Developing Mobile Applications with Oracle Mobile Application Framework*.

Example 3-5 lists the issuers of CA root certificates included in MAF 2.1.0's `cacerts` file. Use JDK 8's `keytool` utility, as previously described, to manage the certificates in this file to meet the requirements of the environment where your MAF application will be used.

Example 3-5 CA Root Certificate Issuers in MAF 2.1.0 cacerts File

```
Issuer: CN=DigiCert Assured ID Root CA, OU=www.digicert.com, O=DigiCert Inc, C=US
Issuer: CN=TC TrustCenter Class 2 CA II, OU=TC TrustCenter Class 2 CA, O=TC TrustCenter GmbH, C=DE
Issuer: EMAILADDRESS=premium-server@thawte.com, CN=Thawte Premium Server CA, OU=Certification
Services Division, O=Thawte Consulting cc, L=Cape Town, ST=Western Cape, C=ZA
Issuer: CN=SwissSign Platinum CA - G2, O=SwissSign AG, C=CH
Issuer: CN=SwissSign Silver CA - G2, O=SwissSign AG, C=CH
Issuer: EMAILADDRESS=server-certs@thawte.com, CN=Thawte Server CA, OU=Certification Services
Division, O=Thawte Consulting cc, L=Cape Town, ST=Western Cape, C=ZA
Issuer: CN=Equifax Secure eBusiness CA-1, O=Equifax Secure Inc., C=US
Issuer: CN=SecureTrust CA, O=SecureTrust Corporation, C=US
Issuer: CN=UTN-USERFirst-Client Authentication and Email, OU=http://www.usertrust.com, O=The
USERTRUST Network, L=Salt Lake City, ST=UT, C=US
Issuer: EMAILADDRESS=personal-freemail@thawte.com, CN=Thawte Personal Freemail CA, OU=Certification
Services Division, O=Thawte Consulting, L=Cape Town, ST=Western Cape, C=ZA
Issuer: CN=AffirmTrust Networking, O=AffirmTrust, C=US
Issuer: CN=Entrust Root Certification Authority, OU="(c) 2006 Entrust, Inc.", OU=www.entrust.net/CPS
is incorporated by reference, O="Entrust, Inc.", C=US
Issuer: CN=UTN-USERFirst-Hardware, OU=http://www.usertrust.com, O=The USERTRUST Network, L=Salt Lake
City, ST=UT, C=US
Issuer: CN=Certum CA, O=Unizeto Sp. z o.o., C=PL
Issuer: CN=AddTrust Class 1 CA Root, OU=AddTrust TTP Network, O=AddTrust AB, C=SE
Issuer: CN=Entrust Root Certification Authority - G2, OU="(c) 2009 Entrust, Inc. - for authorized use
only", OU=See www.entrust.net/legal-terms, O="Entrust, Inc.", C=US
Issuer: OU=Equifax Secure Certificate Authority, O=Equifax, C=US
Issuer: CN=QuoVadis Root CA 3, O=QuoVadis Limited, C=BM
Issuer: CN=QuoVadis Root CA 2, O=QuoVadis Limited, C=BM
Issuer: CN=DigiCert High Assurance EV Root CA, OU=www.digicert.com, O=DigiCert Inc, C=US
Issuer: EMAILADDRESS=info@valicert.com, CN=http://www.valicert.com/, OU=ValiCert Class 1 Policy
Validation Authority, O="ValiCert, Inc.", L=ValiCert Validation Network
Issuer: CN=Equifax Secure Global eBusiness CA-1, O=Equifax Secure Inc., C=US
Issuer: CN=GeoTrust Universal CA, O=GeoTrust Inc., C=US
Issuer: OU=Class 3 Public Primary Certification Authority, O="VeriSign, Inc.", C=US
Issuer: CN=thawte Primary Root CA - G3, OU="(c) 2008 thawte, Inc. - For authorized use only",
OU=Certification Services Division, O="thawte, Inc.", C=US
Issuer: CN=thawte Primary Root CA - G2, OU="(c) 2007 thawte, Inc. - For authorized use only",
O="thawte, Inc.", C=US
Issuer: CN=Deutsche Telekom Root CA 2, OU=T-TeleSec Trust Center, O=Deutsche Telekom AG, C=DE
