Using webMethods Mobile Designer

Version 9.8

April 2015

This document applies to webMethods Mobile Designer Version 9.8 and to all subsequent releases.

Specifications contained herein are subject to change and these changes will be reported in subsequent release notes or new editions.

Document ID: MD-UG-98-20150415

Table of Contents

About this Guide

This guide contains the information about using webMethods Mobile Designer to create mobile applications for multiple device platforms.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention *folder.subfolder:service* .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
Italic	Identifies variables for which you must supply values specific to your own situation or environment. Identifies new terms the first time they occur in the text.
`Monospace font`	Identifies text you must type or messages displayed by the system.
{ }	Indicates a set of choices from which you must choose one. Type only the information inside the curly braces. Do not type the { } symbols.
|	Separates two mutually exclusive choices in a syntax line. Type one of these choices. Do not type the | symbol.
[ ]	Indicates one or more options. Type only the information inside the square brackets. Do not type the [ ] symbols.
...	Indicates that you can type multiple options of the same type. Type only the information. Do not type the ellipsis (...).

Online Information

Software AG Documentation Website

You can find documentation on the Software AG Documentation website at http://documentation.softwareag.com. The site requires Empower credentials. If you do not have Empower credentials, you must use the TECHcommunity website.

Software AG Empower Product Support Website

You can find product information on the Software AG Empower Product Support website at https://empower.softwareag.com.

To submit feature/enhancement requests, get information about product availability, and download products, go to Products.

To get information about fixes and to read early warnings, technical papers, and knowledge base articles, go to the Knowledge Center.

Software AG TECHcommunity

You can find documentation and other technical information on the Software AG TECHcommunity website at http://techcommunity.softwareag.com. You can:

- Access product documentation, if you have TECHcommunity credentials. If you do not, you will need to register and specify "Documentation" as an area of interest.
- Access articles, code samples, demos, and tutorials.
- Use the online discussion forums, moderated by Software AG professionals, to ask questions, discuss best practices, and learn how other customers are using Software AG technology.
- Link to external websites that discuss open standards and web technology.

I Getting Started

1 About Mobile Designer

webMethods Mobile Designer Overview

webMethods Mobile Designer provides a set of standardized coding abstraction layers, processes, and utilities that help you to develop mobile applications and port the applications across multiple platforms.

Mobile Designer also includes a cross compiler that you can use to compile the same source code so that it can run on a wide variety of platforms. You do not have to wait until the post-development process to apply the requirements for all mobile platforms required. You can apply cross-compilation requirements throughout the development process and launch a multi-platform solution simultaneously.

Mobile Designer Build-Time Component

The Mobile Designer build-time component is a standard build script that each project uses for each device to:

- Compile and compact the resources including data-arrays, text, audio, images, fonts, and palettes.
- Create the parameters a specific build uses.
- Replace (hot-swap) the generic code in the underlying run-time Mobile Designer code branches with code specific to the device.
- Reference any project-specific code branches, and compilation stubs or inclusion libraries.
- Compile the code.
- Optionally obfuscate the code.
- Package, build, and sign the final binary.

Mobile Designer Cross Compiler and Run-time Classes

In Mobile Designer you can cross compile your mobile application's Java code into C++, C#, or Java. Mobile Designer can then compile the application code using the native tools that were installed with the target platform's SDKs. Mobile Designer then links the compiled code to the libraries installed with the SDK. When building an application for the target platform, the relevant target-platform SDKs must be installed and configured. For more information, see "Setting Up Platforms" on page 27.

You can use the run-time classes described in the *webMethods Mobile Designer Java API Reference* to provide a wide array of features found on mobile devices. To handle device-specific differences, Mobile Designer provides the device profile database, the classes, and abstraction layers, to provide a consistent base for building your application. The

abstraction layers provide the ability to load images and sound, detect interruptions, and handle text, as well as other functions, such as to establish an HTTP connection.

Mobile Designer Device Profiler

Mobile Designer comes with profile settings for many devices. Mobile Designer maintains the settings for each device in its device database. The settings for each device are in individual profile XML files in the following directory:

Mobile Designer_directory/Devices

The device profile XML files are grouped by platforms and manufacturer within this directory.

If you want an application to support a device for which Mobile Designer does not provide settings, you can use the Device Profiler sample application. You can find the Device Profiler sample application in the following location:

Mobile Designer_directory/Samples/_DeviceProfiler_

You can run the Device Profiler sample application on any supported mobile platform. The Device Profiler provides a set of tools that you use to discover the appropriate settings for devices. For more information, see "Determining Device Settings by Running the Device Profiler" on page 91.

After determining the settings for a device, you can add the profile settings to the Mobile Designer device database. For more information, see "Adding a Device Profile to the Device Databases" on page 97.

Parameter-Driven Projects

Mobile Designer projects use parameters and properties to simplify the process of including and/or excluding features based on a target device. Mobile Designer predefines some parameter and property values, for example, the profile settings for devices. You can override the predefined settings for a specific mobile application project and/or for a specific target device.

When creating an application, typically you have common logic that works for all target devices. However, you might require branches in the logic to address the needs of a specific target device. For example, you might need to omit or alter a feature for a target device, or you might need to position an image relative to the screen dimensions for a target device. To accommodate device-specific logic, your application logic can branch based on parameter values that are set using the device profile settings.

Properties and parameters also drive how Mobile Designer builds applications for a target device. Mobile Designer drives builds using a combination of properties, parameters, and paths, all of which you can customize and override. For example, these properties, parameters, and paths control stubs against which to compile, packagers to use when making the final binaries, details about screen dimensions, the sound

APIs and settings that are most appropriate for a target device, and other general and application-specific settings.

2 Configuring Mobile Designer

About Configuring Mobile Designer

You configure Mobile Designer by altering the properties located in the following file:

Mobile Designer_directory/sdk.properties

The properties in the sdk.properties file apply globally to all Mobile Designer projects. See the following for more information:

- **For instructions for how to update and add properties to the sdk.properties file**, see "Updating the sdk.properties File to Configure Mobile Designer" on page 18.
- **For a description of the configuration properties**, see "Mobile Designer Configuration Properties (sdk.properties)" on page 19.

Note: You can set properties for a project that override the settings in the sdk.properties file.

Updating the sdk.properties File to Configure Mobile Designer

The sdk.properties file is an ASCII text file that you can edit with any text editor. Update the sdk.properties file with name value pairs, using the following format:

```
property=value
```

Note: The sdk.properties file contains default values for some properties. If the `value` you need to specify for a property is a path, a best practice is to use the convention shown in the default unless you are working on a different operating system. For example, if a path uses c:\\a\\b\\c it is best to specify the path using that notation rather than c:/a/b/c. Although both notations are technically correct, some third-party tools might encounter issues.

When updating Windows paths use a forward slash character (/) or an escaped slash character (\\) in the properties file.

To add or update properties in the sdk. properties file

1. Open the sdk.properties file in your Mobile Designer installation directory using a text editor.
2. Locate the property you want to update, or add a property if it does not already exist. For more information about the properties you can specify and valid values, see "Mobile Designer Configuration Properties (sdk.properties)" on page 19.
3. **Save** and close the sdk.properties file.

Mobile Designer Configuration Properties (sdk.properties)

The properties in the sdk.properties file are settings that apply globally to all Mobile Designer projects.

JAD and Manifest Files

Use the JAD and manifest files properties to configure the vendor name and URL that Mobile Designer uses when creating project and metadata files.

Although the JAD and manifest files properties have default values, you should set values for these properties to specify information for your own organization.

Tip: You can override the JAD and manifest files properties on a project-by-project basis, such as when creating ported builds on an application for a separate development company. To do so, specify the JAD and manifest files properties in your projects targets/_defaults_.xml file. For more information about setting project properties, see "Setting Project Properties" on page 119.

project.jad.vendor.name

Specifies the vendor name included in project and/or application metadata files that are bundled with the final binary. This is usually used as part of the data that uniquely identifies an application in an App store.

Value Vendor name.

Default `Software AG`

Example `project.jad.vendor.name=My Company`

project.jad.vendor.url

Specifies the URL to the vendor's website.

Value URL

Default `http://www.softwareag.com`

Example `project.jad.vendor.url=http://www.mycompany.com`

Java Compiler Check

When Mobile Designer executes an Ant target, the Java compiler is usually required. Use the `mobiledesigner.javac.detection.mode` property to specify whether Mobile Designer checks for the Java compiler when executing an Ant target and the action to take if the Java compiler is not present.

mobiledesigner.javac.detection.mode

Specifies whether you want Mobile Designer to check whether the Java compiler is present in a user's currently configured version of Java when the user executes a Mobile Designer Ant target. Mobile Designer checks for javac.exe on Windows or javac on Macintosh.

Value One of the following:

- `fail` if you want Mobile Designer to check for the Java compiler when a user executes an Ant target. If the Java compiler is not found, Mobile Designer immediately stops the running Ant target with an error message.
- `warn` if you want Mobile Designer to check for the Java compiler when a user executes an Ant target. If the Java compiler is not found, Mobile Designer displays a warning message and continues to execute the Ant target.
- `none` if you want Mobile Designer to execute the Ant target without checking for the Java compiler.

Default `warn`

Example `mobiledesigner.javac.detection.mode=fail`

KZip Property

You can configure Mobile Designer to use KZip for JAR compression. When KZip is not available, Mobile Designer uses 7Zip as the default JAR packager.

You can download the Windows version of KZip from Ken Silverman's Utility Page. You can download the Linux and Mac OS X version from JonoF's Games and Stuff.

j2me.packager.kzip

Specifies the location of the KZip executable.

Value Path to the installed KZip executable.

Default `C:/Program Files/KZip/kzip.exe`

Example `j2me.packager.kzip=D:/Program Files/KZip/kzip.exe`

Localization Property

You can configure the language you want Mobile Designer to use for text in dialogs and samples. This configuration affects Mobile Designer dialogs, such as the Activate-Handset and Multi-Build dialogs.

mobiledesigner.locale

Specifies the language code.

Value One of the following language codes:

- `en` (English)
- `zh` (Chinese)
- `fr` (French)
- `de` (German)
- `iw` (Hebrew)
- `ja` (Japanese)
- `pl` (Polish)
- `ru` (Russian)
- `es` (Spanish)
- `tr` (Turkish)

Default `en`

Example `mobiledesigner.locale=de`

Proguard Obfuscator Settings

Use the Proguard obfuscator settings to provide information about your installed version of Proguard if you want to use Proguard for Android. Mobile Designer only supports obfuscation using Proguard for Android.

Note: If you have the Android SDK installed, Proguard is included in the SDK.

Mobile Designer has been tested with Proguard versions 3.7 through to 5.1.

Caution: Proguard version 4.0.1 and later includes method collapsing. If your application has collapsible methods, this might cause problems if a device has built in method size limits. If this is the case, you can change parameters to use an earlier version of Proguard or different execution parameters for that particular device.

proguard.library.root

Specifies the Proguard directory that contains the proguard.jar file.

Value Path to the Proguard directory.

Default None.

Example `proguard.library.root=C:/Proguard/lib`

obfuscator.proguard.version

Specifies the Proguard configuration and mapping you want to use.

Value Configuration and mapping from the Tools/Proguard folder.

Default One of the following:

- Proguard version 4.7 if the version supplied with the Android SDK is available
- Otherwise:
  - If building for the Android platform, Proguard version 4.3
  - If building for another platform, Proguard version 4.0.1

Example `obfuscator.proguard.version=proguard_4.7`

obfuscator.proguard.library.filename

Specifies the name of your Proguard library.

Value Proguard library name without the .jar extension.

Default None

Example `obfuscator.proguard.library.filename=proguard`

obfuscator

Enables or disables use of the obfuscator.

Value
- `proguard` if you want to use Proguard as the obfuscator.
- `none` to disable the obfuscator.

Default `none`

Example `obfuscator=proguard`

Project Build Settings

You can specify the name of the project directory to use for the output of builds.

project.build.dir.rel.root

Specifies the name of the folder that you want to use for the output of builds. This folder is relative to a project's base directory.

Value Folder name to use for the output of project builds.

Default `Builds`

Example `project.build.dir.rel.root=Output`

Proxy Settings

Configure proxy settings if you need to use a proxy server to connect to the Internet.

Note: The Software AG Installer prompts for proxy settings during installation and the values provided at installation are saved to the sdk.properties file.

proxy.hostname

Specifies the proxy server host name.

Value Host name of the proxy server.

Default No default.

Example `proxy.hostname=proxyserver`

proxy.port

Specifies the proxy server port number.

Value Port number for the proxy server.

Default No default.

Example `proxy.port=1080`

proxy.username

Specifies the user name of a user with authority to connect to the proxy server.

Value User name

Default No default

Example `proxy.username=Administrator`

proxy.password

Specifies the password associated with the user name specified in the `proxy.username` property.

Value Password for the proxy server.

Default No default.

Example `proxy.password=secret`

Platform-Specific Properties

You must configure the location of the third-party SDKs and compilers that you want Mobile Designer to use when producing mobile application bundles. The following table lists where you can find information about the required configuration for each platform.

Note: For a list of supported SDKs, see "Supported SDK Versions" on page 29.

Platform	Description
Android	"Configuring Mobile Designer for the Android SDK" on page 38
iOS	"Configuring Mobile Designer for the iOS Platform" on page 48
Windows Phone 8	"Configuring Mobile Designer for the Windows Phone 8 Platform" on page 55
Windows RT Windows 8	"Configuring Mobile Designer for the Windows RT or Windows 8 Platform" on page 59

Environment Variable for Mobile Designer

With previous versions, Mobile Designer created an environment variable, `MOBILE_DESIGNER`, on the computer when Mobile Designer was installed. As a result, the installation process set the value of `MOBILE_DESIGNER` to the location of the latest Mobile Designer installed instance. As of version 9.8, this environment variable is considered deprecated and is no longer created by default. However, you can continue to use it if you want. If you wish to continue using this feature, you will need to create/alter the variable yourself in the indicated locations. If you do not want to use the `MOBILE_DESIGNER` environment variable any more, you can disable it. Then use the following when calling Ant from the command line or IDE:

```
-Denv.MOBILE_DESIGNER=Mobile Designer_dir
```

There are 3 different scenarios:

- You are already using Mobile Designer with the Mobile Development view (through the Software AG Designer) to create applications, and you want to replace your 9.7 installation with 9.8. In this case, you should ensure that the environment variable `MOBILE_DESIGNER` is unset manually before using Mobile Designer 9.8.
- You want to continue using Mobile Designer 9.8 with your current applications that are not created through Mobile Development and you want to continue with your own choice of build environment (another IDE, Jenkins, etc.). Although the `MOBILE_DESIGNER` environment variable is now considered deprecated, you may continue to use it for this release if you wish. Alternatively, you may wish to disable this variable as described above.
- You want to continue developing an application created outside of the Mobile Development view, but you also want to make use of the Mobile Development view for your projects. In this case, you should ensure that the `MOBILE_DESIGNER` environment variable is unset manually before using Mobile Designer 9.8.

Mobile Designer uses the environment variable at run time to determine its location so that it can locate Mobile Designer-specific information, for example, the sdk.properties file.

If a developer has multiple Mobile Designer installations on a single machine, the developer can set the `MOBILE_DESIGNER` environment variable to indicate the current instance of Mobile Designer to use.

The following indicates where you can find the environment variable:

- On Windows, in **System Properties > Advanced > Environment Variables** in the **User variables** group
- On Macintosh:
  - OSX 10.7 or earlier in the ~/.MacOSX/environment.plist file
  - Later versions in /etc/launchd.conf

II Setting Up Platforms

3 Supported SDK Versions

SDK Versions that Mobile Designer Supports

The following table lists the SDK versions that Mobile Designer supports for each platform.

Platform SDK	Supported versions
Android SDK	2.3.3 through 5.0
iOS SDK	6.0 through 8.1.x
Windows Phone 8 SDK	8.0 and 8.1
Windows 8 (x86 architecture version) Windows RT (ARM architecture version) **Note:** Only Windows Store/Metro applications are supported.	8.0 and 8.1

4 Setting Up the Android Platform

About Setting Up the Android Platform

Before you can use Mobile Designer to build applications for the Android platform, you need to set up your environment. The following table lists required and optional tasks to set up the environment.

Setup	Required or Optional	Tasks
Development Environment	Required	Install the Android stand-alone SDK. For more information, see: ■ "Installing the Android SDK on Windows" on page 33 ■ "Installing the Android SDK on Macintosh" on page 34
	Optional	Install the ADT Eclipse plug-in if you want to use Eclipse to develop Android applications and if you want to manage virtual devices and debug applications via Eclipse. For more information, see "Installing the Android Development Tools Eclipse Plug-In" on page 36
Mobile Designer Configuration	Required	Update the Mobile Designer sdk.properties file to provide information about the installed Android SDK. For more information, see "Configuring Mobile Designer for the Android SDK" on page 38
Emulators	Optional	Define Android emulators if you want to test Android applications on emulated devices. For more information, see "Setting Up an Android Virtual Device (Emulator)" on page 39.
	Optional	Specify a proxy server to use if the Android emulator must access the Internet through a proxy server. For more information, see "Using the Android Emulator with a Proxy Server" on page 40

The procedures in this documentation do *not* cover all possible setups and scenarios. Refer to the Android Developer website at http://developer.android.com/index.html for further details.

Installing the Android SDK on Windows

If you use a Windows personal computer, use this procedure to install the SDK so that you can use it with Mobile Designer.

To install the Android SDK on Windows

1. Open the following webpage in a browser: http://developer.android.com/sdk/index.html.
2. On the Android website, locate the system requirements and review them to ensure that your Windows environment meets the requirements for the Android SDK.
3. Locate the installer you want to use.

   You can install either the standalone SDK or the full bundle, which includes Eclipse. If you are using Software AG Designer or other IDE and only need to install the standalone SDK, click **USE AN EXISTING IDE** to reveal information about the standalone SDK.

   Important: Install the 32-bit version. The 32-bit JDK is generally required for compatibility.

4. Run the Android SDK installer for the Windows platform.
5. When prompted for the install location, set the destination folder to: `c:\android-sdk-windows`

   When the installer completes, it starts the Android SDK Manager.
6. In the Android SDK Manager, select the following:
   - Tools folder to select all Android SDK tools
   - Android SDK 5.0 (API 21)
   - Other Android SDK versions you want to ensure compatibility

     Note: For a list of supported SDKs, see "SDK Versions that Mobile Designer Supports" on page 30.

   - Optionally, any other SDK versions that you might need.

     Note: At a later time if you decide you need another SDK version, you can download and install it by running the Android SDK Manager in the C:\android-sdk-windows directory.

- Latest Android Support Library
- Latest Google Play services
- Google Cloud Messaging for Android Library

> **Note:** This is only visible if the "Obsolete" option is selected in the Android SDK Manager.

7. Install the packages.

   When prompted, review the licensing and packaging dependencies. If you agree, accept all the licenses and continue the installation.

   The Android SDK Manager downloads and installs the APIs you selected.

8. If the Android SDK Manager prompts you to restart the ADB (Android Debug Bridge) command-line tool, select **Yes**.

> **Note:** The ADB tool is a tool you can run manually and also has a background service component that manages communications to and from Android devices, both virtual and physical. When installing the Android SDK, the Android SDK Manager restarts the ADB tool to restart the background service.

Installing the Android SDK on Macintosh

If you use a Macintosh, use this procedure to install the SDK so that you can use it with Mobile Designer.

To install the Android SDK on Macintosh

1. Open the following webpage in a browser: http://developer.android.com/sdk/index.html.
2. On the Android website, locate the system requirements and review them to ensure that your Macintosh environment meets the requirements for the Android SDK.
3. Locate the installer you want to use.

   You can install either the standalone SDK or the full bundle, which includes Eclipse. If you are using Software AG Designer or other IDE and only need to install the standalone SDK, click **USE AN EXISTING IDE** to reveal information about the standalone SDK.

> **Important:** Install the 32-bit version. The 32-bit JDK is generally required for compatibility.

4. Download the .zip file containing the install files for the Mac OS X platform to your desktop.

5. Double-click the .zip file to open it with the Archive utility and extract its contents.

   This creates a new folder named android-sdk-macosx on your desktop.

6. Perform the following to copy the android-sdk-macosx folder to the root of the hard disk to ensure that the SDK is available to all users.

   Note: You might require administrator privileges to write to the root of the hard disk.

   a. Start the Terminal application, which is in the Applications/Utilities folder.

   b. Using the Terminal application, issue the following commands, substituting *username* with your Macintosh user name:

   ```
   sudo su -
   cd /
   cp /Users/username/Desktop/android-sdk-macosx /
   chmod -R a+rw android-sdk-macosx
   exit
   ```

   Note: If you do not know the Macintosh user name, you can determine it by using the `whoami` command in the Terminal application.

   You can use the Tab key to help auto-complete paths.

   This series of commands does the following:

   - The `cp` command copies the android-sdk-macosx folder from your desktop and places it in the root of the hard disk.
   - The `chmod` command makes the android-sdk-macosx folder readable and writeable to all users.

     Note: You may wish to consider a more restrictive set of permissions than this.

   - The `exit` detaches from the root shell.

   The remaining steps in the procedure do not require super-user privileges.

7. Start the start the Android SDK Manager. To do so, using the Terminal application, issue the following commands;

   ```
   cd /android-sdk-macosx
   ./tools/android
   ```

8. In the Android SDK Manager, select the following:

   - Tools folder to select all Android SDK tools
   - Latest version of the Android SDK
   - Other Android SDK versions you want to ensure compatibility

   Note: For a list of supported SDKs, see "SDK Versions that Mobile Designer Supports" on page 30.

- Optionally, any other SDK versions that you might need.

  Note: At a later time if you decide you need another SDK version, you can download and install it by running the Android SDK Manager again.

- Latest Android Support Library
- Latest Google Play services
- Google Cloud Messaging for Android Library

  Note: This is only visible if the "Obsolete" option is selected in the Android SDK Manager.

9. Install the packages.

   When prompted, review the licensing and packaging dependencies. If you agree, accept all the licenses and continue the installation.

   The Android SDK Manager downloads and installs the APIs you selected.

10. If the Android SDK Manager prompts you to restart the ADB (Android Debug Bridge) command-line tool, select **Yes**.

    Note: The ADB tool is a tool you can run manually and also has a background service component that manages communications to and from Android devices, both virtual and physical. When installing the Android SDK, the Android SDK Manager restarts the ADB tool to restart the background service.

11. Delete the .zip file that you downloaded from the Android Developer website and the android-sdk-macosx folder from your desktop.

Installing the Android Development Tools Eclipse Plug-In

You can optionally install the Android Development Tools (ADT) Eclipse plug-in. Install the ADT Eclipse plug-in if you want to use Eclipse to develop Android applications. You also use the ADT Eclipse plug-in to manage virtual devices.

Note: If you installed the version of the Android SDK that comes bundled with Eclipse rather than the standalone SDK, the ADT was also installed. You do not need to install it again.

This section provides only the minimal information for installing the ADT Eclipse plug-in. For additional details, refer to information about the ADT Eclipse plug-in on the Android Developer website at http://developer.android.com/sdk/eclipse-adt.html and http://developer.android.com/sdk/installing/installing-adt.html.

Important: You must install the Android SDK before you can install the ADT Eclipse plug-in.

Caution: It may also be possible to install the Android SDK plug-in through Software AG Designer, but this is currently not supported by Mobile Designer - do this at your own risk.

To install the ADT Eclipse plug-in

The description below was valid at the time of the release. It is recommended that you also check the http://developer.android.com/sdk/installing/installing-adt.html website for any changes in the process.

1. If you do not have Eclipse Classic installed, install it.
   a. Download the Eclipse install files from http://www.eclipse.org/downloads/.
   b. Extract the .zip file to the location where you want to install Eclipse.
2. Start Eclipse if it is not already started.

   Note: If you have Eclipse installed under Program Files (x86), you need to start Eclipse as administrator to install and update Android tools.
3. Save and close all files you have open in Eclipse. You will need to restart Eclipse during this procedure to install the ADT Eclipse plug-in.
4. From Eclipse, select **Help > Install New Software**.
5. In the Install window, click **Add** to add a new software repository.
6. In the Add Repository window, specify the following information, then click **OK**:

In this field...	Specify...
Name	`Eclipse ADT Plug-in`
Location	`https://dl-ssl.google.com/android/eclipse/` **Note:** You can use `http://` if you encounter problems using `https://`.

7. In the Install window, ensure that the **Work with** list is set to the **Eclipse ADT Plug-in** repository that you just added.
8. In the Install window, select the **Developer Tools** item to install all the developer tools.
9. Continue through the install wizard.
   - When presented with the license agreement, review and accept if its conditions are acceptable.

- When presented with the panel with the **Finish** button, click **Finish** to download and install the ADT Eclipse plug-in.
- During the installation, you might be warned that the content being installed has not been signed. This is normal. Click **OK** to complete the installation.

10. When prompted to restart the Eclipse SDK, click **Restart Now**.

    When Eclipse restarts, you are prompted to configure the Android SDK.

11. Select **Use existing SDKs** and set the **Existing Location** field to the location where you installed the Android SDK, which should be one of the following:
    - For Windows: `c:\android-sdk-windows`
    - For Macintosh: `/android-sdk-macosx`

Configuring Mobile Designer for the Android SDK

After installing the Android SDK, you need to configure Mobile Designer to provide information about the SDK.

To configure Mobile Designer for the Android SDK

1. Use a text editor to open the following file:

   Mobile Designer_directory/sdk.properties

2. Locate the Android section of the file.
3. Set the values for the properties in the following table.

> **Note:** When specifying paths in the sdk.properties file, use a forward slash character, “/” or an escaped slash character, "\\", to separate folders, even when specifying Window paths.

Property and Setting
android.package Set to the default package prefix that Mobile Designer uses when creating the final build for an Android device.
android.bin.dir.root Set to the path of the location of the default Android SDK that you want Mobile Designer to use to compile mobile applications for Android devices. - For Windows: `c:\android-sdk-windows` - For Macintosh: `/android-sdk-macosx`

4. Optionally, if you want to override default values for a project, set the project property `project.android.sdk.version.override` to the API number you want to use.

   Note: Using an API earlier than 21 may cause failed builds.

5. Optionally configure Proguard, which is included in the Android SDK. For information about the properties you need to set to configure Proguard, see "Proguard Obfuscator Settings" on page 21.

6. Save and close the file.

Setting Up an Android Virtual Device (Emulator)

You can define Android emulators that you can use to test mobile applications.

To create an Android Virtual Device (Emulator)

1. Launch the Android Virtual Device (AVD) Manager tool.
   - **For Windows**, the AVD Manager is located at C:\android-sdk-windows\SDK Manager.exe. You can directly launch it by double clicking the SDK Manager.exe file.
   - **For Macintosh**, open the terminal window and enter the following to start the AVD Manager:

     ```
     cd /android-sdk-macosx
     ./tools/android
     ```

2. From the AVD Manager tool, select **Tools > Manage AVDs**.

   The AVD Manager displays the virtual devices that you have defined.

3. Click **New** to create a new device.

4. Perform the following in the window for creating a new Android virtual device:

   a. In the **Name** field, type a meaningful name for the device you are adding.

      Tip: It is helpful to include the Android API level and screen size in the name.

   b. Select the API level you want to emulate.

      The selected API level determines the version of the Android operating system that runs on your virtual device, as well as default features for that device.

   c. Set the remaining settings that you want to emulate. These settings include:
      - Specifying and/or creating an SD Card image
      - Altering the screen size of the device (the "skin" section)
      - Adding any extra hardware features, such as GPS

5. Click **Create AVD**.

Starting the Android Virtual Device (Emulator)

Note: If the Android emulator must access the Internet through a proxy server, you need to define proxy information. See "Using the Android Emulator with a Proxy Server" on page 40.

To start an Android virtual device

1. Launch the Android Virtual Device (AVD) Manager tool.
   - **For Windows**, the AVD Manager is located at C:\android-sdk-windows\SDK Manager.exe. You can directly launch it by double clicking the SDK Manager.exe file.
   - **For Macintosh**, open the terminal window and enter the following to start the AVD Manager:

     ```
     cd /android-sdk-macosx
     ./tools/android
     ```
2. Select the virtual device you want to start.
3. Click **Start**.

Using the Android Emulator with a Proxy Server

If the Android emulator must access the Internet through a proxy server, you must specify the proxy server to use.

Proxy Information to Provide

To provide proxy server information, use one of the following formats based on whether authentication is required for the proxy server:

- If user authentication is not required, use:

  proxyMachineName`:`*port*
- If user authentication is required, use:

  username`:`*password*`@`*proxyMachineName*`:`*port*

Specifying the Proxy Information

You can specify the proxy server by either providing the proxy information when starting the emulator or by defining an environment variable.

- **To specify the proxy server when starting the emulator,** start the Android Emulator using the `-http-proxy` *proxy* option, where *proxy* is the proxy information using one of the formats described above.

- **To specify using an environment variable,** set the `http_proxy` to the value you want to use for *`proxy`*, where *`proxy`* is the proxy information using one of the formats described above.

  The Android emulator checks the value of the `http_proxy` environment variable when it starts up and uses the value if one is defined.

Note: Launching the Android emulator with the `-verbose` flag displays the current host name and port used for the proxy.

5 Setting Up the iOS Platform

About Setting Up the iOS Platform

Before you can use Mobile Designer to build applications for the iOS platform, you need to set up your environment. The following table lists required and optional tasks to set up the environment.

Setup	Required or Optional	Tasks
Development Environment	Required	Install the Apple Xcode IDE and iOS SDK.
	Required	Perform setup for signing applications. To deploy applications to iOS devices, the applications must be signed. For more information, see "About Signing iOS Applications" on page 45.
Mobile Designer Configuration	Required	Update the Mobile Designer sdk.properties file to provide information about the iOS platform setup. For more information, see "Configuring Mobile Designer for the iOS Platform" on page 48
Simulators	Optional	Use tools such as Proxifer to capture and re-direct messages sent over your Internet connect, inspect network traffic, and assist in debugging. For information about Proxifier, see http://www.proxifier.com/mac/

The procedures in this documentation do *not* cover all possible setups and scenarios. Refer to the Apple Developer website at https://developer.apple.com/devcenter/ios/index.action for further details.

Installing the Apple Xcode IDE

Mobile Designer supports Xcode 5.0 and later. For information about supported iOS SDK versions, see "SDK Versions that Mobile Designer Supports" on page 30.

Note: The iOS SDK is included as part of the Xcode installation. Be aware of the following:

- Use of Xcode 5 requires OS 10.8.

- Developing and testing applications that run on iOS 7 requires Xcode 5.
- Developing and testing applications that run on iOS 8 requires Xcode 6.

This section provides only the minimal information for installing the Apple Xcode IDE so that you can use it with Mobile Designer. For additional details, refer to information about the Apple Xcode IDE on the Apple Developer website at https://developer.apple.com/xcode/index.php.

To install the Xcode IDE from the Mac App Store

1. Ensure that your environment meets the requirements for the Xcode IDE.

   You can find the requirements at https://developer.apple.com/support/ios/ios-dev-center.html

2. On your Macintosh, open the Mac App Store.
3. Search for Xcode.

   Note: You can find Xcode in iTunes at: https://itunes.apple.com/us/app/xcode/id497799835?mt=12. From iTunes, use the link to view Xcode in the Mac App Store.

4. Install Xcode from the Mac App Store.

   Note: You might need to sign in with a user account that has membership in the iOS development program.

5. After installation, open Xcode.
6. On the System Component Installation screen, install **Device Support**,
7. When the installation completes, click **Start Using Xcode**.
8. Select **Xcode > Preferences**.
9. Select the **Downloads** tab.
10. Next to **Command Line Tools**, click **Install**

About Signing iOS Applications

To deploy applications to iOS devices, the applications must be signed. During development, you can sign applications using a development certificate that is limited to a small set of devices. To deploy to the App Store, you must sign the application with a distribution certificate.

If you develop on a Macintosh for iOS, you can use your existing signing environment on the same machine, or you can import an existing environment to another machine. If you do not have an existing signing environment, you need to create a signing environment.

Using an Existing Signing Environment

If you are already developing on a Macintosh for iOS and intend to use the same machine for developing iOS applications with Mobile Designer, you can use your existing signing environment.