```

```
Issuer: CN=Buypass Class 3 Root CA, O=Buypass AS-983163327, C=NO
Issuer: CN=UTN-USERFirst-Object, OU=http://www.usertrust.com, O=The USERTRUST Network, L=Salt Lake
City, ST=UT, C=US
Issuer: CN=GeoTrust Primary Certification Authority, O=GeoTrust Inc., C=US
Issuer: CN=Buypass Class 2 Root CA, O=Buypass AS-983163327, C=NO
Issuer: CN=Baltimore CyberTrust Code Signing Root, OU=CyberTrust, O=Baltimore, C=IE
Issuer: OU=Class 1 Public Primary Certification Authority, O="VeriSign, Inc.", C=US
Issuer: CN=Baltimore CyberTrust Root, OU=CyberTrust, O=Baltimore, C=IE
Issuer: OU=Starfield Class 2 Certification Authority, O="Starfield Technologies, Inc.", C=US
Issuer: CN=Chambers of Commerce Root, OU=http://www.chambersign.org, O=AC Camerfirma SA CIF
A82743287, C=EU
Issuer: CN=T-TeleSec GlobalRoot Class 3, OU=T-Systems Trust Center, O=T-Systems Enterprise Services
GmbH, C=DE
Issuer: CN=VeriSign Class 3 Public Primary Certification Authority - G5, OU="(c) 2006 VeriSign, Inc.
- For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US
Issuer: CN=T-TeleSec GlobalRoot Class 2, OU=T-Systems Trust Center, O=T-Systems Enterprise Services
GmbH, C=DE
Issuer: CN=TC TrustCenter Universal CA I, OU=TC TrustCenter Universal CA, O=TC TrustCenter GmbH, C=DE
Issuer: CN=VeriSign Class 3 Public Primary Certification Authority - G4, OU="(c) 2007 VeriSign, Inc.
- For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US
Issuer: CN=VeriSign Class 3 Public Primary Certification Authority - G3, OU="(c) 1999 VeriSign, Inc.
- For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US
Issuer: CN=XRamp Global Certification Authority, O=XRamp Security Services Inc,
OU=www.xrampsecurity.com, C=US
Issuer: CN=Class 3P Primary CA, O=Certplus, C=FR
Issuer: CN=Certum Trusted Network CA, OU=Certum Certification Authority, O=Unizeto Technologies S.A.,
C=PL
Issuer: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 3
Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US
Issuer: CN=GlobalSign, O=GlobalSign, OU=GlobalSign Root CA - R3
Issuer: CN=UTN - DATACorp SGC, OU=http://www.usertrust.com, O=The USERTRUST Network, L=Salt Lake
City, ST=UT, C=US
Issuer: OU=Security Communication RootCA2, O="SECOM Trust Systems CO.,LTD.", C=JP
Issuer: CN=GTE CyberTrust Global Root, OU="GTE CyberTrust Solutions, Inc.", O=GTE Corporation, C=US
Issuer: OU=Security Communication RootCA1, O=SECOM Trust.net, C=JP
Issuer: CN=AffirmTrust Commercial, O=AffirmTrust, C=US
Issuer: CN=TC TrustCenter Class 4 CA II, OU=TC TrustCenter Class 4 CA, O=TC TrustCenter GmbH, C=DE
Issuer: CN=VeriSign Universal Root Certification Authority, OU="(c) 2008 VeriSign, Inc. - For
authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US
Issuer: CN=GlobalSign, O=GlobalSign, OU=GlobalSign Root CA - R2
Issuer: CN=Class 2 Primary CA, O=Certplus, C=FR
Issuer: CN=DigiCert Global Root CA, OU=www.digicert.com, O=DigiCert Inc, C=US
Issuer: CN=GlobalSign Root CA, OU=Root CA, O=GlobalSign nv-sa, C=BE
Issuer: CN=thawte Primary Root CA, OU="(c) 2006 thawte, Inc. - For authorized use only",
OU=Certification Services Division, O="thawte, Inc.", C=US
Issuer: CN=Starfield Root Certificate Authority - G2, O="Starfield Technologies, Inc.", L=Scottsdale,
ST=Arizona, C=US
Issuer: CN=GeoTrust Global CA, O=GeoTrust Inc., C=US
Issuer: CN=Sonera Class2 CA, O=Sonera, C=FI
Issuer: CN=Thawte Timestamping CA, OU=Thawte Certification, O=Thawte, L=Durbanville, ST=Western Cape,
C=ZA
Issuer: CN=Sonera Class1 CA, O=Sonera, C=FI
Issuer: CN=QuoVadis Root Certification Authority, OU=Root Certification Authority, O=QuoVadis
Limited, C=BM
Issuer: CN=AffirmTrust Premium ECC, O=AffirmTrust, C=US
Issuer: CN=Starfield Services Root Certificate Authority - G2, O="Starfield Technologies, Inc.",
L=Scottsdale, ST=Arizona, C=US
Issuer: EMAILADDRESS=info@valicert.com, CN=http://www.valicert.com/, OU=ValiCert Class 2 Policy
Validation Authority, O="ValiCert, Inc.", L=ValiCert Validation Network
Issuer: CN=AAA Certificate Services, O=Comodo CA Limited, L=Salford, ST=Greater Manchester, C=GB
Issuer: CN=America Online Root Certification Authority 2, O=America Online Inc., C=US
```

```
Issuer: CN=AddTrust Qualified CA Root, OU=AddTrust TTP Network, O=AddTrust AB, C=SE
Issuer: CN=KEYNECTIS ROOT CA, OU=ROOT, O=KEYNECTIS, C=FR
Issuer: CN=America Online Root Certification Authority 1, O=America Online Inc., C=US
Issuer: CN=VeriSign Class 2 Public Primary Certification Authority - G3, OU="(c) 1999 VeriSign, Inc.
- For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US
Issuer: CN=AddTrust External CA Root, OU=AddTrust External TTP Network, O=AddTrust AB, C=SE
Issuer: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 2
Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US
Issuer: CN=GeoTrust Primary Certification Authority - G3, OU=(c) 2008 GeoTrust Inc. - For authorized
use only, O=GeoTrust Inc., C=US
Issuer: CN=GeoTrust Primary Certification Authority - G2, OU=(c) 2007 GeoTrust Inc. - For authorized
use only, O=GeoTrust Inc., C=US
Issuer: CN=SwissSign Gold CA - G2, O=SwissSign AG, C=CH
Issuer: CN=Entrust.net Certification Authority (2048), OU=(c) 1999 Entrust.net Limited,
OU=www.entrust.net/CPS_2048 incorp. by ref. (limits liab.), O=Entrust.net
Issuer: OU=ePKI Root Certification Authority, O="Chunghwa Telecom Co., Ltd.", C=TW
Issuer: CN=Global Chambersign Root - 2008, O=AC Camerfirma S.A., SERIALNUMBER=A82743287, L=Madrid
(see current address at www.camerfirma.com/address), C=EU
Issuer: CN=Chambers of Commerce Root - 2008, O=AC Camerfirma S.A., SERIALNUMBER=A82743287, L=Madrid
(see current address at www.camerfirma.com/address), C=EU
Issuer: OU=Go Daddy Class 2 Certification Authority, O="The Go Daddy Group, Inc.", C=US
Issuer: CN=AffirmTrust Premium, O=AffirmTrust, C=US
Issuer: CN=VeriSign Class 1 Public Primary Certification Authority - G3, OU="(c) 1999 VeriSign, Inc.
- For authorized use only", OU=VeriSign Trust Network, O="VeriSign, Inc.", C=US
Issuer: OU=Security Communication EV RootCA1, O="SECOM Trust Systems CO.,LTD.", C=JP
Issuer: OU=VeriSign Trust Network, OU="(c) 1998 VeriSign, Inc. - For authorized use only", OU=Class 1
Public Primary Certification Authority - G2, O="VeriSign, Inc.", C=US
Issuer: CN=Go Daddy Root Certificate Authority - G2, O="GoDaddy.com, Inc.", L=Scottsdale, ST=Arizona,
C=US
```