To use an existing signing environment on the same machine

1. Start the Keychain Access application, which is in the Applications/Utilities folder.
2. In the Keychain Access application, in the **Keychains** panel, ensure that **login** is selected.
3. In the **Category** panel, select **Certificates**.
4. Note the following information so that you will have it available to configure Mobile Designer.

   - Full name of your iPhone Developer certificate. The full name is typically:

     iPhone Developer: *Firstname Lastname*

   - Full name of your iPhone Distribution certificate, if applicable. The full name is typically:

     iPhone Distribution: *CompanyName*

   You will need this information when you configure the Mobile Designer`ios.devcodesign` and `ios.distcodesign` properties. For more information, see "Configuring Mobile Designer for the iOS Platform" on page 48.

5. Locate your .mobileprovision files and note the path to the following so that you will have it available to configure Mobile Designer.

   - Ad-hoc profile you intend to use for development
   - Distribution profile, if you have one

   You will need path to the .mobileprovision files when you configure the Mobile Designer `ios.adhocprov` and `ios.appstoreprov` properties. For more information, see "Configuring Mobile Designer for the iOS Platform" on page 48.

6. Export your existing certificates using one of the following methods:

   - If you use Xcode to automatically manage your certificates, you can export your existing certificates using the Xcode Organizer.
   - Download your certificates from the iOS provisioning portal, at https://developer.apple.com/ios/manage/overview/index.action

Importing the Signing Environment from Another Macintosh

To use your existing signing setup on another Macintosh, you export signing information from an existing Macintosh and copy it to the Macintosh where you want to develop iOS applications with Mobile Designer.

To import an existing signing environment to another machine

1. On the Macintosh with signing environment you want to use, start the Keychain Access application, which is in the Applications/Utilities folder.
2. In the Keychain Access application, in the **Keychains** panel, ensure that **login** is selected.
3. In the **Category** panel, select **Certificates**.
4. Make a note of the following information:
   - Full name of your iPhone Developer certificate. The full name is typically:

     iPhone Developer: *Firstname Lastname*
   - Full name of your iPhone Distribution certificate. The full name is typically:

     iPhone Distribution: *CompanyName*
5. Export the private key associated with the developer certificate:
   a. In the **Category** panel, select **Keys**.
   b. Select the private key that is associated with the developer certificate. Select **File > Export Items**.
   c. When prompted, create a password for exporting. You will need to supply this password when importing the private key to the target Macintosh.

      Important: Do not use the password you use to login to your Macintosh.

   d. When saving the private key, be sure to save in **Personal Information Exchange (p12)** format.

   Important: You should keep a backup copy of your private key by copying the key to removable media and storing it somewhere safe. If you lose your private key, for example, due to a hardware failure, you will not be able to deploy applications. This is particularly important when you want to update old versions of an application submitted to the App Store. Apple does not keep information about your private key.
6. Move the exported p12 key file to the Macintosh where you want to develop iOS applications with Mobile Designer.
   a. Copy the p12 key file to the target Macintosh and save in any location.

b. Double-click the p12 key file to begin the key import process.

c. When prompted for a password, supply the password you created when exporting the private key.

The private key and required certificates are imported into the target Macintosh

7. Download the appropriate .mobileprovision files for ad-hoc and distribution from the developer portal, https://developer.apple.com/ios/manage/provisioningprofiles/index.action, or copy the. mobileprovision files from your existing environment to an appropriate location from the target Macintosh.

Creating a New Signing Environment

If you do not have a signing environment, you need to set up your environment. You can find setup instructions on the iOS Provisioning Portal at https://developer.apple.com/ios/manage/overview/index.action.

The setup steps include the following:

- Create and install iOS development certificates
- Nominate device IDs for development
- Nominate an App ID
- Create a Development Provisioning Profile

Note: When your environment is ready to distribute iOS applications as Ad-Hoc or App Store builds, you need to create the appropriate .mobileprovision files. You configure Mobile Designer to specify where the .mobileprovision files are located.

Configuring Mobile Designer for the iOS Platform

After installing the Apple Xcode IDE and setting up your environment for signing iOS applications, you need to configure Mobile Designer to provide information about the iOS platform setup.

To configure Mobile Designer for the iOS platform

1. Use a text editor to open the following file:

   Mobile Designer_directory/sdk.properties

2. Locate the iOS section of the file.

3. Set the values for the properties in the following table.

Property and Setting

ios.bundle

Set to the prefix to use for the CFBundleIdentifier.

The CFBundleIdentifier is a unique identifier for your application bundle. For the prefix, it is recommended that you use your company's domain name, with each portion in reverse order. For example, for the domain name mycompany.com, the recommended identifier is “com.mycompany”. An example of setting the property for this identifier is:

```
ios.bundle=com.mycompany.
```

Note: It is important to include a trailing period to act as a separator when specifying the `ios.bundle` property. Mobile Designer appends your application's name directly to the `ios.bundle` value to create your application's unique CFBundleIdentifier. For example, for an application named “MyApp”, the name is "com.mycompany.MyApp".

ios.xcode.app.path

Set to the path of the contents of the Xcode application. The location is typically /Applications/Xcode.app/Contents.

ios.devcodesign

Set to the name of the developer certificate to use for signing iOS builds that you can deploy only to a limited range of known devices, for example, for testing or demonstrating applications.

Specify a String that identifies the certificate stored in your keychain. Typically, the format is "iPhone Developer: *Firstname Lastname* ". An example is:

```
ios.devcodesign=iPhone Developer: John Doe
```

ios.distcodesign

Set to the name of the certificate to use for signing iOS builds intended for distribution to the App Store.

Specify a String that identifies the certificate stored in your keychain. Typically, the format is "iPhone Distribution: *MyCompanyName* ". An example is:

```
ios.distcodesign=iPhone Distribution: SoftwareAG
```

Note: If you do not intend to use this configuration for creating builds to submit to the App Store, you can leave this property blank.

ios.adhocprov

Property and Setting
Set to the path of the Ad-Hoc provision file to use with the developer certificate for signing iOS builds that you can deploy only to a limited range of known devices, for example, for testing or demonstrating applications. An example is: `ios.adhocprov=/Users/softwareag/Desktop/OfficeDevices_Ad_Hoc.mobileprovision`
ios.appstoreprov Set to the path of the provision file to use with the distribution certificate to use for signing iOS builds intended for distribution to the App Store. An example is: `ios.appstoreprov=/Users/softwareag/Desktop/App_Store.mobileprovision` **Note:** If you do not intend to use this configuration for creating builds to submit to the App Store, you can leave this property blank.
project.make.simultaneous.jobs Specifies the number of make jobs that you want Mobile Designer to run simultaneously. Running make jobs simultaneously can help improve the performance when building mobile applications. Specify a positive integer between 1 and *n* *3, where *n* is the number of CPU cores in the build machine. The default is 2. An example is: `project.make.simultaneous.jobs=3` **Caution:** Specifying a number that is too large can cause a slowdown.

4. Save and close the file.

6 Setting Up the Windows Phone 8 Platform

About Setting Up the Windows Phone 8 Platform

Before you can use Mobile Designer to build applications for the Windows Phone 8 platform, you need to set up your environment. The following table lists the tasks you need to perform to set up the environment.

Setup	Required or Optional	Tasks
Development Environment	Required	Enable Hyper-V on your PC. For more information, see "Enabling Hyper-V" on page 53.
	Required	Install the Windows Phone SDK 8. For more information, see "Installing the Windows Phone SDK Version 8" on page 53.
	Required	Install the Visual Studio Express 2012 for Windows Phone. For more information, see "Setting Up Visual Studio Express 2012 for Windows Phone" on page 54.
Mobile Designer Configuration	Required	Update the Mobile Designer sdk.properties file to provide information about the installed Windows Phone 8 platform. For more information, see "Configuring Mobile Designer for the Windows Phone 8 Platform" on page 55.
Emulators	N/A	**Note:** A virtual emulator is installed with the Windows Phone SDK 8.

The procedures in this documentation do *not* cover all possible setups and scenarios. For more information, see the Windows Phone Developer website.

Enabling Hyper-V

The PC you use to develop applications must be running the 64-bit version of Windows 8 Pro or later. To develop mobile applications, the Hyper-V feature must be enabled. However, by default, Windows has Hyper-V disabled.

Important: Enabling Hyper-V requires that you restart your PC. Be sure to save your work before enabling Hyper-V.

To enable Hyper-V

1. Open the Control Panel.
2. In the Control Panel, select **Programs**, then select **Programs and Features**.
3. Click **Turn Windows features on or off**.

   Windows displays the Windows Features dialog.
4. In the Windows Features dialog, select the **Hyper-V** check box.
5. Click **OK**, and then click **Close**.
6. Close the Programs and Features window.
7. Restart your PC.

Two new tiles are added to your Start screen.

Installing the Windows Phone SDK Version 8

Use this procedure to install the Windows Phone SDK 8 so that you can use it with Mobile Designer.

Note: For a list of supported SDK versions, see "SDK Versions that Mobile Designer Supports" on page 30.

To install Windows Phone SDK version 8

1. Open the following webpage in a browser: http://dev.windowsphone.com/en-us/downloadsdk
2. Click **Download** to download the 8.0 version of the SDK.
3. Run the installer to install the Windows Phone 8.0 SDK.

Setting Up Visual Studio Express 2012 for Windows Phone

Visual Studio Express 2012 is automatically installed with Windows Phone SDK 8.0.

Use this procedure to set up Visual Studio Express 2012 for Windows Phone and download the WPToolkit library package.

To set up Visual Studio Express 2012 for Windows Phone

1. Start Visual Studio.
2. Follow the prompts to register Visual Studio with Microsoft for a developer's license.
3. Ensure that the version of NuGet installed with Visual Studio is the latest version by selecting **Tools > Extensions and Updates** in Visual Studio.
4. Create a new project so that you can download the WPToolkit.

   Note: You download the WPToolkit library package from within a project. As a result, to install WPToolkit, you need to set up a new project.

   a. In Visual Studio, close the Updater window, and select **File > New Project**.

      Visual Studio opens the New Project wizard.

   b. In the tree on the left, expand **Installed > Templates > Visual C#**, and select **Windows Phone**.

   c. In the middle panel, select **Windows Phone App**.

   d. In the **Name** field, type a name for the project, for example, `PhoneApp1`.

   e. Note the location where Visual Studio will create the project's root directory, for example:

      C:\Users\developer\Documents\Visual Studio 2012\Projects\PhoneApp1

      Note: You will need to find this location in a subsequent step. Visual Studio displays the location at the bottom of the window.

   f. Click **OK** to create the project.
5. Install WPToolkit by performing the following steps:

   a. In Visual Studio, select **Tools > Library Package Manager > Package Manager Console**.

      Visual Studio opens the package manager terminal inside the Visual Studio window. By default, this is in the bottom-left.

   b. In the console, type the following, and then press ENTER.

      ```
      Install-Package WPToolkit
      ```

      The package manager installs the WPToolkit extras inside your new project.

6. Save a copy of the WPToolkit folder.
   a. Navigate to the root directory of the project that you created, for example:

      C:\Users\developer\Documents\Visual Studio 2012\Projects\PhoneApp1

   b. Open the packages folder and locate the WPToolkit folder, which will have a name, such as, WPtoolkit.4.2012.10.30.
   c. Copy the WPToolkit folder to a convenient location on your hard drive, for example, C:\Development\WPToolkit.
   d. Note the location where you copied the WPToolkit folder.

      Note: When configuring Mobile Designer for the Windows Phone 8 platform, you will configure a property for the location where you copy the WPToolkit folder.

7. Install the SQLite package by performing the following steps.

   Note: Install the SQLite package if you require applications running on Windows Phone 8 to access database information. For more information about using databases, see "Run-Time Database Classes" on page 76.

   a. In Visual Studio, select **Tools > Extensions and Updates**.
   b. Select the **Online** tab and type `SQLite` in the search field.
   c. In the search results, locate the row for **SQLite for Windows Phone** and click **Download**.
   d. Install the SQLite for Windows Phone package.

Configuring Mobile Designer for the Windows Phone 8 Platform

After installing the Windows Phone SDK 8 and setting up Visual Studio Express 2012 for Windows Phone, you need to configure Mobile Designer to provide information about the Windows Phone 8 platform setup.

Note: When specifying paths in the sdk.properties file, use a forward slash character, “/” or an escaped slash character, "\\", to separate folders, even when specifying Windows paths.

To configure Mobile Designer for the Windows Phone 8 Platform

1. Use a text editor to open the following file:

   Mobile Designer_directory/sdk.properties

2. Locate the Windows 8 section of the file and set the values for the properties in the following table.

Property and Setting
microsoft.windows8.phone.path Set to the path of the framework installed with the Windows Phone 8 SDK. The following shows an example using the default installation location is: `microsoft.windows8.phone.path=C:\\Program Files (x86)\\Reference Assemblies\\Microsoft\\Framework\\WindowsPhone\\v8.0`
microsoft.winphone.toolkit.8.0.path Set to the path of the location of the lib\wp8 folder of the WPToolkit package, for example: `microsoft.winphone.toolkit.8.0.path=C:\\Development\\WPtoolkit\\lib\\wp8` **Note:** Use the location in the WPToolkit folder that you saved when performing the instructions in "Setting Up Visual Studio Express 2012 for Windows Phone" on page 54. The \wp8 folder contains a file named "Microsoft.Phone.Controls.Toolkit.dll".
microsoft.sdk.winphone.extensions.sqlite Set to the path of the location of the SQLite extension SDKs installed for Windows Phone. The following shows an example using the default installation location is: `microsoft.sdk.winphone.extensions.sqlite=C:\\Program Files (x86)\\Microsoft SDKs\\Windows Phone\\v8.0\\ExtensionSDKs\\SQLite.WP80`
microsoft.net.framework.path Set the path of the location of the .NET Framework. The default value is: `C:\\WINDOWS\\Microsoft.NET\\Framework\\v4.0.30319` **Note:** This property is located within the XNA/Silverlight section.

3. Save and close the file.

7 Setting Up the Windows RT/Windows 8 Platform

About Setting Up the Windows RT/Windows 8 Platform

Before you can use Mobile Designer to build applications for the Windows RT or Windows 8 platform, you need to set up your environment. The following table lists tasks you need to perform to set up the environment.

Setup	Required or Optional	Tasks
Development Environment	Required	Install the Visual Studio Express 2012 for Windows 8. For more information, see "Installing Visual Studio Express 2012 for Windows 8" on page 58.
Mobile Designer Configuration	Required	Update the Mobile Designer sdk.properties file to provide information about the installed Windows RT and Windows 8 platforms. For more information, see "Configuring Mobile Designer for the Windows RT or Windows 8 Platform" on page 59.
Emulators	N/A	**Note:** A virtual emulator is installed with Visual Studio Express 2012 for Windows 8.

The procedures in this documentation do *not* cover all possible setups and scenarios. For more information, see the Windows Developer website.

Installing Visual Studio Express 2012 for Windows 8

Use this procedure to install Visual Studio Express 2012 for Windows 8.

To install Visual Studio Express 2012 for Windows 8

1. Ensure your system meets the system requirements for Visual Studio Express 2012 for Windows 8.

   Tip: If you will be developing applications for both Windows RT/Windows 8 and Windows Phone 8, use the system requirements for Visual Studio Express 2012 for Windows Phone, which are more restrictive. For more information, see "Setting Up Visual Studio Express 2012 for Windows Phone" on page 54.

a. Open the following webpage in a browser: http://www.microsoft.com/visualstudio/eng/products/visual-studio-express-for-windows-8

b. Scroll down to display and review the system requirements listed on the webpage.

2. Open the following webpage in a browser: http://www.microsoft.com/visualstudio/eng/downloads#d-express-windows-8

3. Download the installer.

> **Note:** The correct installer has a blue icon.

4. Run the installer to install Visual Studio.

5. Start Visual Studio.

6. If required, follow the prompts to register Visual Studio with Microsoft for a developer's license.

7. Install the SQLite package by performing the following steps.

> **Note:** Install the SQLite package if you require applications running on Windows RT/Windows 8 to access database information. For more information about using databases, see "Run-Time Database Classes" on page 76.

a. In Visual Studio, select **Tools > Extensions and Updates**.

b. Select the **Online** tab and type `SQLite` in the search field.

c. In the search results, locate the row for **SQLite for Windows Runtime** and click **Download**.

d. Install the SQLite for Windows Runtime package.

Configuring Mobile Designer for the Windows RT or Windows 8 Platform

After installing the SDK for Windows RT/Windows 8 and Visual Studio Express 2012, you need to configure Mobile Designer to provide information about the Windows RT/Windows 8 platform setup.

> **Note:** When specifying paths in the sdk.properties file, use a forward slash character, “/” or an escaped slash character, "\ \", to separate folders, even when specifying Windows paths.

To configure Mobile Designer for the Windows RT/Windows 8 Platform

1. Use a text editor to open the following file:

   Mobile Designer_directory/sdk.properties

2. Locate the Windows 8 section of the file and set the values for the property in the following table.

Property and Setting
microsoft.windows8.netcore.path Set to the path of the .NETCore 4.5 framework folder. The following shows an example using the default installation location is: `microsoft.windows8.netcore.path=C:\\Program Files (x86)\\Reference Assemblies\\Microsoft\\Framework\\.NETCore\\v4.5`
microsoft.sdk.winrt.extensions.sqlite Set to the path of the location of the SQLite extension SDKs installed for Windows RT/Windows 8. The following shows an example using the default installation location is: `microsoft.sdk.winrt.extensions.sqlite=C:\\Program Files (x86)\\Microsoft SDKs\\Windows\\v8.0\\ExtensionSDKs\\SQLite.WinRT`

3. Save and close the file.

III Creating Mobile Application Projects

8 Setting Up a Mobile Application Project

About Mobile Application Projects

You set up a mobile application project for each mobile application you want to develop. The project contains the application code, defines the devices you want the application to support, and references all the resources that the project requires.

Perform the following actions to set up a mobile application project:

- Create the mobile application project. You start a project by cloning an existing project. webMethods Mobile Designer provides several sample applications that you can use. For more information, see "Creating a New Mobile Application Project" on page 65 and "Sample Projects Provided with Mobile Designer" on page 66.
- Code your application using Java, specifically J2ME. Mobile Designer provides several run-time classes that you can use in your application. For more information, see " Mobile Designer-Provided Run-Time Classes" on page 72 and *webMethods Mobile Designer Java API Reference*.
- Create the resource handler for your application to identify the resources that your project requires. For more information, see "Defining Resources for a Mobile Application Project" on page 101.
- Set properties for your project. Although there are numerous properties you can define for a project, Mobile Designer provides initial settings and/or defaults for most. However, there are a few properties you must set for your project. For more information, see "Setting Project Properties" on page 119.
- Set parameters for your project. Parameters contain settings about devices and resources. Additionally, you can define your own application-specific parameters. In your application code, you use parameters to perform such tasks as loading resources or branching the logic based on parameter values to address the needs of specific target devices. For more information, see "Where You Define Parameters" on page 120.
- Add devices to your project to define the devices you want your application to support. For more information, see "Adding Devices to a Mobile Application Project" on page 87.

Using Software AG Designerwith Mobile Designer

To create a mobile application project using Mobile Designer, you can use Software AG Designer.

Mobile Designer uses Ant tasks. You can use the standard Eclipse Ant view to edit the XML files for Ant tasks. By default, Software AG Designer does not display the Ant View. For information about how to display this view, see "Displaying the Ant View" on page 65.

Creating a New Mobile Project Using the Mobile Development Wizard

When you create a mobile project as described below, you can use all features of Mobile Development.

To create a new mobile project in Software AG Designer

- In Software AG Designer, go to **Help > Help Contents > Software AG Designer Guides > webMethods Mobile Development Help > Creating and Building a Mobile Application** and see the instructions for creating a project using the wizard.

Displaying the Ant View

To display the Eclipse Ant View in Software AG Designer

1. In Software AG Designer, select **Window > Show View > Ant**
2. (Optional) In the Ant view, click **Hide Internal Targets** icon to hide the unused Ant targets.

Creating a New Mobile Application Project

This is the old way for creating new applications. You can use this method to create an old-style application without Mobile Development features.

You can begin a mobile application project by cloning an existing project. You can clone a sample project that Mobile Designer provides or clone a project that you have already created. Clone a project that is similar to the one you want to create.

For example, if you want to create an application that uses NativeUI controls for the user interface, you could start with the _NativeUIHelloWorld_ project. If you want to create an application that uses pixel-level access to the screen, but no NativeUI controls, you could use the _FunctionDemo_ project.

For a description of the sample projects that Mobile Designer, see "Sample Projects Provided with Mobile Designer" on page 66.

To create a new mobile application project

1. If the project you want to clone is not in Software AG Designer, import the project.
   a. In Software AG Designer select **File > Import > General > Existing Projects into Workspace**, and then click **Next**.
   b. In **Import Projects**, click **Browse** and select the project source directory of the project you want to clone.
   c. Click **OK**, and then click **Finish**.
2. Use the Clone-Project Ant target to make a copy of the original project.

a. In the Project Explorer view, expand the project you want to clone and drag its build.xml file to the Ant view.

   If the Ant view is not open, see "Displaying the Ant View" on page 65.

b. In the Ant view, double-click **Clone-Project**.

c. In the Clone Project dialog, specify the following:

   - Directory path where you want the save the cloned version of the project
   - Name for the cloned project
   - MIDlet name (application name of your project) for the cloned project

   Caution: Do not create projects in directories that have spaces in their names. Some third-party tools are less tolerant of spaces in directory names, and you might get compilation errors as a result.

Mobile Designer creates the cloned project in the location you specified.

After you clone a project, you can begin editing it to the requirements for you application. For example, you can change the names used in the final build, the version of Mobile Designer to use at both build time and run time, and you can start adding new code.

3. In the Project Explorer view, expand the project you just cloned, and drag the build.xml file to the Ant editor. In the build.xml file, you can:

   - Rename the *MIDlet*, which is the application name of your project.
   - Identify the targets for which you want to build this project.

     Add a call to the importxmldirectory Ant task to import all individual handset targets set up for the project. You can replace this with a list of individual import calls. However, doing so requires updating the build.xml file each time you add a handset.

4. Define project-specific properties. For instructions, see "Where You Set Properties" on page 116 and "Setting Project Properties" on page 119.

Sample Projects Provided with Mobile Designer

Mobile Designer comes with the sample projects described in this section. The sample projects are located in the following directory:

Mobile Designer_directory/Samples

Expense Tracker

The Expense Tracker sample project uses many NativeUI objects. It demonstrates how to solve design and implementation difficulties when developing mobile applications.

It features user interface conventions that are common requirements for mobile applications, such as dynamic list population and display; data entry, storage and reporting mechanisms; handling for multiple device platforms and form factors.

Library JSON

The Library JSON sample project is an example of a library project. It demonstrates how you can precompile parts of your codebase into separate libraries that you can then reference in another project. Specifically, this project uses third-party JSON Java code. For more information about creating and using libraries, see "Creating and Using Code Libraries" on page 81.

You compile the project for the target platforms for which you want to use the library. A library project's build.xml references libtargets.xml rather than targets.xml. The target libraries for this sample project are J2ME (for Phoney), Android, and iOS. To use this sample project, you can import it, then cross-compile and build the library using the +Library-Build Ant task.

Note: For a sample of a project that references this library, see the NativeUI JSON sample project.

NativeUI Demo

The NativeUI Demo application demonstrates the use of all the major native user interface (NativeUI) classes in Mobile Designer.

The sample also demonstrates how to support tablet devices. It contains code to determine whether it is running on a tablet based on the screen size of the device. When running on a tablet, the application uses multiple panes in the user interface. For more information about using panes, see *webMethods Mobile Designer Native User Interface Reference*.

Mobile Designer provides two versions of the NativeUI Demo.

- _NativeUIDemo_ was hand coded using Mobile Designer functionality.
- _NativeUIDemoX_ was created using Mobile Development.

  This version of the sample illustrates how to use Mobile Development to create a mobile application. In this sample, many of the user interface objects were added explicitly in the Outline Editor. For example, a button was added using the Mobile Development **Button** object. However, some objects were defined using the dynamic objects that Mobile Development offers, for example, the **DynamicDisplayObject** object. The dynamic objects were used to illustrate how to use dynamic objects and how to provide user code for dynamic objects.

  Mobile Development provides a default resource handler that handles most resources. However, this project also uses a custom resource handler to illustrate handling cases when custom code/dynamic objects requires resources that the default resource handler cannot accommodate.

Note: To use the _NativeUIDemoX_ version of the sample, you need to import the sample project into Software AG Designer. The _NativeUIDemoX_ sample project includes the information for the model. It also includes user logic in the src folder (that is, the user space). However, before you can use the project, you must use the Mobile Development **Generate Source Code > Application Model and API** command to generate sources for the project. This command generates the logic to execute the model in the gen/src folder and the Mobile Development API in the gen/api-src folder. For more information about generating sources, see *webMethods Mobile Development Help*.

NativeUI Contacts

The NativeUI Contacts application demonstrates the use of the Personal Information Management (PIM) APIs defined for JSR 75.

The sample demonstrates using the PIM APIs to:

- Read information for contacts that already exist in a device's address book. This can be done for devices running on any platform.
- Edit existing contacts in a device's address book and adding new contacts to a device's address book. This can be done for devices running on platforms that support editing the address book.

NativeUI Exercise

The NativeUI Exercise sample project shows the process of creating a simple native user interface voting application, complete with model answers for each of the steps.

NativeUI Hello World

The NativeUI Hello World sample project contains the bare minimum needed to display some text and transition between two Views. You can copy this project to assist in learning the native user interface classes in Mobile Designer.

NativeUI JSON

The NativeUI JSON sample project shows the interaction with a JavaScript Object Notation (JSON)-based server, fetch and display data.

Note: This sample references the library created using the Library JSON sample project.

NativeUI Location

The NativeUI Location sample project shows the use of the Location API with native user interface classes to display the user's current location.

NativeUI My Graphical Element

The NativeUI My Graphical Element project demonstrates the use of a custom user-created native user interface element to draw a chart.

NativeUI My Native Element

The NativeUI My Native Element project demonstrates how to add platform-specific native code to create a new custom visual component that works along with the NativeUI objects that Mobile Designer provides.

NativeUI PDF Demo

The NativeUI PDF Demo sample project demonstrates the use of native platform code injection to display a PDF.

NativeUI SOAP

The NativeUI Soap sample project uses SOAP to communicate with a remote server, fetch data, and display it on the device.

NativeUI Push Notifications

The NativeUI Push Notifications sample project demonstrates the use of push notifications on supported platforms.

NativeUI Database

The NativeUI Database sample project demonstrates the use of databases on supported platforms.

9 Coding a Mobile Application

Mobile Designer-Provided Run-Time Classes

webMethods Mobile Designer provides many run-time classes that provide an array of features that you can use in your application. You can find details about all the classes in the *webMethods Mobile Designer Java API Reference*.

Application and Parameter Classes

com.softwareag.mobile.runtime.core.Application

The Application class contains only the minimal functionality to start an application and detect core interrupt and termination events.

The Application class also provides debug functionality to output messages. The messages are prefixed with `MD:` so that you can visually differentiate Mobile Designer debug information from other generated debug information. You can define flags to indicate the debug messages you want displayed.

When running on a PC or device that has a connected console output solution, the Application class displays the debug messages on the console. Otherwise, the debug messages are available in a String Array. You can add logic to your application to obtain the messages from the String Array and output them to the screen if you want visual debugging on a device.

com.softwareag.mobile.runtime.Parameters

When Mobile Designer runs the resource handler that you create for your project, it automatically creates the Parameters static class, which contains the parameters that drive the Mobile Designer run-time code for a particular build. The Parameters class, com.softwareag.mobile.runtime.Parameters, contains information about the devices and project-specific parameters, such as resource, resource block, and text IDs.

Mobile Designer uses the following naming conventions for the parameter names:

- `PARAM_MD_***` defines parameters controlling Mobile Designer run-time source code functionality
- `PARAM_***` defines application-specific parameters
- `RESBLOCKID_***` defines identifiers for all the resource blocks
- `RESID_***` defines identifiers for all the individual resources
- `TEXTID_***` defines identifiers for all the lines of text
- `MENUID_***` defines identifiers for all the menus

When the remaining run-time classes are not used in a project, you can generate the Parameters class by reference in third-party code.

Run-Time Canvas Classes

com.softwareag.mobile.runtime.core.CanvasCore

Use the CanvasCore class to control application state transitions and to create new threads when the application enters loading states so that the primary thread can animate the screen as required. In addition, the CanvasCore class:

- Detects interrupts and feeds information through to the application code so that it can respond appropriately, while automatically stopping any playing music or vibrations.
- Provides a frame-rate handling solution using either a `variableRateUpdate`, where millisecond update times are passed into the application code, or a `fixedRateUpdate`, a potentially preset rate.
- Performs safety checks to prevent large visual stutters if random long pauses occur in the device's JVM.
- Provides touchscreen pointer support and the ability to define regions on screen that perform the same as defined keypresses would on non-touchscreen devices.
- Detects QA test-codes for ease of debugging.
- Provides standardized keypress detection.

Mobile Designer stores keypress information for each device. You can decide how the keypress response works in your application by setting the Mobile Designer project.numeric.keys.emulate.directionals project property.

You can set the parameter `PARAM_MD_CORE_DEBUGFLAGS_TO_DISPLAY` to `0` (zero) to avoid storing debug or output, removing of the debug method. However, this might not necessarily be true for all of the data associated with the debug method call. For example, in the following code:

```
debug ("Having a problem loading object #" + i, PARAM_MY_DEBUGFLAG);
```

The method call itself is obfuscated, but the creation of the String included in the debug method call could remain due to the way the compiled Java byte code is created. As a result, Mobile Designer recommends changing the call to:

```
if ((PARAM_MD_CORE_DEBUGFLAGS_TO_DISPLAY & PARAM_MY_DEBUGFLAG) != 0)
debug ("Having a problem loading object #" + i, PARAM_MY_DEBUGFLAG);
```

com.softwareag.mobile.runtime.core.CanvasBase

The CanvasBase class provides the canvas and interaction to events. Mobile Designer sets the value for each device. You can override the value Mobile Designer sets using the mobiledesigner.runtime.core.class.graphics.canvas project property.

com.softwareag.mobile.runtime.core.CanvasDimensions

Use the CanvasDimensions class to specify the screen height and width Mobile Designer sets the `CURRENT_SCREEN_HEIGHT` and `CURRENT_SCREEN_WIDTH` for

each device. You can override the value Mobile Designer sets by setting the mobiledesigner.runtime.core.class.graphics.dimensions project property to either `fixed` or `dynamic`.

If the parameters `MD_BASE_SCREEN_WIDTH` and `MD_BASE_SCREEN_HEIGHT` are set to `0` (zero) in the build script, or the device is set up to require dynamic dimensional detection, this layer queries the canvas's getWidth and getHeight methods with each call to retrieve the current screen dimensions, rather than embedding the dimensions as static compile-time constants. Using `dynamic` dimensions means that means that some compile-time optimizations do not occur. If you want the user to be able to rotate the device screen, you need to use the `dynamic` dimensions value.

com.softwareag.mobile.runtime.core.CanvasInterrupts

Use the CanvasInterrupts class to control how your mobile application detects interrupts. Mobile Designer sets the value for each device in the project. You can override the value Mobile Designer sets using the mobiledesigner.runtime.core.class.interrupts project property.

com.softwareag.mobile.runtime.core.CanvasKeysandTouch

Use the CanvasKeysandTouch class to control how the mobile application detects keypress, touch, or pointer events. Mobile Designer sets the value for each device. You can override the value Mobile Designer sets using the mobiledesigner.runtime.core.class.keysandtouch project property.

When you want to override the keypress detection methods, the Mobile Designer run-time classes pass these keypress actions to the standard keyPressed and keyReleased methods, enabling methods overrides to work consistently regardless of the specified target.

com.softwareag.mobile.runtime.core.CanvasMenu

Use the CanvasMenu class to control the loading, creation and general data structures for all menus. You can create menus in the resource handler and manipulate them at run time.

Mobile Designer must determine whether the menu item type is the `MENU_ITEMTYPE_SOFTKEY`. Mobile Designer can automatically create the appropriate soft keys when a menu is loaded. You can create any other menu item based on the basic standard and soft key standard, as long as its initial data chunk remains unchanged, and default items such as header, text-item, button are included in Mobile Designer.

Menus are constructed from a list of menu types and a general menu definition that includes information about the number of items present, the menu type, and information on the previous menu and previous selected item.

Individual menu items are a simple list of integers of arbitrary length with the leading integer specifying:

- The top 1 byte (0xff000000) specifies the item type.
- The middle 2 bytes (0x00ffff00) specifies the item flags. You can use item flags as masks to indicate the items the user can select, or for any other user functionality.

- The bottom 1 byte (0x000000ff) specifies the item length including this standard int.

com.softwareag.mobile.runtime.core.CanvasNativeUI

Use the CanvasNativeUI class to control the loading, creation and general data structures of all the Native User Interface (NativeUI) menus. To enable the CanvasNativeUI class, set the project.runtime.uses.nativeui and the mobiledesigner.runtime.core.class.ui properties to `true`.

com.softwareag.mobile.runtime.core.CanvasSoftKeys

The CanvasSoftKeys class controls the creation and update of soft-key labels. Mobile Designer attempts to follow the forward and back standard expected for each device by informing the code where to display the forward and backward soft keys. The Mobile Designer run time and device profiling are based on a forward or back soft-key naming convention, rather than left or right.

Mobile Designer sets the value for each device. You can override the value Mobile Designer sets using the mobiledesigner.runtime.core.class.softkeys project property.

com.softwareag.mobile.runtime.core.CanvasThreading

Use the CanvasThreading class to control how the mobile application manages the primary thread using a java.lang.Thread or a java.lang.TimerTask. The mobile application always launches all secondary threads using a new java.lang.Thread.

Mobile Designer sets the value for each device. You can override the value Mobile Designer sets by setting the mobiledesigner.runtime.core.class.threading project property to either `thread` or `timertask`.

Run-Time Comms Classes

com.softwareag.mobile.runtime.comms.HttpConnectionHandler

Use the HttpConnectionHandler class to initiate HTTP connections and manage sending and downloading data in separate threads.

Mobile Designer sets the default value for a device based on the device's capabilities. The default value is set in the device's profile in the Mobile Designer device database. For more information about the device profiles and the device database, see "Devices that a Mobile Application Supports" on page 88.

You can override the value Mobile Designer sets using the mobiledesigner.runtime.core.class.comms.httpconnection project property.

com.softwareag.mobile.runtime.comms.MessageConnectionHandler

Use the MessageConnectionHandler class to control the detection of incoming SMS messages using the Wireless Messaging API and to send SMS messages to other phones.

Mobile Designer sets the default value for each device. You can override the value Mobile Designer sets setting the mobiledesigner.runtime.core.class.comms.messageconnection project property to `none` or `wma`.

Note: The MessageConnectionHandler implements the MessageListener class.

Run-Time Database Classes

Mobile platforms can support a database. For example, many platforms support the SQLite database. Mobile Designer provides the com.softwareag.mobile.runtime.database library that contains classes and methods you can use in your mobile applications to execute SQL statements. Using the database library allows your mobile application to access database information. The database library provides a uniform manner for using a database independent of the target platform and database system. For more information about the database library, see the *webMethods Mobile Designer Java API Reference*.

Note: Support for the database library is provided only for the following platforms:

- Android
- iOS
- Phoney
- Windows Phone 8
- Windows 8
- Windows RT

To use the database library in a mobile application, you must set the project.handset.uses.Database project property to `true`.

For an example that illustrates how to use the database library, see the NativeUI Database sample.

Important: On most platforms, the Cursor class is a Buffered Cursor, which stores all query results in memory. Use caution when querying database tables that contain data that uses the Blob data type, for example, images.

Run-Time Media Classes

com.softwareag.mobile.runtime.media.AudioHandler

Use the AudioHandler class to control the sound and vibration functionality when writing for the main common audio libraries available on mobile devices. The media classes try to keep the phone's backlight on when possible for the selected device.

The Mobile DesignerAudioHandler class sets the value for each device. You can override the value Mobile Designer sets using the mobiledesigner.runtime.core.class.sound project property.

com.softwareag.mobile.runtime.media.CameraHandler

Use the CameraHandler class to initialize a device's camera, take snapshots, and stop the camera.

Mobile Designer sets the values for the CameraHandler class for each device. You can override the value Mobile Designer sets using the mobiledesigner.runtime.core.class.camera project property.

com.softwareag.mobile.runtime.media.ImageBase

The ImageBase class provides the base level of the image creation and drawing functionality, including:

- Decoders for the various image encoding methods that are part of the Mobile Designer Resource Handler.
- Image caching used with some devices that have problems freeing images from memory.
- Multi-celled image support, stored as individual images at run time, or as a single image which is clipped and drawn.
- Transformations for all devices
- Image dimension information to support any multi-cell images or transformations.

You can access the ImageBase functionality through the ImageHandler class.

The `DRAWIMAGETRANSFORM_***` values and the `LOADIMAGETRANSFORM_***` values are not interchangeable. The `LOADIMAGETRANSFORM_***` values are used to cache multiple images at load time. For more information, see the *webMethods Mobile Designer Java API Reference*.

com.softwareag.mobile.runtime.media.ImageHandler

Use the ImageHandler class to load, draw, and manage images. Mobile Designer sets the values for the ImageHandler class for each device. You can override the value Mobile Designer sets using the mobiledesigner.runtime.core.class.graphics.image project property.

com.softwareag.mobile.runtime.media.PngParser

The PngParser class provides a PNG-encoding method that can create an image from pixel and palette data. In Mobile Designer you can create the image dynamically at run time, or use image compression methods that exceed the default PNG format.

When opaque mutable images render faster on a device than immutable images, the PngParser makes adjustments to improve performance when rendering the image.

You can generate palettized (PNG-8) and true-color (PNG-24) PNG files using the `PngParser` methods when full pixel or palettized information is provided.

com.softwareag.mobile.runtime.media.TextHandler

The Mobile DesignerTextHandler class provides the following functionality:

- Loads text files
- Supports multi-languages
- Stores hyphenation guidelines for text splitting
- Dynamically splits text based on hyphenation guidelines
- Injects text
- Supports system and bitmap font
- Draws proportional and fixed width font in all systems
- Handles all the expected metric queries such as font height, character width, and string width

Run-Time Serializer Class

com.softwareag.mobile.runtime.serialize.Serializer

The Serializer class provides the ability to serialize a Java Class into a binary stream and to deserialize a binary stream back to a given Java class.

To enable a class to be serialized, you must ensure it implements the com.softwareag.mobile.runtime.serialize.Serialzable interface (not the J2SE java.io.Serializable interface). A class that implements the Serializable Interface can then be passed to the Serializer class for serialization and deserialization.

Mobile Designer sets the values for the Serializer class for each device. You can override the value Mobile Designer sets by setting the mobiledesigner.runtime.core.class.serialize project property to `cldc11`.

Run-Time Storage Classes

com.softwareag.mobile.runtime.storage.RecordStoreHandler

Use the Mobile DesignerRecordStoreHandler class to control saving data in the application's RecordStore. The RecordStoreHandler class also determines whether saving the data is performed immediately or is cached and saved on termination. Use the cache and save option with slow devices.

For access to the RecordStore, call the derivatives of setRecordStoreData and getRecordStoreData.

com.softwareag.mobile.runtime.storage.ResourceDataTypes

The ResourceDataTypes class contains a set of helper methods to retrieve primitive data types from your resources with the run-time ResourceHandler class. Two variants are present that enable including the float and double methods with Connected Limited Device Configuration (CLDC) 1.1 devices.

Mobile Designer sets the ResourceDataTypes value for each device. You can override the value Mobile Designer sets by setting the mobiledesigner.runtime.core.class.datatypes project property to `cldc11`.

com.softwareag.mobile.runtime.storage.ResourceHandler

Use the ResourceHandler class to manage the loading and caching of resources, resource packs, and resource blocks.

Depending on parameter settings, devices can:

- Cache the entire resource packs
- Cache individual blocks
- Load only the resources that are individually stored in the application bundle

You need to use clean-up methods to ensure that the mobile application handles memory management in the most appropriate way for all devices.

Run-Time Utility Classes

com.softwareag.mobile.runtime.utility.Maths

The Maths class is a fixed point math library that contains methods you can use in your applications for conversion to and from a fixed-point number, trigonometric functions, square and cube roots methods, and a random method. The random method is included for instances when your application relies on the random method returning the same value across all devices. Built-in JVM implementations can differ from one device to another in their number handling. As a result, if you are porting your code to anything other than Java, you could encounter differences in the output produced by the random function.

The parameter `PARAM_MD_MATHS_FP_SHIFT` controls the accuracy of the math performed, with a fixed point value of 1 equal to `1 << PARAM_MD_MATHS_FP_SHIFT`.

The `Maths` class is based on Connected Limited Device Configuration (CLDC) 1.1. The Maths class does not contain references to the primitive types `float` or `double`.

com.softwareag.mobile.runtime.utility.PlatformRequest

Use the PlatformRequest class to launch the browser on devices that support Wireless Application Protocol (WAP) browsers.

Mobile Designer Logging API

Mobile Designer provides a java.util.logging API that is based on the Java Logging API standard. The java.util.logging package that Mobile Designer provides contains classes and interfaces that are based on the Connected Limited Device Configuration (CLDC) 8 standard. You can find information about this package in the javadocs for the java.util.logging package provided with the CLDC 8 standard.

Note: The Mobile Designer version of java.util.logging does *not* include the LoggingPermission class.

The following table describes some limitations and differences in the Mobile Designer version of the java.util.logging API based on platforms:

Platform	Limitation or Difference
All Platforms	The Logger class that Mobile Designer provides does *not* include a Logger.getGlobal() method. To access the global logger, use the following: `Logger.getLogger(Logger.GLOBAL_LOGGER_NAME)`

Android	When running applications on the Android platform, the application uses the Java Standard Edition (Java SE) Logging API. It does not use the java.util.logging package that Mobile Designer provides.
	When configuring loggers, take into consideration that the global logger is not a root logger. The following code shows how to obtain the root logger in the logger's hierarchy.
	`Logger rootLogger = Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);`
	`if (rootLogger.getParent() != null)`
	`{`
	`  rootLogger = rootLogger.getParent();`
	`}`

iOS	When running applications on the iOS platform, the application uses the java.util.logging API that Mobile Designer provides rather than the Java SE Logging API. The standard output for the ConsoleHandler class is console in Xcode.
Windows Phone Windows RT Windows 8	When running applications on the Windows platforms, the application uses the java.util.logging API that Mobile Designer provides rather than the Java SE Logging API. The standard output for the ConsoleHandler class is output in Visual Studio.

Platform	Limitation or Difference
Phoney	When running applications in the Mobile Designer Phoney utility, the application uses the Java SE Logging API. It does not use the java.util.logging package that Mobile Designer provides.

Creating and Using Code Libraries

If there are parts of your codebase that you use repeatedly, rather than copy the code into multiple mobile application projects, you can create a library that you can reference in your mobile applications. To create a library, you precompile the parts of your codebase that you want to reference into separate libraries. Building a library of frequently-used code allows you to avoid repeated cross compilation of the code you add to the library and speeds up development.

Building a Library that You Want to Reference in Other Projects

For an example of a sample library project, see the Library JSON (_LibraryJSON_) sample project, which you can find in the following directory:

Mobile Designer_directory/Samples

In a library project, the build.xml file references a libtargets.xml file rather than a targets.xml file. Additionally, to compile, use the +Library-Build Ant task rather than the +Multi-Build Ant target.

Important: Be sure to precompile your library for each target platform with which you want to use the library.

To successfully activate an application project that references a library project, you must build the J2ME libraries in the referenced library project first.

To build a library

1. Start a new project, or you can import and clone the provided Library JSON sample project. For more information, see "Creating a New Mobile Application Project" on page 65.

   Note: To clone the project, use the Clone-Project Ant target.

2. Use the +Library-Build Ant task to compile the codebase.
   a. In the Project Explorer view, expand the project, and drag the build.xml file to the Ant view.
   b. In the Ant view, double-click **+Library-Build** to launch the Library Build dialog.

c. In the Library Build dialog, select the platforms for which you want to build the library.

The +Library-Build Ant task compiles the library code into the libraries for the chosen platforms and places the result in a library in the Builds folder.

Referencing a Library

After you have created a library, you can reference the library in a mobile application project. You must also add the `project.library.name` and `project.library.path` properties to your mobile application project. To apply the setting to all targets, set the property in the project's _defaults_.xml file. To apply the setting to a specific target device, set the property in the target file for that device, *target_name* .xml.

Property and Description
project.library.name Specifies the name of one or more libraries that the mobile application references. Use a semi-colon delimited list to specify multiple libraries. Example value: `_LibraryJSON_`
project.library.path Specifies the path to one or more Builds folders where the precompiled libraries reside for each platform. Use a semi-colon delimited list to specify multiple libraries. Example value: `$(env.MOBILE_DESIGNER}/Samples/_LibraryJSON_/Builds/0.0.0`

When running the +Multi-Build Ant target for the project, Mobile Designer will automatically inject any references as required in the build process for each target platform to enable it to build and run.

For an example of referencing a library that you create using the +Library-Build Ant task, see the NativeUI JSON (_NativeUIJSON_) sample project, which you can find in the following directory:

Mobile Designer_directory/Samples

Using System.getProperty to Obtain Device Information

You can use the Java System.getProperty(String) method to return system information for the device on which your application is running. The table below lists properties that you might find useful.

When invoking the Java method, supply the property name as the String input variable. For example, if you want to use the `mobiledesigner.device.name` property, invoke the following:

```
System.getProperty("mobiledesigner.device.name")
```

Property and Description
mobiledesigner.device.firmware The System.getProperty method returns information that identifies the firmware of the device. If you are running the application in Phoney, the System.getProperty method returns the full Mobile Designer version number including the build number, for example, “9.5.1.2.344”.
mobiledesigner.device.name The System.getProperty method returns information that identifies the device hardware, for example, “iPhone4S”. If you are running the application in Phoney, the System.getProperty method returns “Phoney”.
mobiledesigner.device.uid The System.getProperty method returns information that uniquely identifies the specific device on which the application is running. This is typically a unique String.
mobiledesigner.display.ppi The System.getProperty method returns the resolution of the device’s screen in pixels per inch, for example, “240”.
mobiledesigner.display.scaling.percent The System.getProperty method returns a percentage value that you can use to determine the physical size of a screen. Some Windows devices provide a virtual screen size that is not equal to the number of physical pixels in the device's display. In this case, the operating system automatically scales the content to fit. Use this property to obtain a percentage value that indicates the scaling factor to apply to the device's reported width and height values to get the actual physical size of the screen in pixels. This property is supported for Windows 8, Windows RT, and Windows Phone 8.
mobiledesigner.domain.availability:*domain_name*

Property and Description
The System.getProperty method returns whether the device can connect to the domain specified by *domain_name*. For example, if you want to determine whether the device can connect to www.softwareag.com, use `mobiledesigner.domain.availability:www.softwareag.com`. The return values are one of the following: ■ `true` if the device can connect. ■ `false` if the device cannot connect.

mobiledesigner.locale

The System.getProperty method returns the language the device is currently configured to use. The language is returned in the following format:

language _*COUNTRY*

where:

- *language* is the two-character, lowercase language code defined by ISO 639.
- *COUNTRY* is the two-character, uppercase country code defined by ISO3166.

Note: In some circumstances, the country code might not be returned, for example, if the device does not grant access to the country information. In these circumstances, only the two-letter language code is returned, for example, “en”.

mobiledesigner.online.availability

The System.getProperty method returns whether the device is connected to a network. The returned information is one of the following:

- If the device is *not* connected to a network, the method returns “false”.
- If the device is connected to a network, the return information is in the following format:

  true:*method*

  For example, if the device is connected to WiFi, the return information is “true:wifi”.

  If the network interface details are not specified, *method* is “unknown”, for example, “true:unknown”.

mobiledesigner.platform

The System.getProperty method returns the platform for the device. The platform name that the System.getProperty method returns matches how Mobile Designer lists platforms, for example, “Android”, “iOS”, or “WinPhone”.

Property and Description
If you are running the application in Phoney, the System.getProperty method returns the name of the platform you are simulating in Phoney.

Creating the User Interface

To create the user interface for your mobile application, use the Mobile Designer Native User Interface. For more information, see the *webMethods Mobile Designer Native User Interface Reference*.

10 Adding Devices to a Mobile Application Project

Devices that a Mobile Application Supports

Mobile Designer has a device database that provides device profiles for many devices. You are encouraged to use the more generic devices and code for multi-resolution apps.

The device profiles are located in the following directory:

Mobile Designer_directory/Devices

In your mobile application project, you reference the subset of the devices your mobile application project will support. The project's target folder contains an XML file for each device that your mobile application supports. Mobile Designer creates the XML file and adds it to the project's target folder when you execute the `Add-Handset` Ant target to add a device to your project. You can add a device to your project at anytime during the development phase. For more information, see "Adding a Device to a Project" on page 88.

If needed, you can change the settings that Mobile Designer provides in a device profile by executing the +Add-Update-Handset Ant target. For more information, see "Updating an Existing Device Profile in the Device Database" on page 90.

If you want an application to support a device for which Mobile Designer does not provide a device profile, you can add a device profile to the Mobile Designer device database. To do so, you first use the Device Profiler sample application to determine the settings for the device. After you determine the settings, you execute the +Add-Update-Handset Ant target to manually add the device profile. For more information, see "Determining Device Settings by Running the Device Profiler" on page 91 and "Adding a Device Profile to the Device Databases" on page 97.

Adding a Device to a Project

To add a device to a mobile application project, execute the Add-Handset Ant target from your project. When adding the device, you associate the device with a language or language group.

The Add-Handset Ant target adds an XML file for the device to your project's targets folder. The following is a sample of the target XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<project>
<property name="project.handset.AND_generic_Android3xAPI.langgroups" value="EFIGS"/>
<property name="project.handset.AND_generic_Android3xAPI.mobiledesigner.handsetgroup"
value="AND_generic_Android3xAPI"/>
<target name="-Project.Handset.AND_generic_Android3xAPI">
<!-- properties, parameters and paths -->
<!-- Load the global info for this project and the Mobile Designer          -->
<!-- handset group                                                    -->
<call-mobiledesigner-handset-target handset="${selected.handset}"/>
</target>
</project>
```

To add a device to a project

1. In Software AG Designer, open the project's build.xml file in the Ant view.

   a. In the Project Explorer view, locate the project to which you want to add a device.

   b. Expand the project and drag its build.xml file to the Ant view.

   If the Ant view is not open, for instruction, see "Displaying the Ant View" on page 65.

2. In the Ant view, double-click **Add-Handset**.

   Mobile Designer displays an Add Handset dialog.

3. In Add Handset dialog, select the platform on which the device runs from the **Platform Filter** list.

4. Select the manufacturer that makes the device from the **Manufacturer Filter** list.

   When you select a valid platform and manufacturer combination for which Mobile Designer has a device profile, Mobile Designer populates the **Choose your handset** list and displays a default device name in the **Reference Name** field.

   If Mobile Designer does not have a device profile for the platform and manufacturer combination you specified:

   - Ensure you are specifying the information correctly by reviewing the information in the following directory to determine the names of supported devices and platforms:

     Mobile Designer_directory/Devices

   - If the device you want is not available, you can add a new device profile for the device. For more information, see "Determining Device Settings by Running the Device Profiler" on page 91 and "Adding a Device Profile to the Device Databases" on page 97.

5. Select the device you want to add to the project from the **Choose your handset** list.

6. Accept the default name in the **Reference Name** field or update it if you want to use another name.

7. In **Language Groups**, type a semicolon-separated list of language groups to specify the languages that your mobile application will support for the device. You define language groups using the project.langgroup.group_name property.

   For example, if you have defined the language groups `europe` and `asia` and want to specify those language groups, use the following:

   ```
   europe;asia
   ```

8. Click **Add Handset**.

Updating an Existing Device Profile in the Device Database

You can update the settings in an existing device profile by executing the +Add-Update-Handset Ant target, also known as the *Profile Updater*. When you update an existing device profile in the Mobile Designer device database, the +Add-Update-Handset Ant target modifies the XML profile file in the following directory:

Mobile Designer_directory/Devices

Note: This procedure describes how to use the +Add-Update-Handset Ant target to update a device profile. If you want to add a new device profile for a device, see "Determining Device Settings by Running the Device Profiler" on page 91 and "Adding a Device Profile to the Device Databases" on page 97.

To update an existing device profile

1. If the Device Profiler project is not in Software AG Designer, import the project.
   a. In Software AG Designer select **File > Import> General > Existing Projects into Workspace**, and then click **Next**.
   b. In **Import Projects**, click **Browse** and select the project source directory of the project you want to clone.

      The source code for the Device Profiler sample application is in the following directory:

      Mobile Designer_directory/Samples/_DeviceProfiler_
   c. Click **OK**, and then click **Finish**.
2. Open the Device Profiler project's build.xml file in the Ant view.
   a. In the Project Explorer view, locate the Device Profiler project.
   b. Expand the project and drag its build.xml file to the Ant view.

   If the Ant view is not open, for instruction, see "Displaying the Ant View" on page 65.
3. In the Ant view, double-click **+Add-Update-Handset**.
4. In the Choose Handset dialog, select the device profile you want to update from the **Edit an existing handset** list.
5. Click **Update**.

   The +Add-Update-Handset Ant target displays the Update Handset Information dialog, which displays the default settings for the device and any device-specific overrides.
6. In the Update Handset Information dialog, for each parameter or property you want to update, type a value in the **Handset Override** column.

> **Note:** You can also specify the values for the properties used in the Java run-time code, such as the obfuscator to use, the MIDlet-icon size, and the maximum JAR size.

7. Optionally, in **Comments** field, type a explanation for the change.
8. Click **Update Handset Information**.

   The Console view indicates whether the update was successful or failed.

Determining Device Settings by Running the Device Profiler

If you want your mobile application project to support a device for which Mobile Designer does not have a device profile, you can add a device profile. To do so, you must first determine the settings for the device. To determine the settings, use the Mobile Designer Device Profiler sample application and run it on the device.

After using the Device Profiler to determine the settings, execute the +Add-Update-Handset Ant target to add a device profile for a device. For more information about executing the +Add-Update-Handset Ant target, see "Adding a Device Profile to the Device Databases" on page 97.

The _DeviceProfiler_ project uses the Mobile Designer build process, but does not use the Mobile Designer run-time code.

Mobile Designer provides the source code and build process for the Device Profiler. However, you might need to modify and recompile the application to work on a specific device. For example, the Device Profiler contains tests to determine whether certain APIs and functions are present on a device. However, some devices perform a pre-installation check on the contents of an application bundle, and if the device check finds references to classes it does not implement, the device will not allow the installation of the application. The Device Profiler has parameters that you might need to alter to disable features that prevent the Device Profiler application from working on a given device.

To run the Device Profiler to determine settings for a device

1. If the Device Profiler project is not in Software AG Designer, import the project.
   a. In Software AG Designer select **File > Import> General > Existing Projects into Workspace**, and then click **Next**.
   b. In **Import Projects**, click **Browse** and select the project source directory of the project you want to clone.

      The source code for the Device Profiler sample application is in the following directory:

      Mobile Designer_directory/Samples/_DeviceProfiler_
   c. Click **OK**, and then click **Finish**.

2. Update the Device Profiler code to provide a mobile phone number that the Device Profiler can use for testing SMS messaging capabilities. When you run the Device Profile, during the SMS Test, the Device Profiler attempts to send a text message to the mobile number you provide.

   a. In the Project Explorer view, expand the following to access the TestWirelessMessaging class:

      src > runtime-sms_test

   b. In the runtime-sms_test folder, double-click TestWirelessMessaging.java to open it.

   c. In the TestWirelessMessaging.java file, locate the following lines:

      ```
      // TODO: FILL IN THIS PHONE NUMBER OF WHERE TO SEND THE SMS
      public String phone_number = null;
      ```

   d. Set a phone number for the `phone_number` String. For example:

      ```
      // TODO: FILL IN THIS PHONE NUMBER OF WHERE TO SEND THE SMS
      public String phone_number = "202-555-1234";
      ```

   e. Save your changes.

3. Open the Device Profiler project's build.xml file in the Ant view.

   a. In the Project Explorer view, locate the Device Profiler project.

   b. Expand the project and drag its build.xml file to the Ant view.

   If the Ant view is not open, for instruction, see "Displaying the Ant View" on page 65.

4. In the Ant view, double-click **+Multi-Build** to launch the Multi Build dialog.

5. In the Multi Build dialog, select the DefaultDevice.

   The Device Profiler project comes with this one device profile, DefaultDevice. The +Multi-Build Ant task cross compiles for the target platform, for example, Android or iOS, so that you can run the Device Profiler on the target device.

6. After building the Device Profiler, install and run it on the new target device.

   You can alter the settings for any particular build. Additionally, several settings are available using the ++Activate-Handset or the +Multi-Build JPanel to enable or disable features of the test suite when these features cause installation problems.

   If the device has problems displaying the screen at startup or on recovering from an interrupt, you might need to set the `MD_BASE_CALL_SERVICEPAINTS` parameter to `false` for the device and recompile.

   The Device Profiler requires that you perform manual steps that require your input. Select the **<SUMMARY>** option on the main menu to get a list of all the settings that the device requires. On the main menu, the Device Profiler denotes completed tests using an asterisk (*).

For details on the Device Profiler tests you need to perform, see "Device Profiler Tests to Determine Device Settings" on page 93.

Device Profiler Tests to Determine Device Settings

The following lists the Device Profiler tests you need to complete to determine the settings for a device.

Keypresses

On loading the Device Profiler, you are asked to perform the keypress test to define the keys the device has and also to define the keys being used for navigation in the Device Profiler application itself. This test checks for:

- Up, Down, Left, Right
- Action button (Fire)
- 0-9 numerical keys
- Back and Forward softkeys
- # (pound)
- * (asterisk)

When asked to press a key:

- If the device has the key you are asked to press, press that key.
- If a device does not have a key that you are asked to press, simply press a previously defined key, for example, 0, to skip this test.

  Caution: If you press a new key, one that you have not previously defined, the application assumes you are identifying that key to be the one the test is trying to define.

- If the device does not have a keypad, wait 5 seconds for the Device Profiler to continue to the main menu.

You might need to perform the keypress test again after you have configured the correct canvas in order to detect the key code values for the soft keys.

If you find that different firmware for the same device returns different keypress values, when you add the device profile using the +Add-Update-Handset Ant task, you can enter a comma-delimited list of values so that mobile applications that use this device will work on all versions of the device.

Touchscreen

Although there is no specific touchscreen test, the Device Profiler should automatically enable touchscreen support if it determines touchscreen support is needed. Alternatively, simply tap the screen, and if the particular device is touchscreen enabled, the Device Profiler should detect the touchscreen functionality. When touchscreen is enabled, the Device Profiler displays a 3x3 grid in the background of each screen. Tapping in the top, bottom, left and right squares emulates up, down, left and right keys

being pressed in the Device Profiler, while tapping in the center square emulates the fire key being pressed.

Canvas

The Canvas test sets the screen area for an application. For the Canvas test you choose the canvas that provides the most-usable screen area. Additionally, on some devices, you can manually alter the screen size beyond what the device returns to enable drawing outside of the indicated screen region.

System Font

The System Font test allows you to detect the best font to use on the device, and any adjustments that need to be applied when using it. Use Up and Down to cycle through the various options that you can change, with Left and Right changing the settings for each one.

Gfx Speed

The Gfx Speed test does not require user interaction. However, it is best to tap one of the device keys or the screen every few seconds to keep the phone from entering the screen saver mode because the screen saver mode might compromise the metrics that the Device Profiler is recording.

Keypress2

Sometimes a device needs an application to regularly call a threading sleep or wait so that the device can perform its other activities in the background. A normal indication that a device is not calling a threading sleep or wait when it needs to is that the application might run slow or keypresses might seem to lag. Use the Keypress2 test to adjust the sleep time per update to achieve the best keypress response time and frame rate.

Sprite Width

The Sprite Width test allows you to indicate whether transparent images with odd pixel widths display properly. For this test, the Device Profiler displays two images. You indicate whether the two images are the same.

Interrupts

The Interrupts test defines how to detect interrupts for the device. For this test, the Device Profiler prompts you to perform various interrupts.

Press the Fire key to initiate an interrupt test. The Device Profiler will not detect the interrupt if you do not press the Fire key. For each interrupt:

- If the interrupt is intrusive, press the key that the Device Profiler indicates.

  Caution: During this test, *only* press the keys that the Device Profiler indicates. Some devices use incoming key codes as indicators of certain interrupts.

- If the interrupt is non-intrusive, such as an incoming SMS resulting in an audio cue, press Right to skip a test.
- If the interrupt is irrelevant to the device, such as receiving a phone call on a non-mobile device, press Right to skip the test.

If you find a device does not detect a particularly intrusive interrupt through these tests, it might be worth trying some tests in the run-time code of the Device Profiler to determine whether other detection methods might work instead. If you find something new that works, contact Software AG can be applied to all other devices.

LCDUI Softkeys

The LCDUI Softkeys test defines LCDUI soft key functionality for the device. The Device Profiler application switches to MIDP1.0 canvas mode for this test. If the soft keys are not displaying as expected, you can alter the LCDUI soft key parameters. By using the soft keys themselves, you should be able to test the following soft key scenarios:

- Test 0: FWRD soft key
- Test 1: Both BACK and FWRD soft keys
- Test 2: BACK soft key

RotCanvas

The RotCanvas test determines whether a user of the device can rotate the screen and whether the rotation is detected at run time. If the user can rotate the screen, the Mobile Designer run-time code detects the screen size for this device on each update cycle. The Resource Handler uses the information about the standard screen dimensions.

Softkeys2

The Softkeys2 test determines whether the device detects LCDUI soft keys with the selected canvas. If the device does detect the soft keys, applications can use the soft keys to minimize screen intrusion.

Vibrate

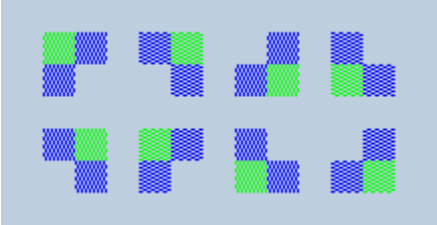
The Vibrate test determines whether the device has a vibration feature. The Device Profiler attempts to execute the vibration feature and then prompts you to indicate whether the device physically vibrated or not (basically we don't want to assume that just because the device API claimed it worked that it actually did).

Backlight

The Backlight test determines whether the backlight can be kept on continuously. The Device Profiler requires you to provide input to adjust and confirm the backlight timing.

Transforms

The Transforms test determines whether the run-time image transforms work as expected. The transforms are displayed in the repeating order, as shown in the following image:



The Device Profiler prompts you to compare the images it displays to the reference image shown above.

Browser Launch

The Browser Launch test determines whether the device can launch its Web Application Protocol (WAP) or Web browser successfully.

Caution: This test can crash the device or Device Profiler. Ensure that you have saved all test results so far.

SMS Test

The SMS test determines whether the device can send an SMS message.

Note: Before you build the Device Profile project, you need to update its TestWirelessMessaging class to set a mobile phone number for the `phone_number` String. The Device Profiler attempts to send a text message to the mobile number you provide, and you verify whether the SMS message was successfully sent.

Audio

The Audio test performs several audio checks that prompt for user feedback to determine the best format on the device, whether a piece of sound plays, and whether the volume is configurable. The test also determines the best volume at which to play a sound, as well as a few other minor tests.

In the Mobile Media API for J2ME, there are audio parameters that Mobile Designer cannot identify by running the Device Profiler. If the audio does not work properly, you need to manually set these parameters in Mobile Designer for the device using the procedure described in "Updating an Existing Device Profile in the Device Database" on page 90. The following are the parameters to update:

- `MD_SOUND_JSR135_MAX_PREFETCHED_PLAYERS` is set to `1` by default. This parameter determines how many player objects are in a pre-fetched state at any one time.

- `MD_SOUND_JSR135_CLEAN_PLAYER_AFTER_PLAYING` is set to `false` by default. This parameter de-allocates or ends the sound when it has finished playing.
- `MD_SOUND_JSR135_CLEAN_PLAYER_IF_STOPSOUND` is set to `false` by default. This parameter de-allocates or ends a sound when it is stopped.
- `MD_SOUND_JSR135_DELETE_PLAYERS` is set to `false` by default. This parameter deletes player objects when no longer in use. Some devices might need to retain the player for future use.
- `MD_SOUND_JSR135_DONT_INTERRUPT_PLAYING` is set to `false` by default. This parameter prevents audio from starting when audio is already playing.
- `MD_SOUND_JSR135_IGNORE_STOPSOUND` is set to `false` by default. This parameter ignores any attempts by the application code to force a sound to stop.
- `MD_SOUND_JSR135_KILL_IF_START_WHEN_PREFETCHED` is set to `false` by default. This parameter clears a player object if the application tries to start it when the player is in its pre-fetched state.
- `MD_SOUND_JSR135_LOADSOUND_CREATES_PLAYER` is set to `false` by default. This parameter ensures that the player object is created when the sound resource is loaded.

Adding a Device Profile to the Device Databases

After you determine the settings for a device by running the Device Profiler, you can use the device settings to manually add a new profile for a device. To add a new device profile, execute the +Add-Update-Handset Ant target, also known as the *Profile Updater*. The +Add-Update-Handset Ant target adds the new device to the Mobile Designer device database by creating an XML file for the device in the following directory:

Mobile Designer_directory/Devices

Note: After adding the device to the Mobile Designer device database:

- You might want to test the settings. For more information, see "Testing Settings in a Device Profile" on page 99.
- If you want to build an application for the device, add the device to the project. For more information, see "Adding a Device to a Project" on page 88.

Note: This procedure describes how to use the +Add-Update-Handset Ant target to add a new device profile. If you want to update an existing device profile, see "Updating an Existing Device Profile in the Device Database" on page 90.

To add a new device profile

1. If the Device Profiler project is not in Software AG Designer, import the project.

a. In Software AG Designer select **File > Import> General > Existing Projects into Workspace**, and then click **Next**.

b. In **Import Projects**, click **Browse** and select the project source directory of the project you want to clone.

The source code for the Device Profiler sample application is in the following directory:

Mobile Designer_directory/Samples/_DeviceProfiler_

c. Click **OK**, and then click **Finish**.

2. Open the Device Profiler project's build.xml file in the Ant view.

a. In the Project Explorer view, locate the Device Profiler project.

b. Expand the project and drag its build.xml file to the Ant view.

If the Ant view is not open, for instruction, see "Displaying the Ant View" on page 65.

3. In the Ant view, double-click **+Add-Update-Handset**.

4. In the Choose Handset dialog, type a name for the new device profile in the **Add a new handset** field.

When specifying the name, use the Mobile Designer naming convention for a device profile. The naming convention is the following:

platform _manufacturer _model

where:

- *platform* represents the platform
- *manufacturer* represents the manufacturer
- *model* represents the model name

For example, the name of the device profile for the AOC Android Breeze is *AND_AOC_Breeze* . *AND* represents the Android platform; *AOC* represents the manufacturer AOC, and *Breeze* represents the model.

To view samples of existing profile names, review the names in the **Choose your handset** list.

5. Click **Add**.

The +Add-Update-Handset Ant target displays the Update Handset Information dialog.

6. In the Update Handset Information dialog, select each parameter or property and enter the value for the device in the **Handset override** column. For the value, use the information you determined when running the Device Profiler.

7. Optionally, in **Comments** field, type a comment for the new device profile.

8. Click **Update Handset Information** when you have completed the device profile.

Testing Settings in a Device Profile

After adding a new device profile to Mobile Designer using the Add-Handset Ant target, you might want to test the settings. To do so, run the NativeUI Demo and Function Demo sample projects, which are provided with Mobile Designer, on the device to ensure those applications work correctly.

To run the sample applications on the device, first build the applications for the device using the +Multi-Build Ant task. For instructions, see "Building a Project for Multiple Target Devices" on page 139. Install the resulting builds for testing.

If needed, you can execute the Add-Handset Ant target again to alter device settings,. For instructions, see "Updating an Existing Device Profile in the Device Database" on page 90.

11 Defining Resources for a Mobile Application Project

About the Resource Handler

Each project requires its own resource handler that you code. The goals of the resource handler are to:

- Define all the resources to include with your mobile application, such as graphics, text, icons, and sounds.
- Set parameters that define the identifiers for resources, text lines, menus, and resource blocks. Mobile Designer includes the parameters in the Parameters.java. For more information about the Parameters class, see "Application and Parameter Classes" on page 72.

For information about how to code the resource handler for your project, see "Coding the Resource Handler" on page 102.

You also must store the resource files that your project requires, for example image files, text files, and audio files. Store the resource files in a location within your project's folder. For more information, see "Storing Resource Files for the Project" on page 106.

When building your project, Mobile Designer runs your project's resource handler before it compiles the run-time code. To let Mobile Designer know about the resource handler, for example, the name and location of your resource handler code and the location of your resource files, you set project properties. For more information, see "Setting Project Properties for the Resource Handler" on page 108.

After defining the resources for your project, you can use the resources in your application code. For more information, see "Accessing Resources in Your Application Code" on page 110.

Coding the Resource Handler

If you started your project by cloning a sample project provided with Mobile Designer, you can update the resource handler provided in the sample to work for your application. Alternatively, you create the resource handler from the beginning on your own.

Resource Handler Requirements

- Your resource handler must extend com.softwareag.mobile.reshandler.ResourceHandler. This class includes the projectResourceScript method.
- In the resource handler, include all resource handling logic that your project requires in the projectResourceScript method.

  When building your project, Mobile Designer calls the resource handler's projectResourceScript method.

Methods that Mobile Designer Provides for the Resource Handler

Mobile Designer provides the com.softwareag.mobile.reshandler.AntTaskResourceHandler that contains methods you can use in your resource handler.

The com.softwareag.mobile.reshandler.ResourceHandler class, which your project's resource handler must extend, includes the `rh` field that defines a link to AntTaskResourceHandler. As a result, you can easily execute the methods in the AntTaskResourceHandler using the following format, where *`method`* is the AntTaskResourceHandler method you want to invoke:

```
rh.method
```

For example, to use the AntTaskResourceHandleraddFile method to create a resource from a file, use the following:

```
rh.addFile
```

The types of actions you can accomplish using methods provided by the AntTaskResourceHandler include:

- Add resources to the project.
- Set and get IDs for resources, text lines, menus, and resource blocks.

  Note: When the resource handler sets identifiers, Mobile Designer adds corresponding parameters to the Parameters.java class. For more information, see "Setting Parameters in the Resource Handler Code" on page 121.

- Bundle resources into packs. For more information, see "Using Resource Blocks and Resource Packs" on page 104.
- Set application-specific parameters. For more information, see "Setting Parameters in the Resource Handler Code" on page 121.

To learn about *all* the methods that Mobile Designer provides for a resource handler, see information about com.softwareag.mobile.reshandler.AntTaskResourceHandler in *webMethods Mobile Designer Java API Reference*.

Setup to Allow the Resource Handler and Application Code to Share Common Code

You can set up your project's resource handler and the application code so that they share common code. For example, the resource handler and application code might use the same set of constant values, or you might have common code that you want to use in both the resource handler and the application code.

To use shared common code, place the shared code in a folder within your project. Then when defining the project.runtime.project.src.path and project.reshandler.src.path Ant paths for your project, include `<pathelement>` tags to the location that contains the shared code. For example, if you placed shared code in the project's src/shared_code folder, you might define Ant paths like the following in your project's _defaults_.xml file:

```
<path id="project.runtime.project.src.path">
   <pathelement path="${basedir}/src/core"/>
   <pathelement path="${basedir}/src/shared_code"/>
```

```
</path>

<path id="project.reshandler.src.path">
   <pathelement path="${basedir}/reshandler"/>
   <pathelement path="${basedir}/src/shared_code"/>
</path>
```

The _FunctionDemo_ sample application provides an example of this shared code setup.

Sample Resource Handler Code

You can find examples of basic and complex resource handler logic in all the samples applications provided with Mobile Designer.

Accessing Resources in Your Application Code

Your resource handler defines the resources available to your application. For information about how to use the resources in your application code, see "Accessing Resources in Your Application Code" on page 110.

Using Resource Blocks and Resource Packs

You can code your resource handler to put resources into *resource blocks* and *resource packs*.

- **Resource blocks** are a group of resources. For example, you might bundle splash screens into one block, images into another block, and audio files into another block.
- **Resource Packs** are bundles of resource blocks.

The use of resource blocks and packs is not required. Reasons to use them might be to save space and/or increase speed. A single larger file compresses better than multiple smaller files. Opening one file can be faster than opening several smaller files. However, with current devices, space and speed are no longer major issues for mobile applications. As a result, using resource bundles and packs for reasons of space and speed is not typically needed.

Another reason you might want to use resource blocks and packs is for better security. When bundling resources, it is more difficult to determine names of resources and more difficult to take malicious actions without having to rebuild the resource packs.

Defining Resource Blocks

To define resource blocks in the resource handler code, execute the startResourceBlock method that is in the com.softwareag.mobile.reshandler.AntTaskResourceHandler class. For information about how to use an AntTaskResourceHandler method in your resource handler, see "Methods that Mobile Designer Provides for the Resource Handler" on page 103.

When you execute the startResourceBlock method, you assign the resource block a name. You can assign resource blocks any name that is appropriate for your application. When you execute the startResourceBlock method, the resource block you create becomes the current resource block and all subsequent resources you add are included in the current

block. For example, the following code sample shows how to start a resource block named "IMAGES" and add the file "logo.png" as the resource name "res_logo.png":

```
public class ResHandler extends com.softwareag.mobile.reshandler.ResourceHandler
{
    public void projectResourceScript()
    {
          .
          .
          .
          rh.startResourceBlock ("IMAGES");
          rh.setResourceReadSubdirectory ("graphics"));
          rh.addFile ("res_logo.png", "logo.png");
          .
          .
          .
    }
}
```

To start a new resource block, execute the startResourceBlock method again.

Using resource blocks helps you to control the memory your application uses. When you bundle resources into a block, you can cache the block(s) your application needs based on the application state. In other words, your application can load and unload blocks so that only the resources that are required for a state in your application are loaded and therefore the application is not using valuable memory space for unneeded resources.

Important: If your resource handler bundles resources into blocks, when your application needs to use the resources that are in a block, be sure the application code caches the block into memory before loading its resources.

A best practice is to use the same resource blocks for all the platforms for which you build your application. However, if the devices on which your application runs have varying memory and you are concerned that some devices cannot keep all the resources in memory, you might want to split your resources up in the application bundle for compression, decompression, or run-time memory management. One approach you can use is to bundle the blocks into packs.

Using Resource Packs

Resource packs are bundles of resource blocks. You can define several packs of different combinations of your resource bundles. You might customize packs for each of the devices your application supports. For example, some devices might have the memory capacity to cache all the blocks in memory at application load time, while other devices might only be able to cache one block at a time due to memory limitations.

To define resource packs in the resource handler code, execute the allocateResourceBlockToPack method that is in the AntTaskResourceHandler class. When you execute the allocateResourceBlockToPack method, you identify a resource block and specify the identifier for a resource pack. For example, the following line of code allocates the resource block named "IMAGES" to the pack with identifier “0”:

```
rh.allocateResourceBlockToPack ("IMAGES", 0);
```

At build time when Mobile Designer runs the resource handlers, it creates an individual file for each resource pack that your resource handler defines. When you use resource

bundles, your resource handler should define, at a minimum, at least one resource pack that contains all the resource bundles. If a block is not included in a pack, Mobile Designer saves that block's resources as individual resources in the final binary.

The resource packs you define in your resource handler are transparent to your application code. When Mobile Designer runs the resource handler, it keeps a record of each resource pack along with the blocks that the pack contains. It also keeps track of the resources that are in each block. When your application code loads a resource block into memory, Mobile Designer determines the appropriate pack to load for that block. As a result, you can customize the packs for each device your application supports without being concerned about altering your application code.

Storing Resource Files for the Project

Save the files that contain the resources that your project uses in a folder within your project. You can set up any structure that is appropriate for your application.

The following shows an example setup for a project named "MyProject":

```
MyProject
   resources
      audio
      icons
      graphics
      text
```

In this example, all the resource files are stored within the "resources" folder. The "resources" folder has subfolders for the different types of resources.

Splash Screens for Applications

When defining the resources for your application, you should include a splash screen image. A *splash screen image* is a static image that a device displays when the user launches the mobile application. The purpose of the image is so that the user can see that the application is starting while the application initializes its initial window and views.

The requirements for splash screen images differ based on the platform on which the application is running.

Android Splash Screen Requirements

Use of splash screen images is discouraged except for game applications that require lengthy times to load.

For more information, refer to the Android developer website, http://developer.android.com/design/patterns/help.html#your-app.

iOS Platform Splash Screen Requirements

Use of splash screen images is required.

The Asset Catalog is used to manage these images. For details, see the information on launch images on the Apple developer website at https://developer.apple.com/library/ios/recipes/xcode_help-image_catalog-1.0/Recipe.html.

You can find the current Asset Catalog configuration for launch images under `MobileDesigner\Platforms\iOS\build\project\Images.xcassets_universal\Images.xcassets\LaunchImage.launchimage\`.

Windows Phone 8 Splash Screen Requirements

Use of splash screen images is discouraged. Windows Phone 8 applications should start quickly enough so that a splash screen is not required.

If you do use a splash screen for a Windows Phone 8 application, use one of the following naming conventions:

To use...	Do the following...
A single image for all Windows Phone 8 devices	Name the image file SplashScreenImage.jpg. The image should have the dimensions 768x1280px. On devices that have a different screen size, this image will be scaled to fit the screen.
Separate image files for each type of Windows Phone 8 device	Provide multiple image files based on the specific device sizes:

Image Size	Filename
480x800px	SplashScreenImage.screen-WVGA.jpg
768x1280px	SplashScreenImage.screen-WXGA.jpg
720x1280px	SplashScreenImage.screen-720p.jpg

For more information, refer to information about splash screens on the Microsoft website at http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769511%28v=vs.105%29.aspx.

Windows 8 (RT) Splash Screen Requirements

Use a splash screen image with the file name SplashScreen.png with the dimensions 620x300px.

Ensure the splash screen image is a transparent image because the image will not cover the entire screen. The operating system fills the background and draws the splash screen image centered on the device screen. By default, the background color is set to black, but you can set a different color inside the Visual Studio project.

For more information, refer to information about splash screens on the Microsoft website at http://msdn.microsoft.com/en-us/library/windows/apps/br211467.aspx and http://msdn.microsoft.com/en-us/library/windows/apps/hh465338.aspx.

Setting Project Properties for the Resource Handler

The following table lists the resource handler project properties. The table specifies the properties you are required to set to provide details about the resource handler for your project. For instructions about how to set properties, see "Where You Set Properties" on page 116 and "Setting Project Properties" on page 119. For further details about the properties, see "Resource Handler Properties" on page 261.

Property and Description
project.java.reshandler.name Required. Specifies the Java package/class name of the resource handler class.
project.reshandler.src.path Required. Specifies the location of your project's resource handler script and any associated classes.
project.resource.dir.root Required. Specifies the location of the resource files (audio files, image files, etc.) for your project.
project.reshandler.additional.libs.path Conditionally required. Specifies the location of classes that the resource handler requires. Required only if your resource handler requires additional classes.
mobiledesigner.run.reshandler.with.beanshell Optional. Specifies whether to use the BeanShell provided with Mobile Designer or the one provided with Software AG Designer.

Property and Description
The default is to use the BeanShell provided with Software AG Designer.
debug.remember.resource.names Optional. Specifies whether you want the Mobile Designer to record the names of the resources included in the build rather than the resource IDs. By default, the resource IDs are recorded, not the resource names.
project.compiled.resources.info.format Optional. Specifies whether you want the _compiled_resources file that Mobile Designer creates when it runs your resource handler to be in XML or text format. For more information, see "Managing Memory for Your Resource Handler and Resources" on page 109. By default, Mobile Designer creates a .txt format file.

The following table lists the resource handler project properties that are driven based on the device profiles in the Mobile Designer device database. It is recommended that you do not change the settings.

Property and Description
project.jar.midlet.icon.spec Specifies the icon(s) to use for the application's MIDLet-icon for a specific device.
project.audio.spec Specifies the type of audio (for example, mp3 or wav) that a device supports.
project.audio.file.extensions Specifies the file extension (for example, .mp3 or .wav) that a device supports.

Managing Memory for Your Resource Handler and Resources

If your mobile application project's resource compilation is memory and resource intensive, you can increase the amount of memory available for Ant to avoid encountering out-of-memory exceptions during the execution of the resource handler. To increase this memory, use a system property to set the amount of memory. The following shows examples:

- On Windows: `set ANT_OPTS=-Xmx256m`

- On Linux: `export ANT_OPTS=-Xmx256m`

Adjust the value, "256m", used in the example to reflect how much memory your resource handling needs.

You can estimate the amount of memory your application will require when it runs using the information in the _compiled_resources_ file, which resides in your project's _temp_ folder. Mobile Designer creates this file when it runs your project's resource handler. Use the project.compiled.resources.info.format project property to indicate whether you want the _compiled_resources file to be in XML or text format. The _compiled_resources_ file contains information on the resources, as well as their resultant resource blocks, resource packs, and file sizes.

Accessing Resources in Your Application Code

To access resources in your application code, Mobile Designer provides the following run-time classes:

- com.softwareag.mobile.runtime.media.AudioHandler

  Use in your application code to access and manage the audio resources. The AudioHandler class includes the loadSound and unloadSound methods that you can use to load and unload audio resources.

- com.softwareag.mobile.runtime.media.ImageBase and com.softwareag.mobile.runtime.media.ImageHandler

  Use in your application code to access and manage image resources. The ImageBase class includes the getImage, loadImageID, and unloadImageID methods that you can use to load and unload image resources.

- com.softwareag.mobile.runtime.media.TextHandler

  Use in your application code to access and manage text line resources. The TextHandler class includes the getString method to get a text string.

- com.softwareag.mobile.runtime.storage.ResourceHandler

  If you use resource blocks and packs, use this class in your application code to manage the blocks and packs. The ResouceHandler class includes the loadResourceBlock and unloadResourceBlock methods that you can use to load and unload resource blocks.

When using a method to load or unload a resource, you specify the resource's identifier. When Mobile Designer runs your resource handler, it sets parameters for the resources in the com.softwareag.mobile.runtime.Parameters class. For example, if you want to load an audio resource with the identifier RESID_STARTUP, you can use the following method call:

```
AudioHandler.loadSound (Parameters.RESID_STARTUP);
```

If your resource handler bundles resources into resource blocks and packs, in your application code, you must load the resource bundle containing a resource before you load the specific resource. Continuing with the previous example, suppose the

RESID_STARTUP audio resource is included in the resource block with the identifier RESBLOCKID_AUDIO, you can use the following method calls:

```
ResourceHandler loadResourceBlock (Parameters.RESBLOCKID_AUDIO);
   AudioHandler.loadSound (Parameters.RESID_STARTUP);
```

When the application is no longer using the resources in a block, it can use the unloadResourceBlock, which is also in the ResourceHandler class to unload the resource block.

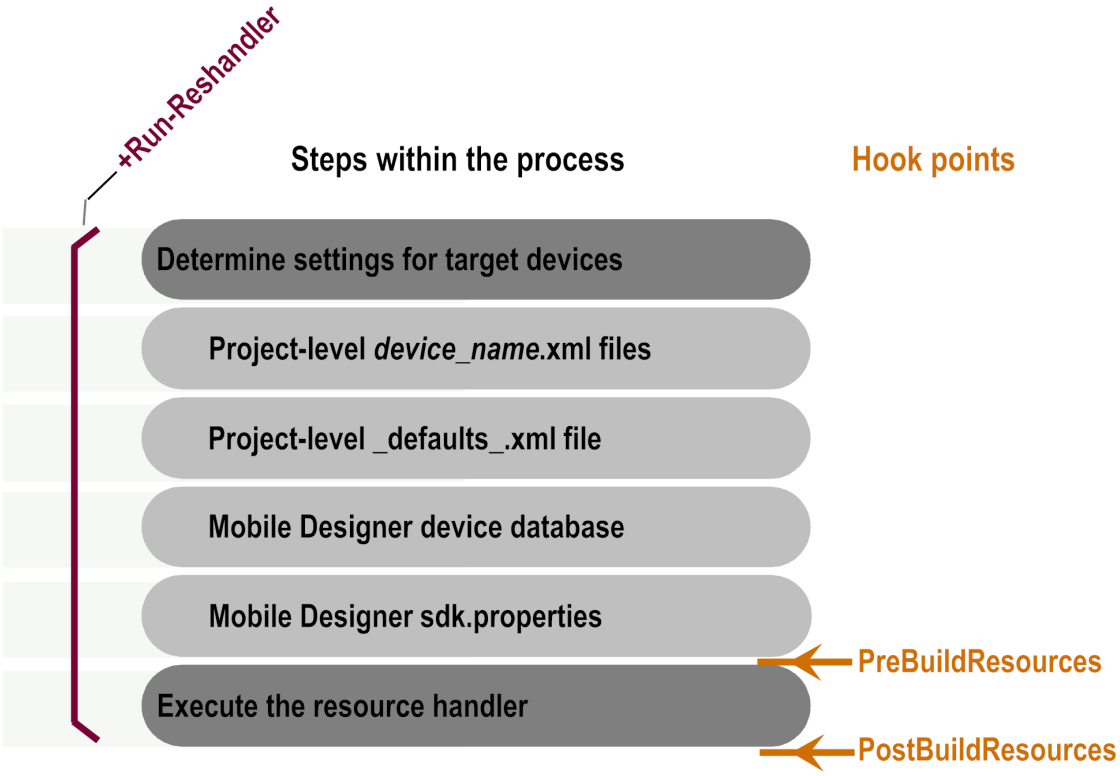
Note: If you are using resource blocks and packs, your application code does not need to load and unload resource packs. Mobile Designer keeps track of the packs required for each resource block. When you load a resource block, Mobile Designer takes care of loading the appropriate pack that contains the resource block.

Compiling Resources Using the +Run-Reshandler Ant Target

Use the +Run-Reshandler Ant target if you want to compile the resources, such as text files and images, for the current device without compiling the source code for the mobile application.

The following diagram shows the steps that Mobile Designer performs when you use the +Run-Reshandler Ant target. See the table below the diagram for a description of the steps. The table also indicates hook points where Mobile Designer can run custom Ant scripts that you provide. For more information about hook points, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

+Run-Reshandler Ant Target



1 Determine settings for target devices

Mobile Designer determines the settings for the device for which it is building the application. It retrieves the settings from the following sources in the order listed.

- Project-level *device_name* .xml files
- Project-level _defaults_.xml file
- Mobile Designer device database
- Mobile Designer sdk.properties file

Mobile Designer uses the first setting it encounters. For example, if Mobile Designer encounters a setting in the project-level target *device_name* .xml file and then again in the project-level _defaults_.xml file, Mobile Designer uses the setting from the target *device_name* .xml file.

For more information about:

- Project-level device files, see "Where You Set Properties" on page 116 and "Setting Project Properties" on page 119

- Mobile Designer device database, see "Devices that a Mobile Application Supports" on page 88
- Mobile Designer sdk.properties file, see "Mobile Designer Configuration Properties (sdk.properties)" on page 19

hook point **PreBuildResources**

If you have created an Ant script to run at the PreBuildResources hook point, Mobile Designer runs the Ant script.

2 **Execute the resource handler**

Mobile Designer runs the resource handler that you created for the project. When running the resource handler, Mobile Designer records all the resources required for your application. Mobile Designer also creates the com.softwareag.mobile.runtime.Parameters class. For more information about creating the resource handler, see "About the Resource Handler" on page 102. For more information about the Parameters class, see "Application and Parameter Classes" on page 72.

At this point in the build process, Mobile Designer uses the following project properties:

- project.reshandler.src.path for the location of the project's resource handler script and any associated classes
- project.java.reshandler.name for the name of the resource handler class you created for your project.
- project.resource.dir.root for the location of the top-level folder that contains the resources (audio files, image files, etc.) for your project
- project.reshandler.additional.libs.path for the location of additional libraries that your project's resource handler requires

For more information about these properties, see "Resource Handler Properties" on page 261.

hook point **PostBuildResources**

If you have created an Ant script to run at the PostBuildResources hook point, Mobile Designer runs the Ant script.

12 Setting Properties and Parameters for a Mobile Application Project

About Properties and Parameters

Properties and parameters are project settings used when building your project or at run time.

- **Properties** are build-time settings that your project's build script can access when building your project. At build time, the build process can access property settings to determine information, for example, the device for which your application is being built. As a result, you can set properties to manipulate how your application is built for specific platforms and/or specific target devices, including and/or excluding features for the devices your mobile application supports.

  Note: Your application code *cannot* reference property settings at run time. Use parameters for settings that are available to your application's run-time code.

- **Parameters** are run-time attributes that you can use in your application code. For example, at run time your application can access parameters to retrieve information abouts the resources included in the application to perform such tasks as loading resources. Another example is that your application code can access information about the device on which the application is running to branch the logic based on the needs of that specific device. For more information, see "Using Parameters in Your Application Code" on page 123.

  When Mobile Designer builds a project, it runs the project's resource handler before it compiles your application. The resource handler creates the com.softwareag.mobile.runtime.Parameters class that includes all the parameters.

  The Parameters.java includes:

  - Parameters that Mobile Designer defines and uses.
  - Application-specific parameters related to the resources in your project.

Where You Set Properties

Properties are generally set in the following files. When you set properties, be aware that when Mobile Designer builds a project, it obtains properties from the files in the order listed and uses the first setting it encounters for a property.

- Project-specific target device files (*device_name* .xml) that contain device-specific settings for devices that the project supports. For more information about target device files, see "Devices that a Mobile Application Supports" on page 88 and "Adding a Device to a Project" on page 88.

  Mobile Designer sets some properties in a target device file when you add a device to a project. You can change and/or add additional project properties. For instructions, see "Setting Project Properties" on page 119.

- Project-level _defaults_.xml file that contains default settings for all devices in a project.

  To start a project, you typically clone an existing project. As a result, your project starts with the settings in the cloned project. You can change and/or add additional properties. For more information, see "Setting Project Properties" on page 119.

- Mobile Designer device database contains profiles for devices. The device profiles that Mobile Designer provides have properties set. You can update the settings by updating the device profiles. For more information, see "Devices that a Mobile Application Supports" on page 88 and "Updating an Existing Device Profile in the Device Database" on page 90. The changes you make in the device profiles apply to all projects.

  If you want to override settings for your mobile application project, set project properties in the project's target device files for the device. For instructions, see "Setting Project Properties" on page 119.

- The Mobile Designer sdk.properties contains default property settings that affect all projects. For more information, see "Updating the sdk.properties File to Configure Mobile Designer" on page 18 and "Mobile Designer Configuration Properties (sdk.properties)" on page 19.

Project Properties You Must Set

The following table lists the properties you must set for your project. For information about how to set properties, see "Setting Project Properties" on page 119.

Property
project.runtime.project.src.path Required. Specifies the location of the run-time code for your mobile application.
project.runtime.additional.classes.path Conditionally required. Specifies the location of classes required for building your project. Required only if your project requires additional precompiled classes to build the application.
project.runtime.additional.stubs.path Conditionally required. Specifies the location of the stubs required for compilation. Required only if your project requires additional stubs to compile the application's run-time source code.
project.langgroup.group_name

Property
Required. Specifies the language(s) that you want your application to support.
project.jarname.format Required. Specifies the file name format that you want Mobile Designer to use when naming your application's final binary.
project.java.reshandler.name Required. Specifies the Java package/class name of the resource handler class you created for your project.
project.reshandler.src.path Required. Specifies the location of your project's resource handler script and any associated classes.
project.resource.dir.root Required. Specifies the location of the resource files (audio files, image files, etc.) for your project.
project.reshandler.additional.libs.path Conditionally required. Specifies the location of classes that the resource handler requires. Required only if your resource handler requires additional classes.
mobiledesigner.buildscript.version Required. Specifies the version of the Mobile Designer build scripting system to use when building your application.
mobiledesigner.runtime.version Required. Specifies the version of the Mobile Designer run-time system to use for your application.
project.java.midlet.name Required. Specifies the name of the root MIDlet/Application class of your project's run-time code. Typically this is the class that extends com.softwareag.mobile.runtime.core.Application.
project.jar.name Required. Specifies a text name you want your application to have when installed on a device.

Setting Project Properties

Mobile Designer defines values for most properties. For a list the properties you must set for your project, see "Project Properties You Must Set" on page 117.

To set a value or change a predefined value for your project, you can set property values within the project's targets folder.

- Specify properties in the project's _defaults_.xml file apply to the settings to *all* devices that the application supports.
- Specify properties in target device XML files to apply settings to a single device that the application supports. The properties you place in target device XML files override settings in the _default_.xml file.

Note: The properties you specify for the project override settings you make for all projects in the sdk.properties file.

To set project properties

1. In Software AG Designer, in the Project Explorer view, expand the Mobile Designer project for which you want to define properties.
2. In the project's targets folder, double-click the file to which you want to add a property:
   - To apply the setting to *all* the devices the application supports, double-click the _defaults_.xml file.

     project_folder /targets/_defaults_.xml
   - To apply the setting to a single device that your application supports, add the property to the XML file for that device:

     project_folder /targets/*device_file* .xml

     For example, if your application supports the Apple iPhone 5 and you want to apply the settings only for this device, add the settings to the IOS_Apple_iPhone5.xml file.
3. Locate the area of the file where you want to add the property. Refer to comments in the file to find good location.
4. Add the property to the file using the following format:

   ```
   <property name="PropertyName" value="PropertyValue"/>
   ```

   For example:

   ```
   <property name="cross.compiler.extractinners" value="true"/>
   ```
5. After adding properties, save the file.

Where You Define Parameters

You define application-specific parameters in the following files:

- You define application-specific parameters related to the resources in your project in the resource handler code. For more information, see "Setting Parameters in the Resource Handler Code" on page 121.
- You define general application-specific parameters in the following locations:
  - Project's _defaults_.xml and target device files. For more information, see "Setting Parameters in the _defaults_.xml and Target Device Files" on page 120.
  - Project's resource handler that you create for a project. For more information, see "Setting Parameters in the Resource Handler Code" on page 121.

    You can define any application-specific parameters you might need.

Setting Parameters in the _defaults_.xml and Target Device Files

To add parameters to the project's _defaults_.xml or one of the project's target device files (*device_name* .xml), use the following format:

```
<param name="ParameterName" value="ParameterValue"/>
```

You can also optionally include a comment when setting the parameter:

```
<param name="ParameterName" value="ParameterValue" comment="comment"/>
```

Specifying the Parameter Name

For *ParameterName* specify the name you want to use. The name can be anything that is acceptable as a Java variable.

Specifying the Parameter Value

For *ParameterValue* use one of the formats specified in the following table.

For this type of value...	Use this format to specify the value...

boolean	`boolean={"true" | "false"}`
	or
	`bool={"true" | "false"}`

byte	`byte=value`

For this type of value...	Use this format to specify the value...
char	`char=value`
double	`double=value`
float	`float=value`
int	`int=value`
long	`long=value`
short	`short=value`
string	`string=value`

Example

Suppose at run time your application code needs to determine whether to display a splash screen. To set the default for all devices to display the splash screen, you can add a parameter named "DISPLAY_SPLASH_SCREEN" to the _defaults_.xml file and set its value to "true":

```
<param name="DISPLAY_SPLASH_SCREEN" bool="true"/>
```

The following shows the resulting parameter declaration in Parameters.java:

```
public static final boolean PARAM_DISPLAY_SPLASH_SCREEN = true;
```

Then for devices for which you do not want to display the splash screen, you can add the parameter to the target device file for those devices and the parameter value to "false":

```
<param name="DISPLAY_SPLASH_SCREEN" bool="false"/>
```

If you build your project for one of the devices for which the splash screen should not be displayed, Mobile Designer reads the target device file, and as a result, uses the following parameter declaration in Parameters.java:

```
public static final boolean PARAM_DISPLAY_SPLASH_SCREEN = false;
```

Setting Parameters in the Resource Handler Code

When you code the resource handler for your project, you can add application-specific parameters. Additionally, for each resource you add to the project and to which you assign an ID, an associated parameter is included in the Parameters.java class.

In your resource handler code, to define the parameters, you invoke methods that Mobile Designer provides. The methods are defined in the com.softwareag.mobile.reshandler.AntTaskResourceHandler class. For information about how

to use the AntTaskResourceHandler methods in your resource handler, see "Methods that Mobile Designer Provides for the Resource Handler" on page 103.

The following table lists:

- The AntTaskResourceHandler method you use to define a parameter.
- The naming convention that Mobile Designer uses for the parameter names. In the naming conventions, *name* is the name assigned to the parameter and *id* is the identifier assigned to the parameter

Method	Naming convention for added parameter	Description
n/a	PARAM_MD_*name*	Parameters that Mobile Designer defines. These parameters are for internal use. The Mobile Designer run-time classes use these parameters. For a description of the run-time classes, see " Mobile Designer-Provided Run-Time Classes" on page 72.
setParam	PARAM_*name*	General application-specific parameters that you define.
setResBlockID	RESBLOCKID_*name*	Parameters that define the identifiers for resource blocks that you defined in your project. For more information about resource blocks, see "Using Resource Blocks and Resource Packs" on page 104.
setResID	RESID_*name*	Parameters that define the identifiers for resources (for example, audio files or image files, etc.) that you added to your project.
setTextID	TEXTID_*name*	Parameters that define the identifiers for lines of text that you added to your project.
setMenuID	MENUID_*name*	Parameters that define the identifiers for menus that you added to your project.

The following list examples:

- If you use the setParam method to add the application-specific parameter "DISPLAY_SPLASH_SCREEN", Mobile Designer includes the parameter `PARAM_DISPLAY_SPLASH_SCREEN` in the Parameters.java class.

- If you use the setResID method to assign an audio file the identifier “BEEP”, Mobile Designer includes the parameter `RESID_BEEP` in the Parameters.java class.
- If you use the setResBlockID method to assign a resource block the identifier “AUDIO”, Mobile Designer includes the parameter `RESBLOCKID_AUDIO` in the Parameters.java class.

Using Parameters in Your Application Code

When creating an application, typically you have common logic that works for all target devices. However, you might require branches in the logic to address the needs of a specific target device. For example, you might need to omit or alter a feature for a target device, or you might need to position an image relative to the screen dimensions for a target device. To accommodate device-specific logic, your application logic can branch based on parameter values that are set using the device profile settings.

To use parameters in your application code, import the com.software.mobile.runtime.Parameters class. You can then easily access parameter values. For example, if you want to branch logic based on an application-specific “PARAM_DISPLAY_SPLASH_SCREEN” parameter, you can use a statement like the following:

```
if (Parameters.PARAM_DISPLAY_SPLASH_SCREEN)
```

You also use parameters that define identifiers for resources and resource blocks to load resources and resource blocks into memory, and unload them from memory, as necessary.

```
ResourceHandler.instance.loadResourceBlock (Parameters.RESBLOCKID_SPLASHES);

splash_screen_image_id = ImageHandler.instance.loadImageID
                         (Parameters.RESID_SPLASH_SCREEN_IMAGE, -1, -1, false);

ResourceHandler.instance.unloadResourceBlock (Parameters.RESBLOCKID_SPLASHES);
```

For more examples about how to use the parameters in application code, review the code provided with sample projects that are provided with Mobile Designer.

IV Building and Compiling Mobile Application Projects

13 Build Process Overview

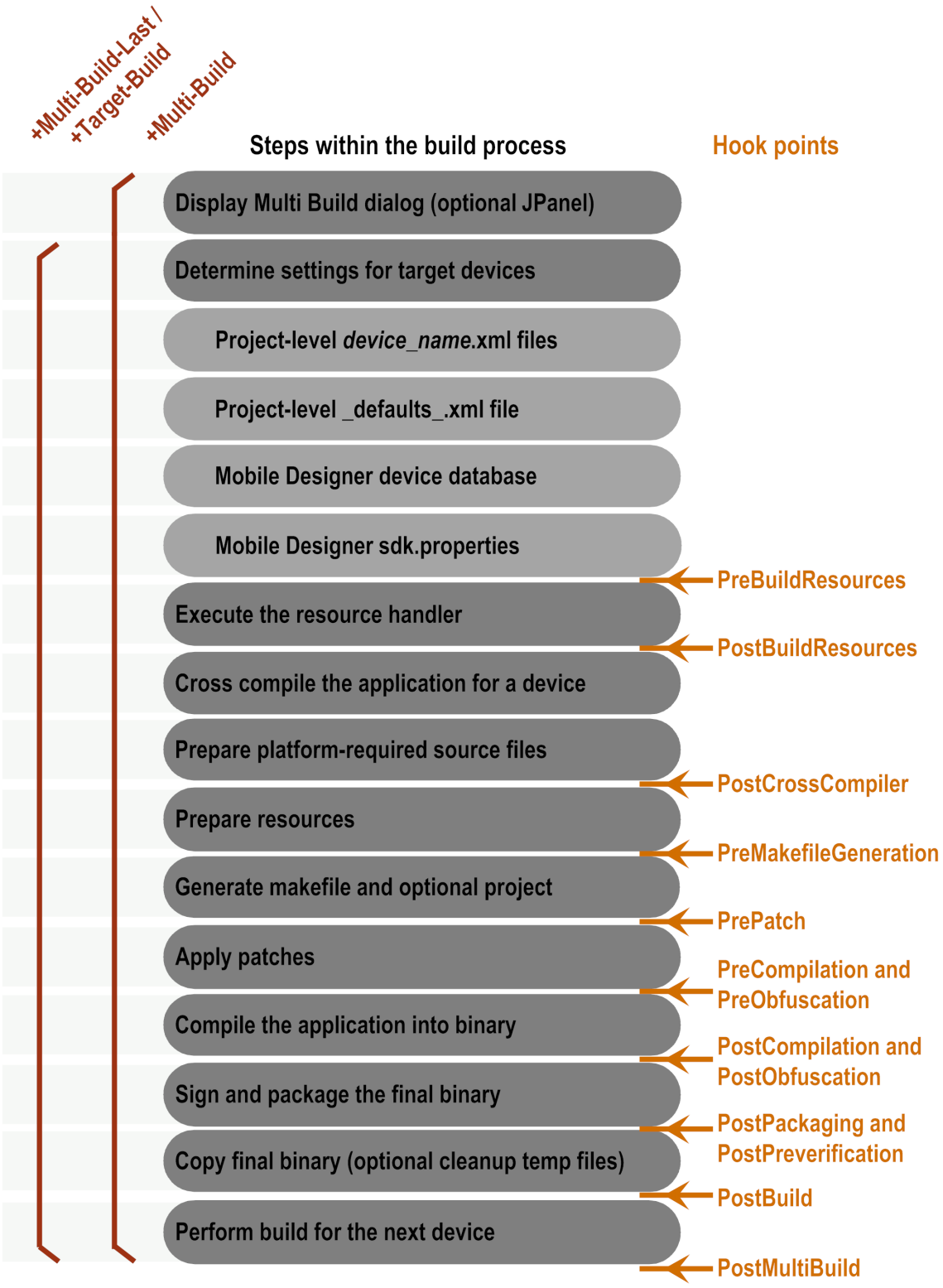
Build Ant Target Summary

The following diagram provides summary information about the process Mobile Designer performs to build a mobile application project. It shows:

- The names of the Ant targets you can use to build a mobile application project.
- The steps Mobile Designer performs in the build process. For details about each step, see "Steps in the Multi-Build Process" on page 130.
- The hook points in the build process where Mobile Designer can run custom Ant scripts that you provide. For more information about hook points, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

Note: For instructions for how to perform a build of your mobile application project, see "Building Mobile Applications" on page 137.

Multi-Build Ant Targets



Steps in the Multi-Build Process

This section describes the steps that Mobile Designer performs when you run the +Multi-Build, +Multi-Build-Last, or +Target-Build Ant targets to build a mobile application project.

The table also indicates hook points where Mobile Designer can run custom Ant scripts that you provide. For more information about hook points, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

Note: For instructions for how to perform a build of your mobile application project, see "Building Mobile Applications" on page 137.

1 Display the Multi Build dialog (optional JPanel)

Mobile Designer displays the Multi Build dialog, and also a custom JPanel for the build, if you defined one.

For more information, see "Setting Properties at Build Time Using a Custom JPanel" on page 146.

Note: Mobile Designer only displays the Multi Build dialog when you run the +Multi-Build Ant target.

2 Determine settings for target devices

Mobile Designer determines the settings for the device for which it is building the application. It retrieves the settings from the following sources in the order listed.

- Project-level *device_name* .xml files
- Project-level _defaults_.xml file
- Mobile Designer device database
- Mobile Designer sdk.properties file

Mobile Designer uses the first setting it encounters. For example, if Mobile Designer encounters a setting in the project-level target *device_name* .xml file and then again in the project-level _defaults_.xml file, Mobile Designer uses the setting from the target *device_name* .xml file.

For more information about:

- Project-level device files, see "Where You Set Properties" on page 116 and "Setting Project Properties" on page 119
- Mobile Designer device database, see "Devices that a Mobile Application Supports" on page 88

- Mobile Designer sdk.properties file, see "Mobile Designer Configuration Properties (sdk.properties)" on page 19

hook point **PreBuildResources**

If you have created an Ant script to run at the PreBuildResources hook point, Mobile Designer runs the Ant script.

3 **Execute the resource handler**

Mobile Designer runs the resource handler that you created for the project. When running the resource handler, Mobile Designer records all the resources required for your application. Mobile Designer also creates the com.softwareag.mobile.runtime.Parameters class. For more information about creating the resource handler, see "Defining Resources for a Mobile Application Project" on page 101. For more information about the Parameters class, see "Application and Parameter Classes" on page 72.

At this point in the build process, Mobile Designer uses the following project properties:

- project.reshandler.src.path for the location of the project's resource handler script and any associated classes
- project.java.reshandler.name for the name of the resource handler class you created for your project.
- project.resource.dir.root for the location of the top-level folder that contains the resources (audio files, image files, etc.) for your project
- project.reshandler.additional.libs.path for the location of additional libraries that your project's resource handler requires

For more information about these properties, see "Resource Handler Properties" on page 261.

hook point **PostBuildResources**

If you have created an Ant script to run at the PostBuildResources hook point, Mobile Designer runs the Ant script.

4 **Cross compile the application for a device**

Mobile Designer copies your project's application code for the target device into the project's _temp__src_ folder. At this point in the build process, Mobile Designer uses the following project properties:

- project.runtime.project.src.path for the location of your project's application code
- project.runtime.additional.classes.path for the location of additional precompiled classes that your application requires

- project.runtime.additional.stubs.path for the location of additional stubs to use when compiling your application code

For more information about these properties, see "Run-Time Code Compilation Properties" on page 273.

Mobile Designer converts the mobile application source code into a neutral language, for example, C++ or Java. The neutral language that Mobile Designer uses is the required neutral language for the target device. You can set the following code conversion properties to customize how Mobile Designer converts the application source code:

- cpp.no.selfprotect for whether to add a safeguard to every method call
- cross.compiler.extractinners for whether to extract inner classes included in the Java source code
- cross.compiler.nodatestamp for whether to include timestamps at the top of every generated C++ (CPP) and H file
- cross.compiler.render.selfprotect for whether to add an extra line at the top of every non-static method
- java.parser.retain.comments for whether to retain comments when compiling

For more information about these properties, see "Code Conversion Properties" on page 218.

The neutral language code that Mobile Designer generates from converting your application code is easy-to-read and, by default, retains comments from the original Java source code.

Note: When Mobile Designer generates C++ code for C++-based platforms (for example, iOS), the C++ code it generates will be the same format for all C++-based platforms. The generated C++ code is uniform, ANSI-compatible, and compiles against all target architectures (such as x86, MIPS, and ARM processors).

5 Prepare platform-required source files

Mobile Designer gathers other source files that it requires to make the final binary. For example, additional platform-specific files might be needed for the final binary.

hook point PostCrossCompiler

If you have created an Ant script to run at the PostCrossCompiler hook point, Mobile Designer runs the Ant script.

6 Prepare resources

Mobile Designer gathers the resources that the project requires and prepares them for building the final binary. These includes, for example, the

resources defined by your project's resource handler, NativeUI resources, and resources that the specific platform might require.

hook point **PreMakefileGeneration**

If you have created an Ant script to run at the PreMakefileGeneration hook point, Mobile Designer runs the Ant script.

7 **Generate makefile and optional project**

Mobile Designer creates the makefile that defines everything that is required to create the final binary.

Additionally, if appropriate for the target platform, Mobile Designer creates the platform-specific project. For example, when building an application that runs on the iOS platform, Mobile Designer creates an Xcode project or when building an application that runs on the Android platform, it creates an Eclipse project.

You can set the cross-compiler properties that affect this step in the build process:

- project.handset.custom.stubfolder for whether to override the makefile and project template files that are provided with Mobile Designer with makefile and project template files that you supply
- Several properties that customize the generation of the makefile. See "Makefile Additions" on page 232.

hook point **PrePatch**

If you have created an Ant script to run at the PrePatch hook point, Mobile Designer runs the Ant script.

8 **Apply patches**

If you created patches, Mobile Designer applies the patches to the cross-compiled code. For more information about when to use patches and how to create patches, see "Creating Patch Files to Apply to the Cross-Compiled Code" on page 152.

Note: If you set the project.handset.skip.compilation property to `true`, Mobile Designer skips the following actions.

hook point **PreCompilation and PreObfuscation**

If you have created Ant script(s) to run at the PreCompilation and/or PreObfuscation hook points, Mobile Designer runs the Ant script(s).

9 **Compile the application into binary**

Mobile Designer executes the makefile it created in a previous step to create binary for the target device. During this step Mobile Designer requires the platform-specific SDKs that you installed when setting up the platform. For more information, see "Setting Up Platforms" on page 27.

hook point **PostCompilation and PreVerification**

If you have created Ant script(s) to run at the PostCompilation and/or PreVerification hook points, Mobile Designer runs the Ant script(s).

10 **Sign and package the final binary**

Mobile Designer packages the output of the build so that is ready for installation on the target device. The packaging process varies from platform to platform. For example, for Android builds, the output is in an application package (apk) file.

Additionally, if required by the target platform, Mobile Designer signs the final build.

hook point **PostPackaging and PostVerification**

If you have created Ant script(s) to run at the PostPackaging and/or PostVerification hook points, Mobile Designer runs the Ant script(s).

11 **Copy final binary (optional cleanup temp files)**

When Mobile Designer builds your project, it adds a Builds folder to your project folder. Within the Builds folder, Mobile Designer creates a folder for the device for which it is building your application:

project /Builds/*x.y.z* /*device*

In the folder location above:

- *project* is the name of your project.
- *x.y,z* is the version number you specified for the build.
- *device* is the name of the device for which Mobile Designer is building your application.

The *device* folder contains a _temp_ folder. Mobile Designer stores temporary files in the _temp_ directory while it is building the application. After it creates the final binary, Mobile Designer copies the final binary from within the _temp_ folder into the *device* folder.

Mobile Designer then optionally performs cleanup. If you clear the **Retain output build files** check box in the Multi Build dialog when you started the build, Mobile Designer cleans up the temporary files by deleting the _temp_ folder.

Note: If you set the project.handset.skip.compilation property to `true`, Mobile Designer resumes processing with the following actions.

hook point **PostBuild**

If you have created an Ant script to run at the PostBuild hook point, Mobile Designer runs the Ant script.

12 **Perform build for the next device**

If you select multiple devices/language combinations in the Multi Build dialog to have Mobile Designer build your application for multiple device/language combinations, Mobile Designer restarts the build process at step 2 for the next device/language to build.

hook point **PostMultiBuild**

If you have created an Ant script to run at the PostMultiBuild hook point, Mobile Designer runs the Ant script.

14 Building Mobile Applications

About Building a Mobile Application Project

Before you can build your project, you must ensure you have completed the required prerequisite tasks. For more information, see "Before You Can Build a Mobile Application Project" on page 138.

To build your project, including compiling your project into binary and packaging your application for target devices, use the following Ant Targets:

- +Multi-Build to build your project for multiple device/language combinations. For more information, see "Building a Project for Multiple Target Devices" on page 139.
- +Multi-Build-Last to rebuild your project for the last target device. For more information, see "Building a Project for the Last Target Devices" on page 140.
- +Target-Build to build your project from the command line. For more information, see "Building a Project from the Command Line" on page 140.

If you want, you can use Mobile Designer to convert your mobile code into a neutral language (for example, C++ or Java), but then use your own platform-specific, native tools to compile your project into binary and package the application for the target device. For more information, see "Using Native Tools to Create the Final Binary" on page 141.

For details about the actions that Mobile Designer performs to build your project, see "Build Process Overview" on page 127.

Before You Can Build a Mobile Application Project

Before you build your mobile application project, be sure you have completed the following tasks:

- Perform platform-specific setup on your machine

  You must have the proper setup for the target platform(s). The setup requires that you have items such as compilers and SDK installed on your machine. For example, if you want to build an Android application, you must have the Android SDK installed on your machine.

  In addition to installing platform-specific tools, you must also set properties in the Mobile Designer sdk.properties file to indicate where the SDKs, compilers, and other tools are located.

  For information about how to set up your platform, see "Setting Up Platforms" on page 27.

- Set required project properties

  Mobile Designer requires that you set some project properties for your project. For example, you need to set the `project.runtime.project.src.path` property to

specify the location of your project's run-time code. For a list of the required project properties, see "Setting Project Properties" on page 119.

Building a Project for Multiple Target Devices

To build your mobile application project, you typically use the +Multi-Build Ant target. The +Multi-Build Ant target allows you to build your project for multiple device/language combinations.

For information about the actions that Mobile Designer performs to build your project, see "Build Process Overview" on page 127.

Note: Before you build your project, be sure you have completed the required setup. See "Before You Can Build a Mobile Application Project" on page 138.

To build a project for multiple targets

1. In Software AG Designer, open the project you want to build.
2. Click the **Project Explorer** view, expand the project, and drag the build.xml file to the **Ant** view.

   If the Ant view is not open, for instruction, see "Displaying the Ant View" on page 65.
3. In the **Ant** view, double-click **+Multi-Build** to launch the Multi Build dialog.

   Note: If you created a custom JPanel for the project, Mobile Designer includes the JPanel on the right side of the Multi Dialog. For more information, see "Setting Properties at Build Time Using a Custom JPanel" on page 146.
4. In the Multi Build dialog, select the device/language combinations for which you want to build the project.
5. In the **Version** field type the version number for the build.
6. Select or clear the **Retain output build files** check box.
   - Select the check box to retain the files from the build.

     Mobile Designer retains the cross-compiled code it generated from your original Java code, along with any project (for example, Xcode project for iOS) it might have generated.

     Choose to retain the files if you need to create a patch for your project, if you want to save the output for subsequent testing, or if you want to use native platform tools to compile to create the final binary for your project.
   - Clear the check box if you do not need the files from the build.
7. Click **Multi Build**.

Building a Project for the Last Target Devices

If you want to rebuild your project for the last target devices, use the +Multi-Build-Last Ant task. Mobile Designer rebuilds the project for all the devices you selected the last time you used the Multi-Build dialog.

For information about the actions that Mobile Designer performs to build your project, see "Build Process Overview" on page 127.

Note: Before you build your project, be sure you have completed the required setup. See "Before You Can Build a Mobile Application Project" on page 138.

To build a project for the last target

1. In Software AG Designer, open the project you want to build.
2. Click the **Project Explorer** view, expand the project, and drag the build.xml file to the **Ant** view.

   If the Ant view is not open, for instruction, see "Displaying the Ant View" on page 65.
3. In the **Ant** view, double-click **+Multi-Build-Last**.

   Mobile Designer does not display a dialog. Mobile Designer rebuilds the project for the last target device that was built, using the settings from the last build, including any setting you might have made in a custom JPanel.

Building a Project from the Command Line

You can build your project from the command line using the +Target-Build Ant target. When using the +Target-Build Ant target, you can build for only a single device/language combination at a time. To execute the Ant target, from the command line navigate into the project's folder.

For information about the actions that Mobile Designer performs to build your project, see "Build Process Overview" on page 127.

Note: Before you build your project, be sure you have completed the required setup. See "Before You Can Build a Mobile Application Project" on page 138.

Syntax

```
version=x.y.z [retain={true | false}] handset=device langgroup=language
platform=platform target=executable_type +Target-Build
```

Options

Option	Description
`version=x.y.z`	Required. Specifies the version number for the application your are building.
`[retain={true | false}]`	Optional. Specifies whether you want Mobile Designer to retain the cross-compiled code it generated from your original Java code, along with any platform-specific project it might have generated The default is `true`.
`handset=device`	Required. Specifies the device for which you want Mobile Designer to build the application.
`langgroup=language`	Required. Specifies the language(s) for which you want Mobile Designer to build the application.
`platform=platform`	Required. Specifies the platform for which you want Mobile Designer to build the application.
`target=executable_type`	Required. Specifies the type of executable you are building. For example, you might specify `release` or `debug`.

Example

To build version 2.0.3 of your project for the IOS_Apple_iPhone5 device, which runs on the iOS platform, for the language group EFIGS to create an executable version for the Apple App Store.

```
ant -Dversion=2.0.3 -Dretain=true -Dhandset=IOS_Apple_iPhone5 -Dlanggroup=EFIGS
-Dplatform=IOS -Dtarget=appstore +Target-Build
```

Because `retain` is set to `true`, Mobile Designer retains the cross-compiled code it generated from your original Java code, along with any project (for example, Xcode project for iOS) it might have generated.

Using Native Tools to Create the Final Binary

If you want to use platform-specific, native tools to create the final binary and package your application, you can use the Mobile Designer+Multi-Build or +Target-Build Ant target to build your project, but have Mobile Designer skip compiling your project into binary

and packaging your application for the target device. You can move the cross-compiled code and platform-specific project that Mobile Designer creates to the platform-specific tool to create the final binary.

For information about the actions that Mobile Designer performs to build your project, including the actions that Mobile Designer skips when you perform the following procedure, see "Build Process Overview" on page 127.

Note: Before you build your project, be sure you have completed the required setup. See "Before You Can Build a Mobile Application Project" on page 138.

To use native tools to create the final binary for your project

1. Set the project.handset.skip.compilation property to `true` to indicate that you want Mobile Designer to skip compiling your project into binary and packaging your application for the target device. For instructions for how to set a property, see "Setting Project Properties" on page 119.
2. Start the build using one of the following Ant targets:
   - +Multi-Build Ant target. For instructions, see "Building a Project for Multiple Target Devices" on page 139.

     Be sure you select the **Retain output build files** in the Multi Build dialog.
   - +Target-Build Ant target. For instructions, see "Building a Project from the Command Line" on page 140.

     Be sure you set the `retain` option to `true`.
3. After the build is complete, locate the cross-compiled code and platform-specific project.

   When Mobile Designer builds your project, it adds a Builds folder to your project folder. Within the Builds folder, Mobile Designer creates a _temp_ folder for each device. The cross-compiled code and the platform-specific project reside within the _temp_ folder:

   project /Builds/*x.y.z* /*device* /_temp_

   In the folder location above:

   - *project* is the name of your project.
   - *x.y,z* is the version number you specified for the build.
   - *device* is the name of a device for which Mobile Designer built your application.

   You can find the cross-compiled code and the platform-specific project in the _temp_/_*language* _ folder. If you created a patch file for the application, you can find the patched versions in the _temp_/_*language* _edit_ folder. For example, for an Android build, you can find the cross-compiled code in the _temp_/_java_ folder, and if a patch was applied, the patched version of the code is in the _temp_/_java_edit_ folder.

4. Copy the files you need to create the final binary to the platform-specific, native tool. Then use the native tool to create the final binary.

Generating Javadocs for a Project

Use the Generate-Javadoc Ant target to generate javadocs for a project. The javadocs that the Generate-Javadoc Ant target generates are primarily javadocs for the customer's project codebase. The javadocs also include Mobile Designer run-time methods and constants.

When you use the Generate-Javadoc Ant target, Mobile Designer adds a _temp_ folder to your main project folder that contains the generated files.

To generate javadocs for a project

1. In Software AG Designer, in the Project Explorer view, locate the project for which you want to generate javadocs.
2. Expand the project and drag its build.xml file to the Ant view.

   If the Ant view is not open, see "Displaying the Ant View" on page 65.
3. In the Ant view, double-click **Generate-Javadoc**.

   Mobile Designer displays the Activate Handset dialog.
4. In the Activate Handset dialog, select the device for which you want generate javadocs and the language group. Then click **Activate Handset**.

Mobile Designer places the results of the Generate-Javadoc Ant target in your project's _temp_ folder.

15 Customizing the Build Process

About Customizing the Build Process

Mobile Designer provides the following features that you can use to customize the standard build process:

- **Custom JPanels.** You can create a custom JPanel that Mobile Designer includes in the Multi Build dialog. The purpose of the custom JPanel is to allow you to change the values of project properties or parameters when you initiate a build without having to manually edit your project's _defaults_.xml file, target device files, or resource handler code. For more information, see "Setting Properties at Build Time Using a Custom JPanel" on page 146.

  Note: If you create a custom JPanel, Mobile Designer also includes the JPanel in the Activate Handset dialog. For more information, see "Activating Devices" on page 173.

- **Hook points.** You can create custom Ant scripts that you want Mobile Designer to run during the build process. Mobile Designer defines various points, called *hook points*, in the build process where it can run an Ant script that you provide. For more information, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

- **Patch files.** You can create patch files that Mobile Designer applies to the cross-compiled code. Use patch files to correct issues that might prevent the cross-compiled code from compiling successfully. Additionally, some times the code might cross-compile successfully, but the resulting code might not execute as expected. You might be able to correct the issue with a patch file. For more information, see "Creating Patch Files to Apply to the Cross-Compiled Code" on page 152.

Setting Properties at Build Time Using a Custom JPanel

You might want to be able to change the values of project properties or parameters when you initiate a build or activate a device without having to manually edit your project's _defaults_.xml file, target device files, or resource handler code. You can achieve this by creating a custom panel (JPanel) for your project. For more information about project properties and parameters, see "Setting Properties and Parameters for a Mobile Application Project" on page 115.

Mobile Designer includes your custom JPanel into its Multi Build and Activate Handset dialogs. When you use the +Multi-Build Ant target to start a build, the Multi Build dialog includes your custom JPanel on the right. Similarly, when you use the ++Activate-Handset Ant target to activate a device, the Activate Handset dialog includes your custom JPanel. If you create a custom JPanel for a project, Mobile Designer incorporates the JPanel into *both* dialogs.

To use a custom JPanel, first you must code the JPanel. For more information, see "Coding Your Custom JPanel" on page 147.

After you code the JPanel, you set properties in your project's build.xml file to indicate that you are using a JPanel and provide information about the location of the code. The JPanel properties must go in the build.xml file rather than the _defaults_.xml file because Mobile Designer must use the properties at the beginning of the build process *before* it reads the project properties in the _default_.xml and target device files. For information about the JPanel properties you need to set, see "Setting JPanel Properties" on page 147.

Coding Your Custom JPanel

If you started your project by cloning a project that included a custom JPanel, you can update the JPanel from that project. Alternatively, you create the JPanel from the beginning on your own.

JPanel Requirements

- You must name the class that contains your JPanel code MyJPanel.java.
- Your MyJPanel.java class must extend com.softwareag.mobile.core.ProjectJPanelHandler and any associated libraries that it needs for compilation.

Methods that Mobile Designer Provides for the JPanel

The com.softwareag.mobile.core.ProjectJPanelHandler class, which your project's MyJPanel.java class must extend, contains methods you can use in your JPanel. For more information about the ProjectJPanelHandler, see the *webMethods Mobile Designer Java API Reference*.

Sample JPanel Code

You can also find an example that demonstrates how to use the ProjectJPanelHandler in the Device Profiler sample project provided with Mobile Designer.

Setting JPanel Properties

This section lists the JPanel properties to specify when you want to use a custom JPanel for your project. Include these properties in your project's build.xml file.

project.uses.defined.jpanel

Specifies whether you want to include a custom JPanel in the Multi Build and Activate Handset dialogs.

Platforms All

Value
- `true` if you are using a custom JPanel.

- `false` if you are *not* using a custom JPanel.

Default `false`

Use the following format to add the Ant property to your project's build.xml file:

```
<property name="project.uses.defined.jpanel" value="value"/>
```

For example to indicate that you are using a custom JPanel:

```
<property name="project.uses.defined.jpanel" value="true"/>
```

project.jpanel.src.path

Specifies the Ant path to the source code for the custom JPanel. The source code for your custom JPanel *must* be called MyJPanel.java.

Platforms All

Value Path to the MyJPanel.java file.

Default null

Use the following format to add the Ant path to your project's build.xml file:

```
<path id="project.jpanel.src.path">
        <pathelement path="path"/>
</path>
```

For example:

```
<path id="project.jpanel.src.path">
        <pathelement path="${basedir}/jpanel"/>
</path>
```

project.jpanel.additional.libs.path

Specifies the Ant path to additional libraries that your custom JPanel requires.

Platforms All

Value Path to the libraries

Default null

Use the following format to add the Ant path to your project's build.xml file:

```
<path id="project.jpanel.additional.libs.path">
        <pathelement path="path"/>
</path>
```

For example:

```
<path id="project.jpanel.additional.libs.path">
```

```
            <pathelement path="${basedir}/jpanel/libs"/>
</path>
```

Creating Custom Ant Scripts to Run at Predefined Hook Points

To customize the standard build process that Mobile Designer performs, you can create custom Ant scripts that Mobile Designer runs at various predefined points, called *hook points*. For example, to automatically upload a build to an FTP server once the build is created, you can use the PostBuild Hook Point, which Mobile Designer runs after it completes a build.

Mobile Designer provides several predefined hook points in the build process where you can inject a custom Ant script to perform custom actions. For example, you can have Mobile Designer run a custom Ant script before it runs the resource handler, or you can have Mobile Designer run a custom Ant script after it runs the resource handler. For more information about the hook points you can use and when each occur in the build process, see "Hook Point Reference" on page 150 and "Ant Target Summary" on page 277.

To run a custom Ant script at a hook point:

- You must add the associated hook point property to your project's _defaults_.xml file. You use the property to provide the name of the custom Ant script that you want Mobile Designer to run. For information about the properties, see "Hook Point Properties" on page 255.
- You must create the custom Ant script. You can include the Ant script in your _defaults_.xml file directly after the property declaration.

  Ant properties, parameters, and paths set in your custom Ant scripts do not flow through into the remaining build process. Your custom Ant scripts are separate Ant target calls that branch off outside the normal build process flow.

Caution: Although invoking custom Ant scripts at the predefined hook points adds flexibility to the build process, be aware that Mobile Designer does *not* perform safety checking on custom Ant scripts before executing them. You should thoroughly test your custom Ant script's functionality before integrating it into Mobile Designer.

Note: Use of hook points is optional. You can use none, one, or multiple hook points. Mobile Designer only attempts to run a custom Ant script at a hook point if your project's _defaults_.xml file contains a hook point property that provides the name of a custom Ant script.

For an example that illustrates how you use hook points, see the NativeUI PDF Demo sample project.

Hook Point Reference

The following sections describe the hook points that you can use for a project. Each section lists:

- When the hook point occurs in the build process
- Property that you specify in your project's _defaults_.xml file to provide the name of the custom Ant script you want Mobile Designer to run at the hook point
- Names of the Mobile Designer Ant targets that use the hook point

To see a diagram that shows where the hook points are in the build process, see "Ant Target Summary" on page 277.

PreBuildResources Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script before it runs your project's resource handler.

Property	project.hookpoint.target.prebuildresources
Ant targets	+Run-Reshandler +Multi-Build +Multi-Build-Last +Target-Build ++Activate-Handset + +Re-Activate-Handset ++Run-Phoney-With-Re-Activation

PostBuildResources Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script after it runs your project's resource handler.

Property	project.hookpoint.target.postbuildresources
Ant targets	+Run-Reshandler +Multi-Build +Multi-Build-Last +Target-Build ++Activate-Handset + +Re-Activate-Handset ++Run-Phoney-With-Re-Activation

PostCrossCompiler Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script after it converts your project's source to the build's target language.

Note: Mobile Designer only calls this hook point for cross-compiled builds.

Property	project.hookpoint.target.postcrosscompiler
Ant targets	+Multi-Build +Multi-Build-Last +Target-Build

PreMakefileGeneration Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script before it generates the makefile and associated Microsoft Visual Studio or Apple Xcode project for a build.

Note: Mobile Designer only calls this hook point for cross-compiled builds.

Property	project.hookpoint.target.premakefilegeneration
Ant targets	+Multi-Build +Multi-Build-Last +Target-Build

PrePatch Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script before it applies the patches to your code.

Note: Mobile Designer only calls this hook point for cross-compiled builds.

Property	project.hookpoint.target.prepatch
Ant targets	+Multi-Build +Multi-Build-Last +Target-Build

PreCompilation Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script before it performs platform-specific compilation for each build.

Property	project.hookpoint.target.precompilation
Ant targets	+Multi-Build +Multi-Build-Last +Target-Build ++Activate-Handset ++Re-Activate-Handset ++Run-Phoney-With-Re-Activation

PostCompilation Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script after it performs platform-specific compilation for each build.

Property	project.hookpoint.target.postcompilation
Ant targets	+Multi-Build +Multi-Build-Last +Target-Build

PostPackaging Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script after it generates the platform-specific build bundle. After executing this hook point, Mobile Designer performs code signing, if any, that is appropriate to the platform being built.

Note: Mobile Designer only calls this hook point for cross-compiled builds.

Property	project.hookpoint.target.postpackaging
Ant targets	+Multi-Build +Multi-Build-Last +Target-Build

PostBuild Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script after it packages and signs the build.

Property	project.hookpoint.target.postbuild
Ant targets	+Multi-Build +Multi-Build-Last +Target-Build

PostMultiBuild Hook Point

When you use this hook point, Mobile Designer invokes your custom Ant script after it creates all the builds you selected in the Multi Build dialog.

Property	project.hookpoint.target.postmultibuild
Ant targets	+Multi-Build +Multi-Build-Last +Target-Build

Creating Patch Files to Apply to the Cross-Compiled Code

A *patch file* is a standard difference (diff) file that you can create. Use patch files when you need to make changes to the code that the Mobile Designer cross compiler creates. For example:

- You might need to correct issues that prevent the code from compiling.
- You might need to correct issues when the code cross compiles correctly, but the resulting code does not execute as expected.

  For example, in your source Java code, you use multiple post increments, such as `i++` in one line of code:

  ```
  int rgb = (data[i++] << 16) + (data[i++] << 8) + (data[i++]);
  ```

This line of code can cross-compile to C++ fine, but cannot be guaranteed to execute as Java does. In some C++ compilers this type of code might result in "undefined behavior." In this case, even though the code will convert to C++ and then compile successfully, you might find the run-time execution is incorrect. For these types of situations, it is usually a best practice to write a more compliant variant in your Java code. Continuing this example, you might change the original Java code to something like the following:

```
int rgb = (data[i] << 16) + (data[i + 1] << 8) + (data[i + 2]);
```

You create the patch file one time, and Mobile Designer automatically applies the fix to all future builds, even if you continue to make changes to your application's original Java code. For instructions about how to create a patch and where to store the patch file, see "Creating a Patch" on page 153.

When Mobile Designer builds your project, it adds a Builds folder to your project folder. Within the Builds folder, there are folders for each device for which your application was built.

project /Builds/*x.y.z* /*device*

In the folder location above:

- *project* is the name of your project.
- *x.y,z* is the version number you specified for the build.
- *device* is the name of a device for which Mobile Designer built your application.

The *device* folder contains a _temp_/_*language* _ folder that contains the version of the cross-compiled code *without* a patch applied. If a patch file exists, Mobile Designer patches the code and stores the resulting patched version of your project in the _temp_/_*language* _edit_ folder. For example, for Android, Mobile Designer generates a _java_edit_ folder along side the _java_ folder, or for Windows Phone Mobile Designer generates a _csharp_edit_ folder along side the _csharp_ folder.

Creating a Patch

Mobile Designer provides patch and diff tools with sample scripts so that you can create patch files. For an example of how to use the patch and diff tools, see the Function Demo sample project.

To create a patch file

1. Build your project using one of the following Ant targets:
   - +Multi-Build Ant target. For instructions, see "Building a Project for Multiple Target Devices" on page 139.

     Be sure you select the **Retain output build files** in the Multi Build dialog.
   - +Target-Build Ant target. For instructions, see "Building a Project from the Command Line" on page 140.

Be sure you set the `retain` option to `true`.

2. Locate your project's Build folder, which Mobile Designer creates during the build process.
3. In the Build folder, locate the cross-compiled version of your code.

   Specifically, look for your code in the following folder:

   project /Builds/*x.y.z* /*device* /_temp_/_*language* _edit_

   In the folder location above:

   - *project* is the name of your project.
   - *x.y,z* is the version number you specified for the build.
   - *device* is the name of a device for which Mobile Designer built your application.
   - *language* is the language into which your application code was cross compiled, for example, java.
4. Review the cross-compiled code and make any minor code adjustments needed to correct the issues you are encountering so that your code compiles as expected on the target platform.
5. Run the patch_maker script that Mobile Designer provides in the following project folder:

   project /Builds/*x.y.z* /*device* /_temp_

   When Mobile Designer builds your project, it places versions of the cross-compiled code in both of the following project folders:

   - _*language* _ folder
   - _*language* _edit_ folder

   The version of the cross-compiled code in the _*language* _edit_ folder includes any patch that you might have already applied.

   When you run the patch_maker script, Mobile Designer generates a diff file with the differences between the two folders.
6. Save the diff file that the patch_maker script created in the project's main folder using the names specified in the following table.

Platform	Patch Name
Android	`android_java.diff`
iOS	`ios_cpp.diff`
Windows Phone	`winphone_csharp.diff`

Platform	Patch Name
Windows RT	`win8rt_csharp.diff`
Windows 8	`win8str_csharp.diff`

Note: If you want to give your diff file an alternate name, for example, if two targets for the same platform require different diff files, you can use a custom name for a diff file. You still save the diff file to the project's main folder. However, if you name your diff file differently from the names specified in the table above, you must add the `patch.name` property to the target device file in the project's targets folder. Set the value of the `patch.name` property to the name of the diff file to use for the device.

V Installing and Testing Mobile Applications

16 Using Phoney for Debugging Your Mobile Application

About Using Phoney to Debug Mobile Applications

The Mobile Designer utility, *Phoney*, is a phone simulator that is not platform-specific. You can test your application by running it in Phoney.

If you use the Mobile Designer NativeUI library to create the user interface for your mobile application, the user interface is rendered using Phoney skins that Mobile Designer provides. Mobile Designer provides some platform-specific Phoney skins that attempt to mimic the look-and-feel of a platform. For platforms that Mobile Designer does *not* provide a platform-specific Phoney skin, Mobile Designer provides a general graphical skin for the user interface. For more information about Phoney skins, see *webMethods Mobile Designer Native User Interface Reference*.

Use the ++Run-Phoney, ++Run-Phoney-With-Activation, or ++Run-Phoney-With-Re-Activation Ant targets to run a mobile application in the Phoney simulator. For information about the actions Mobile Designer takes when you run Phoney, see "Phoney Ant Target Summary" on page 160 and "Steps Performed for Phoney Ant Targets" on page 162.

You can run Phoney from Software AG Designer. For instructions, see "Running Phoney from Software AG Designer " on page 163. Alternatively, you can run Phoney from the command line. For more information, see "Running Phoney from Software AG Designer " on page 163.

If a mobile application accesses services via the Internet and you want to use SSL to secure the communications between the Phoney and the service, install certificates on Phoney. For instructions, see "Installing Certificates on Phoney" on page 169.

You can use the Phoney Metrics panel to get an estimation of an application's memory and thread usage to help determine whether you might encounter issues with memory and thread usage when running the application on physical devices. For more information, "Using Phoney to Monitor an Application's Memory and Thread Usage" on page 169.

Note: You can also test and debug by installing a mobile application on to a physical device or a platform-specific emulator or simulator. See the providers' documentation for the latest instructions for how to install applications and use emulators or simulators that they provide. This documentation includes limited instructions for some platforms. For more information, see "Installing Applications on Devices" on page 179.

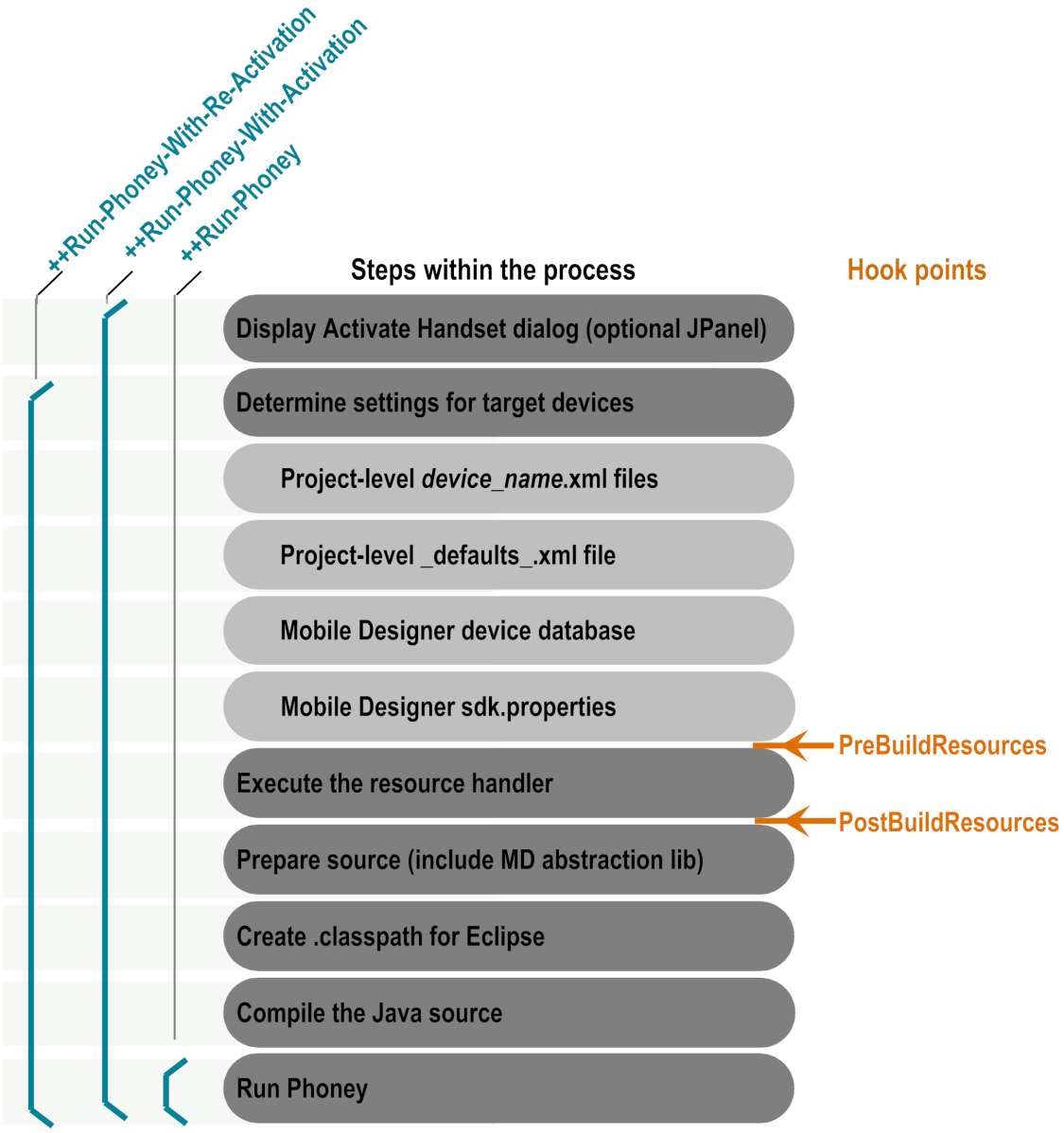
Phoney Ant Target Summary

The following diagram provides summary information about the process Mobile Designer performs when you use Phoney Ant targets. It shows:

- The names of the Ant targets you can use to run Phoney.

- The steps Mobile Designer performs. For details about each step, see "Steps Performed for Phoney Ant Targets" on page 162.
- The hook points in the process where Mobile Designer can run custom Ant scripts that you provide. For more information about hook points, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

Note: For instructions for how to run Phoney, see "Running Phoney from Software AG Designer " on page 163 and "Running Phoney from the Command Line" on page 164.



Steps Performed for Phoney Ant Targets

This section describes the steps that Mobile Designer performs when you run the ++Run-Phoney, ++Run-Phoney-With-Activation, and ++Run-Phoney-With-Re-Activation Ant targets to run a mobile application in the Phoney simulator.

Note: For instructions for how to run Phoney, see "Running Phoney from Software AG Designer " on page 163 and "Running Phoney from the Command Line" on page 164.

++Run-Phoney Ant Target

When you use the ++Run-Phoney Ant target, Mobile Designer performs only the Run Phoney step.

The Run Phoney step starts Phoney for the last activated device. You can specify Phoney startup options using the phoney.base.params property.

++Run-Phoney-With-Activation and ++Run-Phoney-With-Re-Activation Ant Targets

Use the ++Run-Phoney-With-Activation Ant target to activate a device, then run Phoney. Use the ++Run-Phoney-With-Re-Activation Ant target to reactivate the last device, then run Phoney.

When you use the ++Run-Phoney-With-Activation or ++Run-Phoney-With-Re-Activation Ant target, Mobile Designer performs the steps in the following table. The table also indicates hook points where Mobile Designer can run custom Ant scripts that you provide. For more information about hook points, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

The first four steps and hook points are for reactivating the last device. For details about these steps, see "Steps Performed to Activate Handsets" on page 175.

Step	Description
1	**Display the Activate Handset dialog (optional JPanel)** Mobile Designer displays the Activate Handset dialog, and also a custom JPanel for the build, if you defined one. For more information, see "Setting Properties at Build Time Using a Custom JPanel" on page 146. **Note:** Mobile Designer only displays the Activate Handset dialog when you run the ++Run-Phoney-With-Activation Ant target.
2	Determine setting for target devices

Step	Description
hook point	Optional. Run a custom Ant script for the PreBuildResources hook point
3	Execute the resource handler
hook point	Optional. Run a custom Ant script for the PostBuildResources hook point
4	Prepare source (include Mobile Designer abstraction lib
5	Create .classpath for Eclipse
6	Compiles the Java source
7	Run Phoney Mobile Designer starts Phoney for the last activated device. You can specify Phoney startup options using the phoney.base.params property.

Running Phoney from Software AG Designer

To run Phoney, you should have already activated a device for your project because the Phoney Ant targets run the your application for the last activated device. For information about activating a device, see "Activating Devices" on page 173.

Note: To run Phoney from the Mobile Development perspective, refer to the Mobile Development documentation available in Softwage AG Designer via **Help > Help Contents > Software AG Designer Guides > webMethods Mobile Development Help**.

To run Phoney from Software AG Designer

1. In Software AG Designer, click the **Project Explorer** view, expand the Mobile Designer project, and drag the build.xml file to the Ant view.

   If the Ant view is not open, for instructions, see "Displaying the Ant View" on page 65.

2. To create a Run configuration, use the ++Activate-Handset Ant target to activate a handset for your project. For instructions, see "Activating a Device" on page 177.

   Note: As part of activating a device, Mobile Designer creates a Run configuration, which you can use to run or debug the application using Software AG Designer and Phoney.

3. In Software AG Designer run Phoney using either a Run Configuration, to simply execute Phoney, or Debug Configuration, if you want to use debug capabilities, for example, setting breakpoints.
   - To use **Run Configurations**:
     i. Select **Run > Run Configurations**.
     ii. In **Run Configurations**, expand **Java Application**, select your mobile application project and click **Run**.
   - To use **Debug Configurations**:
     i. Select **Run > Debug Configurations**.
     ii. In **Run Configurations**, expand **Java Application**, select your mobile application project and click **Run**.

Running Phoney from the Command Line

You can run Phoney from the command line using any of the Phoney Ant targets. To execute the Ant target, from the command line navigate into the project's folder.

Syntax

`[options] ant_target`

For *`ant_target`*, specify one of the Phoney Ant targets, for example, `++Run-Phoney`. The options you can use are listed in the table below.

Phoney Startup Options

Note: If you want to run Phoney from Software AG Designer, you can specify the startup options in the phoney.base.params property.

Option and Description
{--accelerate_images | -ai} Enables image acceleration, speeding up 32-bit images.
{--device | -fd} *`device_name`* Forces Phoney to emulate the dimensions and keys of the specified device. For *`device_name`* specify the name you selected from the **Choose your handset** list in the Add Handset dialog when you added the device to the project. For example, for the Microsoft Surface RT device, use `WN8_Microsoft_SurfaceRT` for *`device_name`*.
{--dimensions | -d} *`width`***x***`height`*

Option and Description
Specifies the canvas dimensions. The default is 176x220 pixels. **Note:** If you also specify the `{--device | -fd}` option, the dimensions you specify with the `{--dimensions | -d}` option override the dimension specified with the `{--device | -fd}` option. Other settings specified with the `{--device | -fd}` option still apply.

{--fc_pim_roots | -rt} *`simulated_drives`*

Specifies one or more simulated drives that Phoney uses when executing FileConnection classes.

To specify a drive use the format:

`fake_root_name``,`*`path`*

For example, to simulate the D:\ drive in C:\fake\d, use the following

```
D:\,C:\fake\d
```

Use a semicolon-separated list to specify multiple simulated drives. For example, to simulate the D:\ drive in C:\fake\d and the E:\drive in C:\fake\e, use the following

```
D:\,C:\fake\d;E:\,C:\fake\e
```

The default is the following:

```
C:\,%FCDIR%/c/
```

For the default, `%FCDIR%` maps to the following directory:

Mobile Designer_directory\Tools\Phoney\FC_PIM_fake_roots

{--fixed_frame_rate | -ff}

Specifies you want to Phoney to create AVI files using fixed frame rate. If you do not specify this option, Phoney uses variable rate.

{--frame_rate | -fr} *`rate`*

Specifies that you want Phoney to record everything that is displayed in the Phoney window in an AVI file. Specify *`rate`* to indicate the frame rate (frames per second) to use to record the movie. The default frame rate value is 25fps.

Using this option allows you to record a movie of your application, for example, to use as a demonstration video.

{--invert | -i}

Inverts the number pad so that it behaves like a phone keypad.

{--jad_param | -jp} *`parameters`*

Option and Description
Specifies Java Application Descriptor (JAD) parameters. If a JAD file is present, the parameters you specify override the parameters in the JAD file. You can specify the `{--jad_param | -jp}` startup option multiple times.

{--location_position | -lp} *`latitude;longitude;altitude`*

Specifies the default position returned by the Location API. The default is "51:30:26;-0:7:39;24" (London).

{--max_scale | -ms}

Sets the scale of the current screen to the maximum that is allowable given the current screen resolution.

{--metrics_mem_limit_kb | -mm} *`megabytes`*

Specifies the maximum number of kilobytes to use for memory before issuing warnings. Phoney signals a warning if an application's memory usage is approaching or exceeding configured limit. The default is 51200 KB (50 MB). For more information, see "Using Phoney to Monitor an Application's Memory and Thread Usage" on page 169.

Note: It is recommended that you do not change the default. If you need to change it, do so for specific devices rather than the entire project.

{--metrics_thread_limit | -mt} *`number`*

Specifies the maximum number of threads to use before issuing warnings. Phoney signals a warning if an application's thread usage is approaching or exceeding configured limit. The default is 15 threads. For more information, see "Using Phoney to Monitor an Application's Memory and Thread Usage" on page 169.

Note: It is recommended that you do not change the default. If you need to change it, do so for specific devices rather than the entire project.

{--no_fake_fc_pim_roots | -fk}

Disables the behavior of the `{--fc_pim_roots | -rt}` startup option.

Caution: When you use `{--no_fake_fc_pim_roots | -fk}`, Phoney lists your computer's root devices directly. As a result, your mobile application has direct access to the root drives, for example, C:\.

{--no_file_connection | -fc}

Option and Description
Disables the FileConnection portion of FC-PIM. Use this option when you want to simulate devices that do not support the FileConnection API.
{--no_menus | -nm} Disables the menu bar.
{--no_pim | -np} Disables the reading of the Phoney Contacts file.
{--no_settings | -ns} Specifies that you want Phoney to only use the command-line options for user settings and ignore settings in user settings files. By default, Phoney retains settings from previous sessions in a properties file so they can be used again. Using this property ensures that Phoney does not use previously saved settings.
{--open_dialog | -od} Shows the file open dialog so that you can have Phoney load a JAD or JAR file at startup.
{--open_screenshot | -os} Indicates that you want Phoney to automatically open any screen shot you take using F1. The screen shot is open using the system-defined application for viewing PNG images.
{--pim_root | -pr} Specifies where to locate the Phoney Contacts file. The default is the following: `%PIMDIR%` For the default, `%PIMDIR%` maps to the following directory: *Mobile Designer_directory*\Tools\Phoney\FC_PIM_Contacts
{--properties | -p} *`filename`* Specifies you want Phoney to use device settings in the specified properties file. **Important:** Be sure to specify this setting so that Phoney automatically represents the activated device.

Option and Description
{--redirect_output | -ro} Redirects all stdout and stderr text to an internal buffer that you can then view from the **Phoney > Console** menu item.
{--repaint_sleep | -rs} *milliseconds* Specifies the number of milliseconds the MIDP repaint calls sleeps, allowing time for the paint thread to service the queue.
{--rms_files | -r} Specifies you want to store RMS data in files. Phoney uses RMS files for the RecordStore class read/write processes. If you do not use file-based RMS, the save/load operations do not persist between Phoney sessions.
{--scale | -s} *number* Specifies the window scale multiplier. The default is 1. **Note:** While running Phoney, you can change the screen scale by pressing the '+' or '-' key to increase or decrease the scale.
{--single_keypress | -sk} Disables multiple simultaneous key presses. By default, multiple simultaneous key presses are enabled.
{--slow_rms | -sr} *milliseconds* Specifies you want to emulate a record store that performs slowly. Specify the number of milliseconds you want the RecordStore class to use to perform an add or set operation. **Note:** This is mostly useful for legacy J2ME devices that sometimes had issues writing to the RecordStore, causing a lag when trying to save too frequently or not allowing enough time for the write operations to complete. By simulating the time that the device takes, you can determine how the application will perform on devices that exhibit this issue.
{--softkeys | -k} Forces a system soft-key area for LCDUI soft keys.
{--sound | -so}

Option and Description
Enables sound playback using the JDK 1.5 Java Sound API.
{--verbose | -v} Specifies you want verbose output.
{--verbose_mmapi | -vm} Specifies you want verbose debug information from the MMAPI code.

Installing Certificates on Phoney

Phoney makes use of the J2SE keychain. As a result, you can use X.509 v1, v2, and v3 certificates and PKCS #7-formatted certificate chains consisting of certificates of that type.

Note: The J2SE keychain details vary based on the VM you have installed. For more information about the Oracle Java 7 keychain, see http://docs.oracle.com/javase/7/docs/technotes/tools/solaris/keytool.html.

To install custom SSL certificates on Phoney

- Use keytool to install the certificate.

  The keytool is located in %JAVA_HOME%/bin.

  Example:

```
keytool -importcert -file %KEY_FILE_FOLDER%\%KEY_FILE%
-alias %CERTIFICATE_ALIAS% -keystore "%JAVA_HOME%\jre\lib\security\cacerts
```

Using Phoney to Monitor an Application's Memory and Thread Usage

You can use Phoney to run applications on a simulated, generic device. When using Phoney to run an application, you can view memory and thread usage information for the simulated device in the Metrics panel. Use the metrics to determine:

- Estimation of how much memory and threads an application uses.
- When an application's memory and thread use is at its highest.

The metrics can help determine whether you might encounter issues with memory and thread usage when running the application on physical devices.

Additionally, Phoney displays warnings if an application's memory and/or thread use exceeds configured thresholds.

Note: When running the application on a physical device, the metrics might differ somewhat from the values in the Metrics panel. For example, when running the application on a physical device, the memory usage might be a little less because the format of the application resources, such as, data, images, and sound, might be in a format better suited for the physical device. Native Mobile Designer libraries used on physical devices might also differ slightly in their usage of threads.

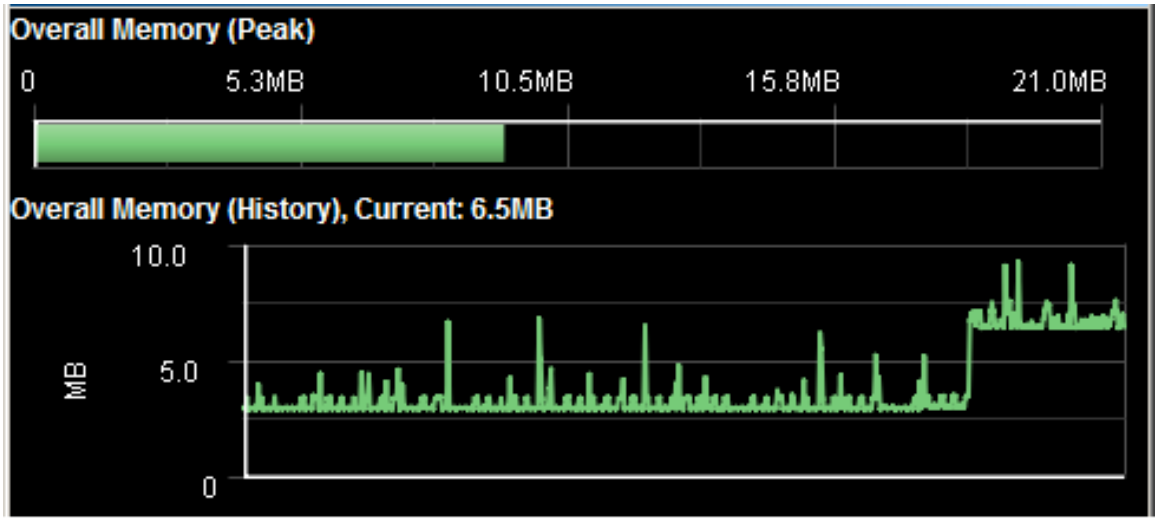
Also note that other applications accessing sound and video hardware can cause the JVM in which Phoney is running to create spurious threads. For example, web browsers displaying video, Flash, or other rich content might cause the JVM in which Phoney is running to create spurious threads. You can safely ignore these additional threads.

Metrics Panel

You can open the Metrics panel from Phoney by selecting **Phoney > App Metrics**, or by pressing CTRL+M on the keyboard. The Metrics panel displays the following information:

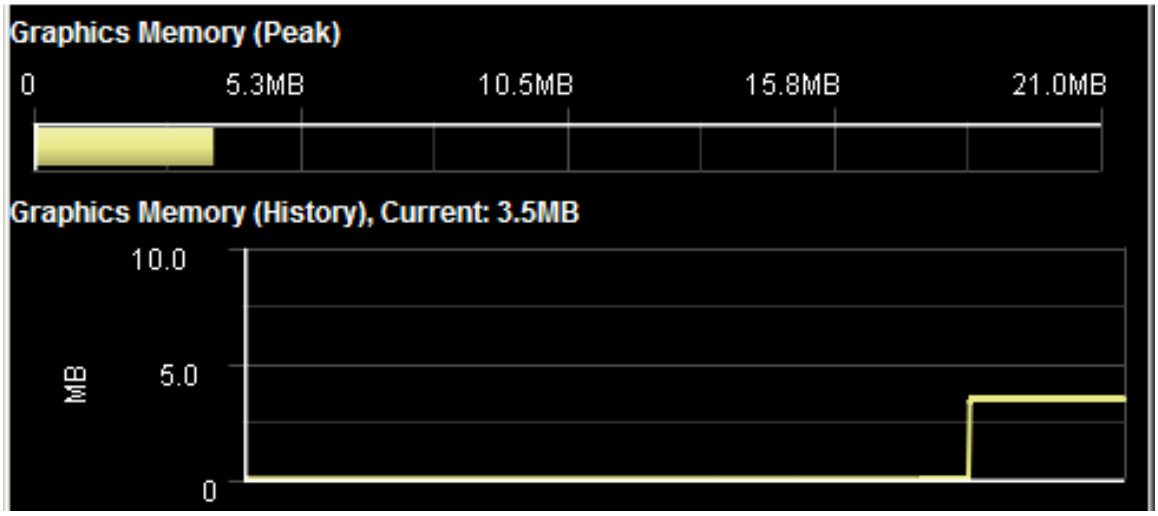
- **Overall memory usage of an application**

  The Peak bar shows the highest recorded use of memory, as a proportion of the current JVM's total size. The History graph shows the last 60 seconds of memory usage.



- **Memory usage for Image classes within an application**

  Similar to the overall memory metrics, these metrics display a Peak bar that shows the highest recorded use of memory for Image classes within the application. The History graph shows the last 60 seconds of memory usage.



Because both the overall memory usage and the usage for Image classes use the same scale, you can determine the proportion of the overall memory that the application uses for Image classes.

- **Thread usage**

  For thread usage, the Metrics panel lists the number of threads that the application has created. The History graph shows the last 60 seconds of thread usage.

Note: Even if the Metrics panel is closed, Phoney continues to monitor memory and thread usage and displays warnings if memory and/or threshold use exceeds configured thresholds.

Setting Warning Thresholds and How Warnings are Displayed

You can configure limits for when you want Phoney to signal a warning if an application's memory and/or thread usage is approaching or exceeding configured limits. To configure the limits, you specify options when you start Phoney. If you do not specify the startup options, Phoney uses default values. You can specify the startup options:

- On the command line when you start Phoney.

- When starting Phoney via Software AG Designer. Set the options in one of the following locations:
  - On the **Arguments** tab inside the Run Configurations window.
  - Using the phoney.base.params property in the project's targets/_defaults_.xml file.

When an application goes over the configured limits, Phoney displays a blinking icon on the bottom-left of the screen. If the Metrics Panel is open, the background behind the graphs also blinks red. The warnings blink for approximately 60 seconds after the application's memory and/or thread usage reduces to the configured limits.

Note: Even if the Metrics panel is closed, Phoney monitors memory and thread usage and displays warning icons.

The following table lists the startup option you use to configure the limits, the default values for the limits, and provides information about when Phoney displays warning.

Memory limit	**Startup option**	`{--metrics_mem_limit_kb | -mm}`
	Default value	50 MB
	Warning	Blinking **RAM** warning icon - **When blinking slowly**, the application's usage is over 90% of the memory limit. - **When blinking rapidly**, the application's usage is over the memory limit.
Thread limit	**Startup option**	`{--metrics_thread_limit | -mt}`
	Default value	15 threads
	Warning	Blinking **CPU** warning icon - **When blinking slowly**, the application's usage is over 80% of the thread limit. - **When blinking rapidly**, the application's usage is over the thread limit.

The following example shows startup options to set the memory usage limit to 20 MB and the thread usage limit to 5 threads.

```
--metrics_thread_limit 5 --metrics_mem_limit_kb 20480
```

Alternatively, you can use the short form of the options:

```
--mt 5 --mm 20480
```

17 Activating Devices

About Activating Devices

When you set up your mobile application project, you add devices to the project. Each device you add is a device your mobile application supports. When testing how your application runs on one of the devices, you need the run-time settings to be set for a specific device. To get the correct settings for a device, you can activate it. For example, if you want to test your application using Phoney, you might first activate the device for which you want to test, then run Phoney to test your application.

When you activate a device, Mobile Designer

- Hot-swaps in the required Mobile Designer run-time classes. For more information, see " Mobile Designer-Provided Run-Time Classes" on page 72 and "Run-Time Classes Properties" on page 266.
- Sets up the project compile path appropriately by creating the .classpath file that provides information about how to compile the Java code.
- Runs the project's resource handler so that the com.softwareag.mobile.runtime.Parameters class contains settings for the device you activate
- Compiles the code.

For more information about the steps Mobile Designer takes when you activate a device, see "Activate Devices Ant Summary" on page 174 and "Steps Performed to Activate Handsets" on page 175.

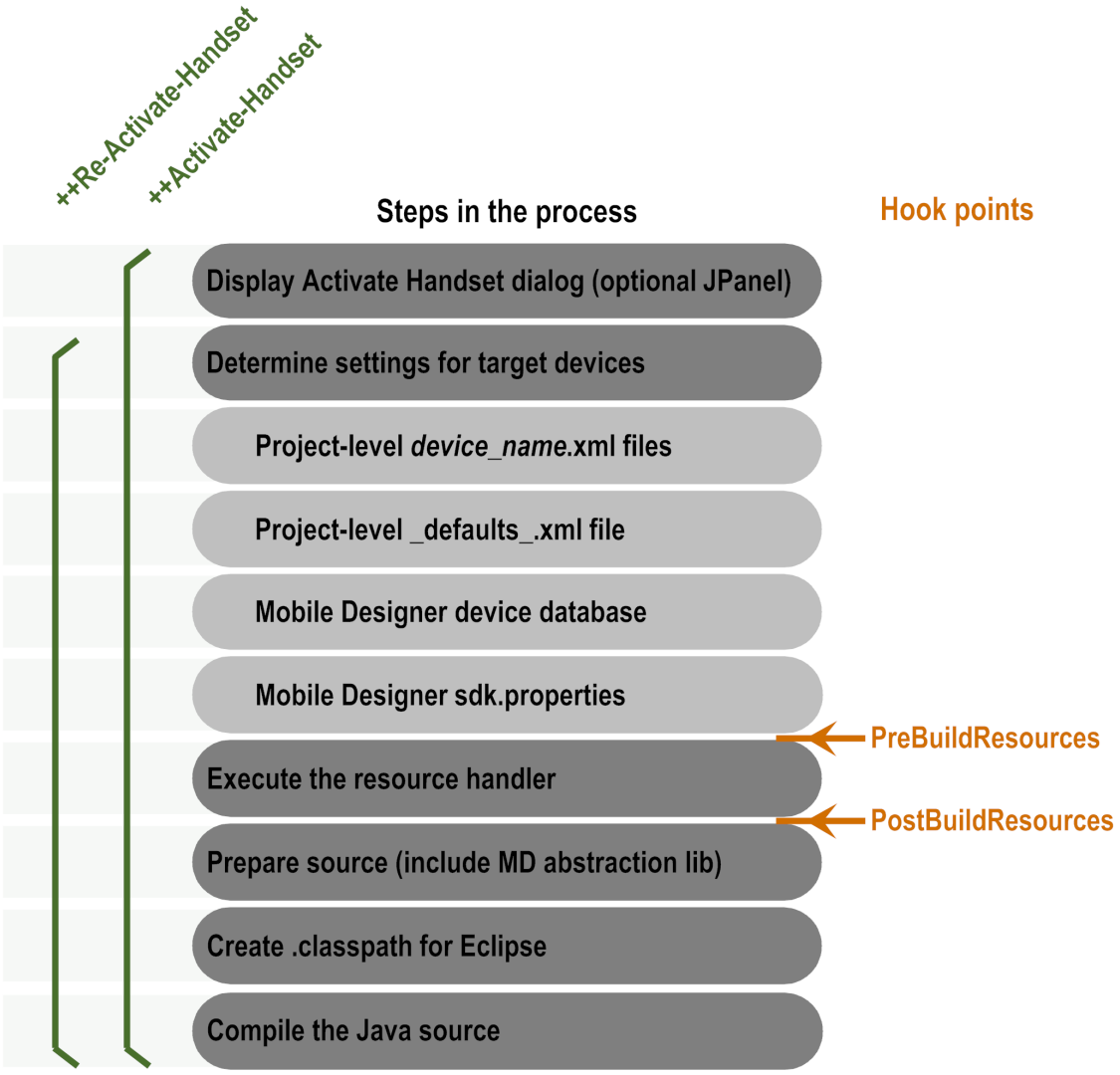
To activate a device, you use the ++Activate-Handset or ++Re-Activate-Handset Ant targets. For instructions, see "Activating a Device" on page 177.

Activate Devices Ant Summary

The following diagram provides summary information about the process Mobile Designer performs when activating a device. It shows:

- The names of the Ant targets you can use to activate devices.
- The steps Mobile Designer performs. For details about each step, see "Steps Performed to Activate Handsets" on page 175.
- The hook points in the process where Mobile Designer can run custom Ant scripts that you provide. For more information about hook points, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

Note: For instructions for how to activate a device, see "Activating a Device" on page 177.



Steps Performed to Activate Handsets

This section describes the steps that Mobile Designer performs when you run the ++Activate-Handset or ++Re-Activate-Handset Ant targets to activate a device.

The table also indicates hook points where Mobile Designer can run custom Ant scripts that you provide. For more information about hook points, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

Note: For instructions for how to activate a device, see "Activating a Device" on page 177.

1 Display the Activate Handset dialog (optional JPanel)

Mobile Designer displays the Activate Handset dialog, and also a custom JPanel for the build, if you defined one.

For more information, see "Setting Properties at Build Time Using a Custom JPanel" on page 146.

Note: Mobile Designer only displays the Activate Handset dialog when you run the ++Activate-Handset Ant target.

2 Determine settings for target devices

Mobile Designer determines the settings for the device you are activating. It retrieves the settings from the following sources in the order listed.

- Project-level *device_name* .xml files
- Project-level _defaults_.xml file
- Mobile Designer device database
- Mobile Designer sdk.properties file

Mobile Designer uses the first setting it encounters. For example, if Mobile Designer encounters a setting in the project-level target *device_name* .xml file and then again in the project-level _defaults_.xml file, Mobile Designer uses the setting from the target *device_name* .xml file.

For more information about:

- Project-level device files, see "Where You Set Properties" on page 116 and "Setting Project Properties" on page 119
- Mobile Designer device database, see "Devices that a Mobile Application Supports" on page 88
- Mobile Designer sdk.properties file, see "Mobile Designer Configuration Properties (sdk.properties)" on page 19

hook point PreBuildResources

If you have created an Ant script to run at the PreBuildResources hook point, Mobile Designer runs the Ant script.

3 Execute the resource handler

Mobile Designer runs the resource handler that you created for the project. When running the resource handler, Mobile Designer records all the resources required for your application. Mobile Designer also creates the com.softwareag.mobile.runtime.Parameters class. For more information about creating the resource handler, see "Defining Resources for a Mobile Application Project" on page 101. For more information about the Parameters class, see "Application and Parameter Classes" on page 72.

At this point in the build process, Mobile Designer uses the following project properties:

- project.reshandler.src.path for the location of the project's resource handler script and any associated classes
- project.java.reshandler.name for the name of the resource handler class you created for your project.
- project.resource.dir.root for the location of the top-level folder that contains the resources (audio files, image files, etc.) for your project
- project.reshandler.additional.libs.path for the location of additional libraries that your project's resource handler requires

For more information about these properties, see "Resource Handler Properties" on page 261.

hook point **PostBuildResources**

If you have created an Ant script to run at the PostBuildResources hook point, Mobile Designer runs the Ant script.

4 **Prepare source (include Mobile Designer abstraction lib)**

Mobile Designer prepares the source by including the relevant run-time classes. For more information, see " Mobile Designer-Provided Run-Time Classes" on page 72 and "Run-Time Classes Properties" on page 266.

5 **Create .classpath for Eclipse**

Mobile Designer creates a classpath file in the project's root folder to reference the stubs that the device's run-time code and that the resource handler requires. The Mobile Designer project is refreshed so that the classpath changes are propagated through to the codebase.

6 **Compile the Java source**

Activating a Device

To activate a device you use one of the following Ant targets:

- ++Activate-Handset to specify the device and the language combination that you want to activate.
- ++Re-Activate-Handset to activate the last device that was activated.

Note: When you activate a device, Mobile Designer also creates a debug or run configuration. You can find the created .launch file in the project's _temp_ folder.

To activate a device

1. In Software AG Designer, click the **Project Explorer** view, expand the Mobile Designer project, and drag the build.xml file to the Ant view.

   If the Ant view is not open, for instruction, see "Displaying the Ant View" on page 65.

2. In the Ant view, double-click the Ant target you want to use:

   - Double-click **++Activate-Handset** if you want to specify a device/language combination.

     Mobile Designer displays an Activate Handset dialog. Continue with the next step to complete the procedure.

   - Double-click **++Re-Activate-Handset** to re-activate the last device.

     Mobile Designer does not display a dialog. It re-activates the last device and does not require you to perform further actions.

3. In Activate Handset dialog, select the device you want to activate from **Choose your handset** list.

4. Select the language group for which you want to test from **Choose your language group** list.

5. Click **Activate Handset**.

18 Installing Applications on Devices

About Installing Applications on Devices

After you use webMethods Mobile Designer to build an application, you can install the final binary on the target devices. This documentation describes how to install applications on the following commonly-used platforms:

- Android
- iOS
- Windows Phone

The procedures in this documentation do *not* cover all possible setups and scenarios. Refer to the device provider's web pages for further details.

Installing Applications on Android Devices

To install Android applications, you can use the following methods to install applications on emulated or physical devices:

- Install an Android application package (APK) file using the Android Debug Bridge (ADB)
- Install an Eclipse project build using the Android Development Tools (ADT) Eclipse plug-in

Note: If you need to install a certificate, this documentation provides information about installing certificates on Android 4.0 and later devices. See "Installing Certificates on Android 4.0 and Later Physical Devices" on page 187. For additional information, see the device provider's web pages.

Installing an APK File to an Emulated or Physical Device Using the Android Debug Bridge

To upload an APK file to an Android device, use the ADB command-line tool. The ADB tool is located in the platform-tools directory of your Android SDK install. See the Android developer website for more information about the ADB tool.

Tip: To make accessing the Android tools from the command line easier, add the platform-tools and tools folder of your SDK install to your default path.

Note: You can use the following procedure for both virtual Android devices and physical Android devices.

To install an APK file using the ADB tool

1. Ensure that your Android device is fully booted.

2. If you are using a physical device, connect the device to your computer and ensure that you have all required drivers installed.

   Note: For physical Android devices, you might need to enable debugging. To do so, launch the Settings application, and select **Applications > Development > USB Debugging**.

3. Open a command prompt and execute one of the following, where *apkFilePath* is the path to the APK file and *filename* is the name of your APK file.

   a. **For Windows:**

   ```
   cd apkFilePath
   C:\android-sdk-windows\platform-tools\adb install filename.apk
   ```

   b. **For Macintosh**:

   ```
   cd apkFilePath
   /android-sdk-macosx/platform-tools/adb install filename.apk
   ```

   The ADB tool interacts with the device and installs your build.

4. Launch your application from the device and test it.

   Note: From the command line, use `adb logcat` to monitor output from your device. This can be useful when debugging your application.

Installing an Application to an Emulated or Physical Device Using the ADT Eclipse Plug-In

The following procedure describes how to install an application to a device using Eclipse with the Android Development Tools (ADT) Eclipse plug-in.

To install an application using the ADT Eclipse plug-in

1. When building your project using the Mobile Designer +Multi-Build Ant task, be sure to select the **Retain output build files** check box.

   When the build completes successfully, Mobile Designer retains a folder named _temp_ in the same folder as your build. The _temp_/_java_edit_ folder is a self-contained Eclipse project that you can import.

2. Import the project in the _temp_/_java_edit_ folder into Eclipse as an existing project:

   a. In the Package Explorer window, right-click the project and select **Import**.

   b. For the type of project to import, select **General > Existing Projects into Workspace**, then click **Next**.

   c. When selecting the root of the project you want to import, select **Select root directory** and browse to and select the _temp_/_java_edit_ folder of the project you want to import.

   d. Click **Finish**.

3. Do one of the following based on whether you are installing to a emulated or physical device.
   - **To install to an emulated device**, ensure the you have configured at least one Android Virtual Device for the target device's API. For more information, see "Setting Up an Android Virtual Device (Emulator)" on page 39.
   - **To install to a physical device**, ensure the device is connected to the machine and turned on.
4. From Eclipse, select **Run > Run**.
5. Select that you want to run the project as an **Android Application**.

   Eclipse launches the appropriate Virtual Device, if it is not already started, and installs the application.

Installing Applications on iOS Devices

To install iOS applications, you can use the following methods:

- **To install to a simulator**, you can install using Apple Xcode.
- **To install to a physical device**, you can:
  - Install using the Apple Xcode IDE
  - Install an Ad-Hoc Build Using iTunes

Note: If you need to install a certificate, this documentation provides information about installing certificates on iOS devices. See "Installing Certificates on iOS Physical Devices" on page 188. For additional information, see the device provider's web pages.

Installing to a Simulated or Physical Device Using the Apple Xcode IDE

This method is useful when installing during the development process and when you want to install to a simulated device.

To install to a simulated or physical device using Xcode

1. When building your project using the Mobile Designer +Multi-Build Ant task, be sure to select the **Retain output build files** check box.

   When the build completes successfully, Mobile Designer retains a folder named _temp_ in the same folder as your build. The temp/_cpp_edit_ folder is an Xcode project (the .xcodeproj file) that you can use to access your cross-compiled source.
2. Double-click the .xcodeproj file.

Xcode starts.

3. Select the device to which you want to install the application:
   - **To install to a simulated device,** on the screen set the scheme to the simulated device to which you want to install.
   - **To install to a physical device**, attach the iOS device to your computer.
4. Click **Run**.

   Xcode compiles your application, signs it with the development certificate, and installs it on the selected simulated or physical device.

Installing an Ad-Hoc Build to a Physical Device Using iTunes

Use the following procedure to install an ad-hoc build to a physical iOS device.

To install an Ad-Hoc build using iTunes

1. Gather the build files you need for installation and make them available on the machine running iTunes.
   a. Locate the application bundle in the project Builds folder.

      After running the Mobile Designer+Multi-Build Ant task to create an ad-hoc device build, Mobile Designer places the application bundle in this location.

   b. Compress the application bundle on the Macintosh where you created the build.

      If you want to distribute the build, for example, via an email message or upload it to a server, you must compress the files before moving them. Distributing an uncompressed application bundle can cause issues, for example, missing symbolic links and/or permissions set for the build.

      Important: It is recommended that you use a program such as 7Zip, WinZip, or WinRar for the decompression. Using the built-in **Extract All** function that comes with Windows can damage the build when it is extracted.

   c. Locate the ad-hoc mobile provision file used when signing the application.

      Tip: This is the file specified in the `ios.adhocprov` property in Mobile Designer sdk.properties file.

   d. Copy the compressed application bundle and the ad-hoc mobile provision file (.mobileprovision file) to the computer you want to use for the installation.

      Place the files in an easy-to-find location, for example, the desktop.
2. Optionally, if the application already exists on the physical device, delete the application if you want to perform a clean installation.
3. Connect the physical iOS device to your computer.

4. Start iTunes.
5. Optionally, install the ad-hoc mobile provision file by dragging the .mobileprovision file into **Library > Applications** in iTunes.

   Important: You must install the .mobileprovision file if:

   - This is the first time installing this application on the device.
   - The .mobileprovision file has changed since the last time you installed the application.

6. Install the application bundle.
   a. Extract the files from the compressed application bundle.
      - On Windows, the application is in a folder with the name *application_name* .app
      - On Macintosh, the application is in a single file with a name that matches the application name.
   b. Drag the application bundle (that is, the Windows folder or Macintosh file) into **Library > Applications** in iTunes.
7. Select **Library > Applications** to verify that the new application is present.
8. Under **Devices**, select the iOS device to which you want to install the application.
9. Select the **Applications** tab.
10. Select **Sync Applications** and that the application you want to install is selected.
11. Select the **Summary** tab.

    iTunes installs the application.

Installing a Windows Phone 8 Application to an Emulated or Physical Device

The following procedure describes how to install a Windows Phone 8 application.

Note: A virtual emulator is installed with the Windows Phone SDK 8.

Note: If you need to install a certificate, this documentation provides information about installing certificates on the Windows Phone emulator. See "Installing Certificates on Windows Phone Emulator" on page 188. For additional information, see the device provider's web pages.

To install a Windows Phone 8 application

1. When building your project using the Mobile Designer +Multi-Build Ant task, be sure to select the **Retain output build files** check box.

   When the build completes successfully, Mobile Designer retains a folder named _temp_ in the same folder as your build. The folder _temp_/_csharp_edit_ contains a .csproj file, which is a Visual Studio project file.

2. Open the Visual Studio project file (.csproj file) in Visual Studio Express 2012 for Windows Phone using one of the following methods:

   - From Windows Explorer, right-click the Visual Studio project file and select **Open with > Microsoft Visual Studio 2012 Express for Windows Phone**.
   - From Microsoft Visual Studio 2012 Express for Windows Phone, do the following:

     i. Select **File > Open Project Dialog**.

     ii. Navigate to the correct _temp_/_csharp_edit_ folder to select the root directory.

     iii. Click **Open**.

3. Do one of the following based on whether you are installing to a emulated or physical device.

   - **To install to an emulated device**, select **Debug > Start Debugging**.

     The Windows Phone Emulator launches and your application starts.

   - **To install to a physical device**, ensure the device is connected to the machine and turned on.

     i. Unlock the device for development. For more information, see http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff769508%28v=vs.105%29.aspx.

     ii. Connect the device via USB. Ensure the device is active. Ensure the that the screensaver is not active.

     iii. In Visual Studio 2012 Express for Windows Phone, select your target, either a device or an emulator, then click anywhere else to build, deploy the application to the device, and start the execution of the application.

Installing a Windows RT/Windows 8 Application to an Emulated or Physical Device

The following procedure describes how to install a Windows RT/Windows 8 application. Only Windows Store/Metro applications are supported.

Note: A virtual emulator is installed with Visual Studio 2012 Express for Windows Phone.

To install a Windows RT/Windows 8 application

1. When building your project using the Mobile Designer +Multi-Build Ant task, be sure to select the **Retain output build files** check box.

   Note: Windows RT and Windows 8 require the use of Visual Studio to sign and deploy the application.

   When the build completes successfully, Mobile Designer retains a folder named _temp_ in the same folder as your build. The folder _temp_/_csharp_edit_ contains a .csproj file, which is a Visual Studio project file.

2. Open the Visual Studio project file (.csproj file) in Visual Studio Express 2012 for Windows 8 using one of the following methods:

   - From Windows Explorer, right-click the Visual Studio project file and select **Open with > Microsoft Visual Studio 2012 Express for Windows 8**.
   - From Microsoft Visual Studio 2012 Express for Windows 8, do the following:

     i. Select **File > Open Project Dialog**.

     ii. Navigate to the correct _temp_/_csharp_edit_ folder to select the root directory.

     iii. Click **Open**.

3. Do one of the following based on whether you are installing to a emulated or physical device.

   - **To install to an emulated device**:

     i. In Visual Studio Express 2012 for Windows 8, switch to **Simulator**.

     ii. Select **Debug > Start Debugging**

   - **To install to a physical device**, ensure the device is connected to the machine and turned on.

     i. Download and install the Remote Debugger on the device itself. For more information, see http://msdn.microsoft.com/en-gb/library/vstudio/bt727f1t.aspx.

     ii. Start the Remote Debugger.

     If the user account and/or password on the remote device does not match the login for your desktop, in Remote Debugger select **Tools > Options** and perform configuration to allow any user.

     iii. Ensure that you know the IP address of the device.

iv. In Visual Studio Express 2012 for Windows 8, switch to **Remote Machine**. You will be prompted for the IP address.

If you have configured Remote Debugger for no authentication, ensure the **Authentication** dropdown is set to **none**.

v. Select **OK**.

vi. Click the **Remote Machine** and Visual Studio builds and deploys the application.

Installing Custom SSL Certificates on Devices

If an application accesses services via the Internet and you want to use SSL to secure the communications between the device and the service, install certificates on the device. This documentation describes the supported certificate types and how to install certificates for the following platforms:

- Android 4.0 and later devices
- iOS physical devices
- Windows Phone Emulator

The procedures in this documentation do *not* cover all possible setups and scenario. Refer to the device provider's web pages for further details.

Installing Certificates on Android 4.0 and Later Physical Devices

Android 4.0 (Ice Cream Sandwich) and later supports DER-encoded X.509 certificates saved in files with a .crt or .cer file extension.

Note: If you do not have a valid certificate installed, you will see the error javax.net.ssl.SSLHandshakeException: java.security.cert.CertPathValidatorException: Trust anchor for certification path not found.

To install custom SSL certificates on Android 4.0 and later physical devices

1. If your certificate file has a .der or other extension, change it to .crt or .cer.
2. Use the Android Debug Bridge (ADB) tool to send a copy of your certificate to the SD card on the device. To do so, with the ADB running, execute the following command, where *certificate* is the name of your certificate:

```
adb push certificate.crt /sdcard/
```

3. On the Android device, select **Settings > Location & Security Settings > Set up screen lock** to ensure you have a password or screen lock for the device.

4. On the device, select **Settings > Personal > Security > Credential Storage > Install from SD Card** to install the custom certificate.

Installing Certificates on iOS Physical Devices

Apple iOS supports DER or Base64-encoded X.509 certificates saved in files with a .crt, .cer, .der or .pem file extension.

Note: If you do not have a valid certificate installed, you might see an error on the console MD: ERROR: Can't reach url "…" and a java.io.IOException.

Important: You can only use the following procedure to install custom certificates on a physical iOS device. The procedure will not work to install certificates to the keychain that the iOS simulator uses.

To install custom SSL certificates on iOS physical devices

1. Send the certificate to your device by doing one of the following:
   - Send an email message with the certificate as an attachment. On the device, open the email attachment.
   - Put the certificate on an accessible server. On the device, download the certificate from Safari.

   When the device receives the certificate, it opens an Install Profile dialog.
2. In the Install Profile dialog, click **Install**.
3. Double-check the root certificate to ensure it is the one you want to install.
4. Click **Install** in the warning dialog.
5. Enter your passcode if you have one set up on your device.

On an iOS device, you can view and remove installed certificates selecting **Settings > General > Profiles**.

Installing Certificates on Windows Phone Emulator

Windows Phone supports DER-encoded X.509 certificates with a .cer file extension.

Note: Without a valid certificate, various errors and exceptions are thrown, including System.Net.WebException in libmidp20_slv_debug.dll at javax.microedition.io.HttpConnectionImplNative.getResponse(HttpWebRequest request) and System.NotSupportedException in mscorlib.dll at System.Threading.Interlocked.Increment(Int64& location).

Important: When using the Windows Phone Emulator, you must install the custom certificate each time the Windows Phone Emulator is started.

To install custom SSL certificates on Windows Phone Emulator

1. Put the certificate on an accessible server. On the emulator, download the certificate from Internet Explorer.
2. Confirm that you want to install the certificate and click **Install**.

VI Distributing Mobile Applications

19 Distributing Applications Using webMethods Mobile Administrator

Using Mobile Administrator to Manage and Distribute Mobile Applications

Mobile Administrator allows you to manage and distribute your mobile applications. Mobile Administrator provides an app store where users can browse the app catalog to select applications to install. Mobile Administrator can send push notifications to users when updates are available for their installed applications.

Mobile Administrator also allows you to set up build nodes that you can use to remotely build a mobile application. You can set up build nodes that run Mobile Designer.

Mobile Designer provides a Mobile Administrator plug-in. Using the Mobile Administrator plug-in, you can:

- **Upload applications** you create using Mobile Designer to the Mobile Administrator app store.

  In this situation, you create and build your mobile application locally on your own machine. Then, use the plug-in to upload the final binaries to Mobile Administrator.

- **Remotely build applications** using a Mobile Administrator build node and make the resulting applications available from the Mobile Administrator app store.

  In this situation, you can use Mobile Designer to code mobile applications locally on your own machine. Then, using the Mobile Administrator plug-in, you transmit your source code to Mobile Administrator. Mobile Administrator, in turn, remotely builds your application on a build node running Mobile Designer. The resulting final binaries are made available in the Mobile Administrator app store.

When your Mobile Designer applications are in the Mobile Administrator app store, users can install Mobile Designer mobile applications from the app store and can receive push notifications when you make updates available. Using Mobile Administrator application-level permissions, you can make test builds available to only members of a development team, while giving users who want a production-ready application access to the latest stable version of the application.

To use the Mobile Administrator plug-in, be sure to perform the setup described in "Requirements for Using the Mobile Administrator Plug-in for a Project" on page 195. After you have completed the setup, see the following for instructions for how to use the plug-in for uploading applications and remotely building applications:

- "Uploading Final Binaries to Mobile Administrator " on page 205
- "Remotely Building a Project" on page 206
- "Monitoring Jobs Used to Remotely Build Projects" on page 209

Requirements for Using the Mobile Administrator Plug-in for a Project

Using the Mobile Administrator plug-in requires setup in both Mobile Administrator and in Mobile Designer.

Mobile Administrator Setup

For more detailed information on the steps below, see the *webMethods Mobile Administrator User's Guide*.

- Create a Mobile Administrator application that you will associate with your Mobile Designer project. You need one Mobile Administrator application for each Mobile Designer project.
- Use an existing Mobile Administrator user account or define a new one, and assign the user account to the Mobile Administrator application.

  Mobile Designer uses this user account when accessing Mobile Administrator to upload or remotely build the project.
- Ensure the Mobile Administrator user account, at a minimum, has the following permissions for the Mobile Administrator application:
  - View and Download Stable Versions
  - Manage Build Jobs

  Mobile Designer requires these permissions so that it can request information, such as the last version of the application or a list of Mobile Administrator build configurations, and so that it can start build jobs.
- Ensure the Mobile Administrator user account has the global Manage Site permission.

  You can set this permission in Mobile Administrator on the **Details** tab for a user.

  Mobile Designer requires this permission so that it can retrieve a list of certificates for the project.
- Determine whether you want Mobile Designer to use an access token or basic authentication when using the Mobile Administrator user account to access Mobile Administrator.

  Note: It is recommended that you use an access token for authentication.
- Generate an access token for the Mobile Administrator user account if you want to use an access token for authentication.
- Set up Mobile Designer build nodes if you want to remotely build your project.

Mobile Designer Setup

- Create your Mobile Designer project or use an existing project.
- Activate the Mobile Administrator plug-in. For instructions, see "Activating the Mobile Administrator Plug-in for a Mobile Designer Project" on page 196.
- Define Mobile Administrator plug-in properties for your Mobile Designer project. For instructions, see "Setting Mobile Administrator Plug-in Project Properties" on page 196.

Activating the Mobile Administrator Plug-in for a Mobile Designer Project

To activate the Mobile Administrator plug-in for a project

1. In Software AG Designer, open the Mobile Designer project.
2. Click the **Project Explorer** view, expand the project, and double-click the build.xml file to open the build.xml file in the default editor.
3. In the build.xml file, insert an `<import>` that imports the plug-in. Place the `<import>` after the last line inside the `<project>` tag, as shown below:

```
.
.
.
  <import
file="${env.MOBILE_DESIGNER}/plugins/MobileAdministrator/v1.0.0/targets.xml"
  />
</project>
```

4. Save the changes to the build.xml file.

Setting Mobile Administrator Plug-in Project Properties

Define the Mobile Administrator plug-in properties for a project in the project's ma.properties file. The ma.properties file resides in the root folder for a project along with the build.xml file.

To define Mobile Administrator plug-in properties

1. If your project does not have a ma.properties file, perform the following to add one:
   a. Locate the ma.properties.template file in the following location:

      Mobile Designer_directory/plugins/MobileAdministrator/v1.0.0

   b. Copy the ma.properties.template file to your project's root folder, renaming it to ma.properties.

2. Set properties using the following format:

   ```
   property=value
   ```

   The properties you can add to the ma.properties files are described in "Project Properties for the Mobile Administrator Plug-In" on page 197.

3. After adding the properties, save the file.

Project Properties for the Mobile Administrator Plug-In

The following sections describe the properties that you can add to the ma.properties file to configure the Mobile Administrator plug-in for a project. Each section lists:

- Description of the property
- Platforms for which the Mobile Administrator plug-in supports the property
- Ant tasks that use the property
- Description of the value you need to specify for the property

These properties do *not* have default values. You must specify a value for each property you want to use. For information about how to set properties, see "Setting Mobile Administrator Plug-in Project Properties" on page 196.

mobile.admin.server

Specifies a URL of the Mobile Administrator instance.

Platforms	Android iOS Windows Phone 8.x Windows RT Windows 8
Ant tasks	Upload-Binaries Upload-Binaries-Last Remote-Multi-Build
Value	URL of the Mobile Administrator instance Example value: `http://localhost:8080`

mobile.admin.project.slug

Specifies the `Slug` property that identifies the Mobile Administrator project that is associated with the Mobile Designer project.

Platforms	Android iOS

Windows Phone 8.x
Windows RT
Windows 8

Ant tasks: Upload-Binaries Upload-Binaries-Last Remote-Multi-Build

Value: Value of the `Slug` property that identifies the Mobile Administrator project

Note: You can find the `Slug` property in Mobile Administrator on the **Detail** tab for the project.

ma.ios.bundle.id

Specifies the `bundleID` you want Mobile Designer to use for the final binary. The bundle ID is a unique identifier of a final binary.

Important: The bundle ID for the Mobile Designer project must match the bundle ID for the Mobile Administrator project.

Note: If you do not specify a value for this property, Mobile Designer uses the bundle ID that it generates by using the value of the `ios.bundle` property from the sdk.properties file, the application name, device name, and language. Because Mobile Designer includes the device name and language in the bundle ID, two devices for the same platform have different bundle IDs. However, Mobile Administrator supports only one bundle ID per platform. Use this property to override the bundle ID Mobile Designer generates for the final binary.

Platforms: iOS

Ant tasks: Upload-Binaries Upload-Binaries-Last

Value: Bundle ID

mobile.admin.use.authentication

Specifies the type of authentication you want Mobile Designer to use when it accesses Mobile Administrator during the process of uploading or remotely building the project.

Platforms: Android
iOS
Windows Phone 8.x
Windows RT
Windows 8

Ant tasks	Upload-Binaries Upload-Binaries-Last Remote-Multi-Build
Value	■ `access_token` if you want to use an access token for authentication. When you specify `access_token`, use the mobile.admin.auth.access_token property to provide the access token. Software AG recommends that you use an access token for authentication. ■ `basic` if you want to use basic authentication. When you specify `basic`, use the following properties to provide a user name, password, and/or to indicate whether you want Mobile Designer to prompt for a password: ■ mobile.admin.basic.auth.user ■ mobile.admin.basic.auth.pass ■ mobile.admin.basic.auth.prompt_for_pass

mobile.admin.auth.access_token

Specifies the access token that you want Mobile Designer to use for authentication when it accesses Mobile Administrator during the process of uploading or remotely building the project.

Note: This property is only applicable when you set the mobile.admin.use.authentication property to `access_token`.

Platforms	Android iOS Windows Phone 8.x Windows RT Windows 8
Ant tasks	Upload-Binaries Upload-Binaries-Last Remote-Multi-Build
Value	Access token that Mobile Administrator generated for a user account.

mobile.admin.basic.auth.prompt_for_pass

Specifies whether you want Mobile Designer to prompt for a password to use for authentication when Mobile Designer accesses Mobile Administrator during the process of uploading or remotely building the project. The password corresponds to the user name you specify in the mobile.admin.basic.auth.user property.

Note: This property is only applicable when you set the mobile.admin.use.authentication property to `basic`.

Platforms
Android
iOS
Windows Phone 8.x
Windows RT
Windows 8

Ant tasks
Upload-Binaries Upload-Binaries-Last Remote-Multi-Build

Value

- `true` if you want Mobile Designer to prompt for a password.

  If you specify `true` and you also specify the password in the mobile.admin.basic.auth.pass property, Mobile Designer uses the password you supply at the prompt and ignores the value of the mobile.admin.basic.auth.pass property.
- `false` if you do not want Mobile Designer to prompt for a password.

  If you specify `false`, be sure to specify the password in the mobile.admin.basic.auth.pass property.

mobile.admin.basic.auth.user

Specifies the user name of the user account that you want Mobile Designer to use when it accesses Mobile Administrator during the process of uploading or remotely building the project.

Note: This property is only applicable when you set the mobile.admin.use.authentication property to `basic`.

Platforms
Android
iOS
Windows Phone 8.x
Windows RT
Windows 8

Ant tasks
Upload-Binaries Upload-Binaries-Last Remote-Multi-Build

Value
User name of a Mobile Administrator user account

Be sure to specify a user account that has permissions to act against the Mobile Administrator project that corresponds to the Mobile Designer project. For a description of the

required permissions, see "Requirements for Using the Mobile Administrator Plug-in for a Project" on page 195.

mobile.admin.basic.auth.pass

Specifies the password associated with the user account that you provide with the mobile.admin.basic.auth.user property.

Note: This property is only applicable when you set the mobile.admin.use.authentication property to `basic`.

Platforms	Android iOS Windows Phone 8.x Windows RT Windows 8
Ant tasks	Upload-Binaries Upload-Binaries-Last Remote-Multi-Build
Value	Password of the Mobile Administrator user account. **Caution:** If you use basic authentication, the password is saved in plain text in the ma.properties file. To avoid this, Software AG recommends that you use an access token for authentication.

mobile.admin.md.latest.version.file

Specifies the Mobile Designer`project.temp.last.version.properties.file` project property that the +Multi-Build Ant task uses for caching the last version of the final binary.

Platforms	Android iOS Windows Phone 8.x Windows RT Windows 8
Ant tasks	Upload-Binaries Upload-Binaries-Last Remote-Multi-Build
Value	Set the value to the `project.temp.last.version.properties.file` project property

mobile.admin.upload.version

Specifies how Mobile Designer determines the version number to use for the project build.

Mobile Designer accesses Mobile Administrator to get the project's last version number. Use this property to indicate whether you want Mobile Designer to use that last version for the build or to calculate a new version number by incrementing the last build version number by one.

Platforms
Android
iOS
Windows Phone 8.x
Windows RT
Windows 8

Ant tasks
Upload-Binaries
Remote-Multi-Build

Value

- `INCREMENT` if you want Mobile Designer to increment the last version number by one.

  Note: If the last version number does not end with a number, for example, "1.0.0.0a", Mobile Designer cannot increment the version number. Instead, Mobile Designer prompts you to provide the version number to use.

- `LATEST` if you want Mobile Designer to use the last version number.

mobile.admin.upload.version.stable

Specifies whether you want to mark the project final binaries that are uploaded to Mobile Administrator as stable.

Platforms
Android
iOS
Windows Phone 8.x
Windows RT
Windows 8

Ant tasks
Upload-Binaries Upload-Binaries-Last Remote-Multi-Build

Value

- `true` if you want to mark the uploaded final binaries as stable.
- `false` if the uploaded final binaries are not stable.

mobile.admin.upload.open_browser_on_success

Specifies whether you want a webpage containing a link to the application displayed if the build is successful. The webpage is displayed in the default browser.

Platforms
Android
iOS
Windows Phone 8.x
Windows RT
Windows 8

Ant tasks
Upload-Binaries Upload-Binaries-Last

Value
- `true` if you want to display a web page containing a link that you can use to download the mobile application if the build is successful.
- `false` if you do *not* want to display a web page containing a link for the application if the build is successful.

mobile.admin.reuse.existing.build_configs

Specifies whether you want Mobile Designer to use the existing build configurations defined in the Mobile Administrator project that is associated with your Mobile Designer project.

Note: You create build configurations from Mobile Administrator on the **Build** tab for the project.

Platforms
Android
iOS
Windows Phone 8.x
Windows RT
Windows 8

Ant tasks
Remote-Multi-Build

Value
- `true` if you want Mobile Designer to use existing build configurations to build targets
- `false` if you want Mobile Designer to create new build configuration in the corresponding Mobile Administrator project based on information you supply on the Remote Multi-Build dialogs.

  Note: After the build is finished, Mobile Designer does not delete the newly created build configurations because deleting

build configurations triggers the deletion of all underlying build jobs in Mobile Administrator.

mobile.admin.data.transmission.zip.includes

Specifies a list of project files and folders that you want Mobile Designer to include in a remote project build.

This property identifies the files and folders that reside within the project folder that you want Mobile Designer to transmit to Mobile Administrator to perform a remote multi-build of the project. When you use this property, Mobile Designer transmits only the files and folders that you specifically *include* using this property.

Note: Specify either the `mobile.admin.data.transmission.zip.includes` property or the mobile.admin.data.transmission.zip.excludes. If you specify both properties, Mobile Designer uses only the `mobile.admin.data.transmission.zip.includes` property.

Platforms	Android iOS Windows Phone 8.x Windows RT Windows 8
Ant tasks	Remote-Multi-Build
Value	Semicolon-separated list of project files and folders to include. Specify the files and folders that are required to remotely build the project.

mobile.admin.data.transmission.zip.excludes

Specifies a list of project files and folders that you want Mobile Designer to exclude from the project build.

This project identifies the files and folders that Mobile Designer transmits to Mobile Administrator to perform a remote multi-build of the project. Mobile Designer transmits all the files and folders from the project folder *except* the ones you specifically exclude using this property.

Note: Specify either the mobile.admin.data.transmission.zip.includes property or the `mobile.admin.data.transmission.zip.excludes`. If you specify both properties, Mobile Designer uses only the mobile.admin.data.transmission.zip.includes property.

Platforms	Android

	iOS Windows Phone 8.x Windows RT Windows 8
Ant tasks	Remote-Multi-Build
Value	Semicolon-separated list of project files and folders to exclude. Specify only files and folders that are *not* required to remotely build the project.

mobile.admin.data.transmission.zip.ignores

Specifies a list of file or folder name prefixes. When Mobile Designer transmits files to Mobile Administrator to perform a remote multi-build, Mobile Designer ignores, that is, does not transmit, files and folders in the project folder that begin with the prefixes you specify. This property applies to all files and folders, including subfolders and their files recursively.

Platforms	Android iOS Windows Phone 8.x Windows RT Windows 8
Ant tasks	Remote-Multi-Build
Value	Semicolon-separated list of prefixes that identify the start of file and/or folder names to ignore. Example value: `.svn`

Uploading Final Binaries to Mobile Administrator

Use the Upload-Binaries or Upload-Binaries-Last Ant tasks to build a project and upload the final binaries to Mobile Administrator so that the application is available via the Mobile Administrator App Store.

Note: If you have an existing final binary, you can upload it directly from Mobile Administrator on **Build** tab for the project.

To build a project and upload the final binaries to Mobile Administrator

1. In Software AG Designer, open the project you want to build and upload.

2. Click the **Project Explorer** view, expand the project, and drag the build.xml file to the Ant view.
3. In the Ant view, double-click **Upload-Binaries** or **Upload-Binaries-Last**.
   - If you select **Upload-Binaries**, Mobile Designer opens the Multi-Build dialog. Continue with the next step.
   - If you select **Upload-Binaries-Last**, no further action is required. Mobile Designer builds the last configuration and uploads it to Mobile Administrator.
4. If you selected **Upload-Binaries**, select the device and language combinations that you want to build.

   The Multi-Build dialog lists devices and language combinations for all the project's targets. However, Mobile Designer only uploads the final binaries for the supported platforms. If you select devices that run on unsupported platforms, Mobile Designer performs the build, but does not upload the final binary and logs a notification in the console output.

   Important: Mobile Designer allows you to build the project for multiple devices for a single platform. However, when downloading an application from Mobile Administrator, Mobile Administrator only displays one version per platform. As a result, it is recommended that you select only the universal device for a platform that will be uploaded to Mobile Administrator. If you build for a specific device, a user that is using the same platform, but a different device might download an incompatible application.
5. Update the **Version** field, if necessary.

   When Mobile Designer displays the Multi-Build dialog, it sets the version based on the mobile.admin.upload.version property in the ma.properties file.
6. If you want to retain the code that Mobile Designer generates for each platform, select the **Retain output build files** check box.
7. Click **Multi Build**.

   After Mobile Designer successfully uploads a final binary to Mobile Administrator, if the mobile.admin.upload.open_browser_on_success property is set to `true`, a webpage containing a link you can use to download the mobile application is displayed in the default browser.

Remotely Building a Project

Use the Remote-Multi-Build Ant task to remotely build a project. When a project is built remotely, Mobile Designer transmits the source code to Mobile Administrator. After receiving the source code, Mobile Administrator builds the project on a Mobile Administrator build node that is running the version of Mobile Designer that you

specify. The final binaries from the build are made available on Mobile Administrator and can be installed via the Mobile Administrator App Store.

To remotely build a project

1. In Software AG Designer, open the project you want to remotely build.
2. Click the **Project Explorer** view, expand the project, and drag the build.xml file to the Ant view.
3. In the Ant view, double-click **Remote-Multi-Build**.

   Mobile Designer displays the Remote Multi-Build dialog. This dialog lists only devices for the supported platforms.

   If the mobile.admin.reuse.existing.build_configs property in the ma.properties file is set to `true`, Mobile Designer presets information in the Remote Multi-Build dialog based on build configurations defined in the Mobile Administrator project that corresponds to this Mobile Designer project. If multiple build configurations exist, Mobile Designer uses the latest version.
4. In the **Selected** column, select the check boxes for each device for which you want to remotely build the project.

   Important: When downloading an application from Mobile Administrator, Mobile Administrator only displays one version per platform. As a result, it is recommended that you select only the universal device for a platform. If you build for a specific device, a user that is using the same platform, but a different device might download an incompatible application.

   If the mobile.admin.reuse.existing.build_configs property is set to `true`, Mobile Designer preselects each device for which there is a build configuration.
5. In the **Target name** column, specify the name of the target device you want to remotely build.

   Important: Be sure names you specify match the names in the build configurations of the corresponding Mobile Administrator project.

   If the mobile.admin.reuse.existing.build_configs property is set to `true`, Mobile Designer specifies device names from build configurations.
6. Leave the information in the **Languages** column as is.

   You cannot configure the languages to use for the build. Mobile Administrator sets the language using the first language specified on the `project.handset.`*`handset`*`.langgroups` project property. For example, if the `project.handset.ios_apple_universal.langgroups` property is set to `en;de`, Mobile Administrator uses `en`.
7. In the **API SDK** column, specify the platform SDK that you want to use for compiling.

   If the mobile.admin.reuse.existing.build_configs property is set to `true`, Mobile Designer presets this field to the platform SDK specified in the build configurations.

8. In the **Certificate** column, specify how to sign the mobile application.
   - For an Android devices, specify a certificate to use for signing.
   - For iOS devices, specify the Provisioning Profile to use for signing.

   If the mobile.admin.reuse.existing.build_configs property is set to `true`, Mobile Designer presets this field based on the information in the build configurations.
9. In the **MD SDK** column, specify the version of Mobile Designer you want to use to build the project, for example, `8.2.7.1.353`.

   Important: Be sure a Mobile Administrator build node running the version of Mobile Designer you specified exists. If no such build node exists, the remote build job will remain in a PENDING status indefinitely.

   If the mobile.admin.reuse.existing.build_configs property is set to `true`, Mobile Designer presets this field based on the information in the build configurations.
10. In the **Retain output** column, select the check box if you want to retain the platform code the remote Mobile Designer generates on the build node.

    If the mobile.admin.reuse.existing.build_configs property is set to `true`, Mobile Designer presets this field based on the information in the build configurations.
11. Leave the **Upload data method** field as is.

    This field describes how Mobile Designer transmits the source code for the project to Mobile Administrator. To transmit the source code, Mobile Designer compresses the project files and folders you specify into a .zip file and sends the .zip file to Mobile Administrator. You specify the source code files and folders that you want to transmit to Mobile Administrator using the following Mobile Administrator plug-in properties:
    - mobile.admin.data.transmission.zip.includes
    - mobile.admin.data.transmission.zip.excludes
    - mobile.admin.data.transmission.zip.ignores
12. Update the **Version** field, if necessary.

    When Mobile Designer displays the Multi-Build dialog, it sets the version based on the mobile.admin.upload.version property in the ma.properties file.
13. Click **Remote Multi-Build**.

    Mobile Designer display status information for the job in the Remote Multi-Build dialog. For more information, see "Monitoring Jobs Used to Remotely Build Projects" on page 209.

Monitoring Jobs Used to Remotely Build Projects

Mobile Designer displays the following status information in the Remote Multi-Build dialog for the build jobs that Mobile Administrator starts to perform the remote build.

Note: You can also view status information in Mobile Administrator on the **Build** tab of the Mobile Administrator project.

Column	Description
Job ID	Specifies a unique ID for a remote build job. You can request information about the build job by opening the following URL in a web browser: http://*Mobile_Administrator_hostname:port* /build_jobs/*jobid* where: ■ *Mobile_Administrator_hostname:port* is the hostname and port number for the Mobile Administrator running the remote build job ■ *jobid* is the job ID listed in the Remote Multi-Build dialog
Handset	Specifies the name of the device for which the build job is running.
Status	Specifies the current status of the build job. The following describes the possible statuses: ■ **PENDING** when Mobile Administrator has queued the job and is waiting for a free build node. **Note:** If Mobile Administrator does not have a build node that is running the version of Mobile Designer that you specified in the **MD SDK** field on the Remote Multi-Build screen, the job will stay in the PENDING status indefinitely. ■ **RUNNING** when the job is running on the build node. ■ **SUCCESS** when the job has completed successfully. ■ **ERROR** when the job is completed, but ended with errors. When errors occur, Mobile Administrator displays the build output in the default web browser.

Column	Description
MD SDK	Specifies the version of Mobile Designer that is running in the build node that Mobile Administrator is using for the remote build. **Note:** This is the version you specified in the **MD SDK** field on the Remote Multi-Build screen.
Host	Specifies the host name build node that Mobile Administrator is using for the remote build.

A Project Properties Reference

Note: You are not required to define values for all the properties described in this reference because Mobile Designer defines values for most of the properties. However, if you need to set a value or change a predefined value for your project, you can. For instructions, see "Setting Project Properties" on page 119.

Build Results Properties

The build results properties customize how Mobile Designer creates the final binary for the project.

packager

Specifies the packager that you want Mobile Designer to use to create the JAR for each J2ME device that your project supports.

Note: Mobile Designer does not use this property for platforms that do not use JAR files, for example, iOS and Android.

If you are building your application for a platform that does use JAR files, be sure to set the property to a packager that creates JARs that the devices in your project can run. Some devices might not be able to execute JARs created by some packagers.

Platforms All

Value Specify one of the following values:

Value	Meaning
`jar`	Mobile Designer uses the built-in JAR packager that is part of the JDK.
`zip`	Mobile Designer provides compression with WinZip.
`7zip`	Mobile Designer provides compression with 7Zip. To create the JAR Mobile Designer uses the following command-line parameters when executing 7Zip: `${packager.7zip.args1} "filename" * ${packager.7zip.args2}` The `packager.7zip.args1` and `packager.7zip.args2` properties are defined in the bs-defaults.xml file, which is in the following location: *Mobile Designer_directory*/Tools/Build/BuildScript/v1.0.0/bs-defaults.xml

The following shows the default property settings in the bs-defaults.xml file:

```
<property name="packager.7zip.args1"
          value="a -r -tzip"/>
<property name="packager.7zip.args2"
          value="-mx=9 -mfb=255"/>
```

`kzip` Mobile Designer provides compression with Kzip.

To create the JAR Mobile Designer uses the following command-line parameters when executing Kzip:

```
${packager.kzip.args} "filename" *
```

The `packager.kzip.args` property is defined in the bs-defaults.xml file.

The following shows the default property setting in the bs-defaults.xml file:

```
<property name="packager.kzip.args"
          value="-r -zl121"/>
```

Default `kzip`

If you do not have the KZip packager installed, at run time Mobile Designer uses 7Zip.

project.handset.skip.compilation

Specifies whether you want Mobile Designer to skip the build step. You can have Mobile Designer skip the build step if you want to use native platform tools to compile and create the final binary.

Platforms All

Value

- `true` to skip the build.

  Mobile Designer performs all the resource handling and cross-compiling (for platforms that require it), but does *not* compile the final binary.

  Important: When you set this property to `true`, be sure to select **Retain output build files** in the Multi Build dialog. When you select **Retain output build files**, Mobile Designer retains the cross-compiled code it generated from your original mobile code, along with any project (for example, Xcode project for iOS) it might have generated. You can then use the retained files and the native platform tools to compile to create the final binary.

- `false` to have Mobile Designer perform the build step to compile and create the final binary.

  Mobile Designer performs all the resource handling, cross-compiling, *and* compiles the final binary.

Default `false`

project.jarname.format

Specifies the file name format that you want Mobile Designer to use when creating the final binary for a build of the project.

Platforms All

Value File name format that you define by specifying one or more of the parameters listed below. At run time, Mobile Designer replaces the parameters with the values listed below.

- **@PROJECT@**

  is replaced with: `${project.jar.name}`

- **@LANGPROJECT@**

  is replaced with: `${project.jar.name.`*`selected-langgroup`*`}`, if it exists, otherwise replace with the value of `${project.jar.name}`

- **@HANDSET@**

  is replaced with: `${mobiledesigner.handset.`*`devicegroup`*`.output.filename}`

- **@LANGGROUP@**

  is replaced with: Selected language group

- **@VMAJOR@**

  is replaced with: X part of the project version number (X.y.z) that you specify in the Multi Build dialog when building the project.

- **@VMINOR@**

  is replaced with: Y part of the project version number (x.Y.z) that you specify in the Multi Build dialog when building the project.

- **@VMICRO@**

  is replaced with: Z part of the project version number (x.y.Z) that you specify in the Multi Build dialog when building the project.

Example:

```
<property name="project.jarname.format"
          value="@PROJECT@@HANDSET@@LANGGROUP@"/>
```

Note: The resulting file name must contain only valid characters are alphanumeric (A-Z, a-z, 0-9), period (.), and hyphen (-).

Default None.

Note: You must specify this property for a project.

project.jarname.format.override.*device_name*

Overrides the JAR file name format value specified in the `project.jarname.format` property for a specific device.

When specifying the property, replace *`device_name`* with the name of the device for which you want to override the JAR file name. You can find the device name in the **Handset** field of the Multi Build dialog. For example, to specify the property for the Apple iPhone 5 phone, use the following property, where the `IOS_Apple_iPhone5` portion of the property name is the device name for the Apple iPhone 5 phone:

```
project.jarname.format.override.IOS_Apple_iPhone5
```

Platforms All

Value See the value for the project.jarname.format property

Default No default.

Mobile Designer uses the value of `project.jarname.format` for all devices that you do not specifically override using the `project.jarname.format.override.`*`device_name`* property.

project.multibuild.built.handset.list

Contains a semicolon-separated list of all the devices that Mobile Designer built when building the project. Mobile Designer sets this property.

Platforms All

Value Semicolon-separated list of all the builds that Mobile Designer created

Default n/a

project.output._temp_.folder

Overrides the default name for the project folder that contains the output data from a build of the project.

Platforms All

Value Folder name that you define by specifying the one or more of the parameters listed below. At run time, Mobile Designer replaces the parameters with the values listed below.

Parameter	Replace with the value of
@LANGGROUP@	${selected.langgroup}
@PLATFORM@	${selected.platform}
@TARGET@	${selected.target}

Example:

If you want Mobile Designer to store the output data in a project folder named "_temp_EFIGS_j2me-jar_release_" for an English, French, Italian, German, Spanish (EFIGS) release J2ME build, specify the following:

```
<property name="project.output._temp_.folder"
          value="_temp_@LANGGROUP@_@PLATFORM@_@TARGET@_"/>
```

Default project _temp_ folder

If you do not set this property, Mobile Designer stores the output and compilation directories in a project _temp_ folder alongside the compiled binary.

Build Script Properties

The build script properties are properties you are required to put in your project's build.xml file.

additional.device.profiles.dir.root

Specifies the location of a folder that contains custom device profiles for your project.

You might want to create your own custom device profiles, but you do not want to include them in Mobile Designer device database. In this situation, set up a separate folder to contain the your additional, custom device profiles, which must use the same XML format that Mobile Designer uses for the device profiles it provides.

To use the custom device profiles for a project, include this property in the project's build.xml file. You must place it in the build.xml file before the following line that imports the targets.xml:

```
<importxmldirectory location="${basedir}/targets"/>
```

Platforms All

Value Directory that contains custom device profiles

For example:

```
<property name="additional.device.profiles.dir.root"
          value="${basedir}/device_info"/>
```

Default None.

Note: Mobile Designer does not use custom profiles if you do not include this property.

mobiledesigner.buildscript.version

Specifies the version of the Mobile Designer build scripting system to use when building your application. Define this property in the project's build.xml file.

Platforms All

Value `v1.0.0`

Note: Currently the only supported value is `v1.0.0`.This property is for backward compatibility in the event Mobile Designer uses a different build scripting system in the future.

Default None.

Note: You must specify this property for a project.

mobiledesigner.runtime.version

Specifies the version of the Mobile Designer run-time code to use for your application. Define this property in the project's build.xml file.

Mobile Designer

Platforms All

Value `v1.0.0`

Note: Currently the only supported value is `v1.0.0`.This property is for backward compatibility in the event Mobile Designer uses a different run-time code in the future.

Default None.

Note: You must specify this property for a project.

project.jar.name

Specifies a text name you want your application to have when installed on a device. Define this property in the project's build.xml file.

Platforms All

Value Name for your project. Valid characters are alphanumeric (A-Z, a-z, 0-9), period (.), and hyphen (-).

For example:

```
<property name="project.jar.name" value="HelloWorld"/>
```

Default None.

Note: You must specify this property for a project.

project.java.midlet.name

Specifies the name of the root MIDlet/Application class of your project's run-time code. Typically this is the class that extends com.softwareag.mobile.runtime.core.Application. Define this property in the project's build.xml file.

Platforms All

Value For example:

```
<property name="project.java.midlet.name"
          value="com.softwareag.mobile.helloworld.MyApplication"/>
```

Default None.

Note: You must specify this property for a project.

Code Conversion Properties

The code conversion properties configure how Mobile Designer creates the generated C#, C++ or Java code for the project.

java.parser.retain.comments

Specifies whether to retain comments when compiling to Java, C# or C++.

Platforms All

Values
- `true` retains comments when compiling to Java, C# or C++.

  Mobile Designer attempts to keep comments connected to the line where the comments are relevant, typically the line of code that follows the comment. However, due to the fundamental difference between Java, C# and C++, some comments might get misplaced or lost.

  Note: If a project has been built without comments and the property is set to `true`, the cached parsed-Java files do not contain comments to use. To resolve this issue, clear out the project's _temp__bcdfs_cache_ folder.
- `false` removes comments when compiling to Java, C# or C++.

Default `true`

cpp.no.selfprotect

Indicates whether to add a safeguard to every method call.

Platforms All

Values
- `true` indicates that you do *not* want to use the safeguard.

  When set to `false`, the cross compiler might inject the macro line, but it does nothing.
- `false` adds the safeguard to every method call using the injected macro line.

  Caution: Adding the injected macro line to every method call adds overhead to all methods and increases the output size.

Default `false`

cross.compiler.extractinners

Indicates whether to extract inner classes included in the Java source code.

Platforms All

Values
- `true` extracts the inner classes.

Note: The examination required to perform this action increases the processor time.

- `false` does not extract the inner classes.

Default `true`

cross.compiler.nodatestamp

Specifies whether to exclude timestamps at the top of every generated C++ (CPP) and H file.

Platforms All

Values
- `true` indicates that timestamps are *not* included.

  If you are storing the files in a source control system, omitting this header information reduces the file differences that occur with each cross compilation.
- `false` indicates that timestamps are included.

Default `false`

cross.compiler.render.selfprotect

Indicates whether to add an extra line at the top of every non-static method to prevent self-destruction of the object during the method call.

Platforms All

Values
- `true` indicates that the extra line is added.

  This is part of a fix needed to prevent object self-deletion on the rare occasion that an object nulls all references to itself. The injected line is a macro whose actual contents you can remove by the value of the `cpp.no.selfprotect` property.
- `false` indicates that the extra line is *not* added.

Default `true`

Cross-Compiler Properties

The webMethods Mobile Designer cross-compiler libraries contain code to support the Mobile Information Device Profile (MIDP) and Connected Limited Device Configuration (CLDC) standards and Java functionality. However, your application might not require the entire library, or you might want to override default values. In this case, you can use the cross-compiler properties to customize your use of the cross-compiler library, for example, to disable areas of the libraries or to override default functionality.

The following lists the types of cross-compiler project properties you can set.

2D and 3 D Rendering

project.handset.bitmapsystemfont

Specifies whether you want Mobile Designer to embed a bitmap font in the final build.

Note: By default, Mobile Designer renders system liquid crystal display user interface (LCDUI) fonts using a bitmap font rendering system.

Platforms All

Value
- `true` if an application uses system fonts and you want to embed a bitmap font resource.

- `false` if an application does not use system fonts. Specifying `false` saves resource space.

  To use native fonts, see the project.runtime.render.system.font.using.native.font property.

Default The default is based on the platform:

Platform	Default
iOS	`true`
Other platforms	`false`

project.runtime.attempts.2d.graphics.over.3d

Specifies whether you want the application to attempt to use the system's underlying 3D API, for example OpenGL-ES, to perform all standard 2D Mobile Information Device Profile (MIDP) rendering calls.

Platforms iOS

Value

- `true` to use the system's underlying 3D API to perform all standard 2D MIDP calls.

  The application translates images into textures and maps all graphical method calls to a corresponding 3D call. This can help access the underlying 3D hardware.

  Note: The texture map size can exceed the equivalent pixel buffer memory allocation. Some platforms require 2D textures. As a result, an image that is 257x17 can turn into a texture map that is 512x32, resulting in a large memory increase required for images in your application, resulting in a large increase in the amount of memory that your application requires for images. Essentially, for both the width and height, the size is rounded up to the nearest power of two.

  Note: For the iOS platform, the application must perform all rendering on the same thread as the OpenGL-ES library, which is not multi-thread safe or capable.

- `false` if you do not want the application to use the system's underlying 3D API to perform standard 2D MIDP calls.

Default `false`

project.runtime.attempts.2d.graphics.over.3d.free.immutable.image.memory

Specifies whether the application frees the internal memory store of all immutable images after rendering the images and creating the associated 3D texture map.

> **Note:** This property is only relevant when project.runtime.attempts.2d.graphics.over.3d is set to `true`.

Platforms All

Value
- `true` if you want the application to free the memory store. Specifying `true` reduces the memory overhead of the image rendering.

  > **Note:** When this property is `true`, the application cannot use the Image.getRGB method with the image.

- `false` if you do not want the application to free the memory store.

Default `false`

project.runtime.render.system.font.using.native.font

Specifies whether the LCDUI Font class uses the device's native font-rendering mechanism.

Platforms iOS

Value
- `true` if an application uses the device’s native font-rendering mechanism.
- `false` if an application does not use the device’s native font-rendering mechanism.

  > **Note:** For more information about using system fonts, see project.handset.bitmapsystemfont.

Default `false`

project.runtime.uses.2d.graphics

Specifies whether the application uses 2D graphics and that you want the application to prepare the renderer for their display.

Platforms All

Value
- `true` if an application uses 2D graphics.
- `false` if an application does not use 2D graphics.

Default `true`

project.runtime.uses.3d.graphics

Specifies whether the application uses 3D graphics and that you want the application to prepare the renderer for their display.

Platforms All

Value
- `true` if an application uses 3D graphics.
- `false` if an application does not use 3D graphics.

Default `false`

Debugging

project.handset.log.debug.filename

Provides the file name to use for the debug log file in which to log debug information when the application uses the debug function call.

Note: To have debug statements written to an output file, you must set the project.handset.log.debug.to.file property to `true`.

Platforms iOS

Value File name for the debug log file

Default `LOG_debug.txt`

project.handset.log.debug.to.file

Specifies whether you want the application to write debug statements to an output file when debugging the application through the Mobile Designer Multi Build dialog box.

Platforms iOS

Value
- `true` to write debug statements to an output file.

By default, the output file name is LOG_debug.txt. You can specify an alternate name using the "project.handset.log.debug.filename" on page 224 property.

- `false` if you do not want the application to write debug statements to an output file.

Default `false`

Extra Libraries and Custom Code

project.handset.hook.startup

Specifies whether you want Mobile Designer to invoke the startup hook, which is a native function, when the application is starting before the J2ME initialization occurs.

When creating the code to invoke, implement the following function in your native code or patch in the created StubInfo.cpp file:

```
void hookStartUp(void);
```

Caution: Mobile Designer invokes the native code before setting up the J2ME environment, for example, before initializing static data or generating system output streams. Do *not* include code related to the J2ME library or to the mobile application in the native code.

Platforms All

Value

- `true` to invoke native code before the J2ME application code is started.

  Mobile Designer invokes hookStartUp() at the beginning of the application in these locations:

  - In iOS at the start of applicationDidFinishLaunching
  - On all other platforms at the start of main or its equivalent
- `false` if you do not want Mobile Designer to invoke native code before the J2ME application code is started.

Default `false`

project.handset.push.notifications

Specifies whether the application receives push notifications.

Note: Use of this property requires the Wireless Messaging API (WMA) library for J2ME and the project.handset.uses.WMA property set to `true`.

Platforms Android
iOS

Value
- `true` if the application receives push notifications.
- `false` if the application does not receive push notifications.

Default `false`

project.handset.uses.camera

Specifies whether the application uses Mobile Media API (MMAPI) to access the camera functionality.

Platforms All

Value
- `true` if the application uses MMAPI to access the camera.
- `false` if the application does not use MMAPI to access the camera.

Default `false`

project.handset.uses.Database

Specifies whether your mobile application uses the com.softwareag.mobile.runtime.database classes, which Mobile Designer provides. For more information, see "Run-Time Database Classes" on page 76.

Platforms All

Value
- `true` if your application uses the database classes.
- `false` if your application does *not* use the database classes.

Default `false`

Note: If you specify a value for the mobilesupportclient.runtime.dir property, the value of this property is automatically set to `true` because the Mobile Support Client library requires the com.softwareag.mobile.runtime.database classes.

project.handset.uses.FCPIM

Specifies whether your application uses the Personal Information Management (PIM) library for J2ME.

Platforms
Android
iOS
Windows Phone 8

Value

- `true` if your application uses the PIM library.

  When you set this property to `true`, necessary permissions or requests are added to native builds.

- `false` if your application does *not* use the PIM library.

  Note: If your applications uses the PIM library, but you set this property to `false` (or it defaults to `false`), your application will fail to compile.

Default `false`

project.handset.uses.libJPEG

Specifies whether you want Mobile Designer to include the libjpeg library in the project.

Platforms All

Value

- `true` if an application uses the JPEG images that are loaded using the Image.createImage methods.
- `false` if an application does not require loading JPEG images using the Image.createImage methods.

  Important: When the property is `false`, attempting to load an image using an Image.createImage method causes the method to return a null value.

Default `true`

project.handset.uses.Location

Specifies whether an application requires the Location API library for J2ME to compile and that you want Mobile Designer to initialize static values for the library.

Platforms All

Value
- `true` if an application uses the Location library.
- `false` if an application does not use the Location library. Specifying `false` reduces the final size of your project's binaries.

> **Important:** If an application uses the Location library, and you set the property to `false`, Mobile Designer cannot compile the application.

Default `true`

project.handset.uses.Sensors

Specifies whether an application requires the Mobile Sensor API library for J2ME to compile and that you want Mobile Designer to initialize static values for the library.

Platforms All

Value
- `true` if the application uses the Sensors library.
- `false` if the application does not use the Sensors library. Specifying `false` reduces the final size of your project's binaries.

> **Important:** If the application uses the Sensors library, and you set the property to `false`, Mobile Designer cannot compile the application.

Default `true`

project.handset.uses.WebServices

Specifies whether an application requires the Web Service API library.

> **Note:** The JSON library is also part of the Web Service API library.

Platforms All

Value
- `true` if the application uses the Web Service API library.
- `false` if the application does not use the Web Service API library.

Default `false`

project.handset.uses.WMA

Specifies whether an application requires the Wireless Messaging API (WMA) library for J2ME to compile and that you want Mobile Designer to initialize static values for the library. WMA enables sending Short Message Service (SMS), Multimedia Messaging Service (MMS), and Cell Broadcast Service (CBS) formats.

Platforms All

Value
- `true` if the application uses the WMA library.
- `false` if the application does not use the WMA library. Specifying `false` reduces the final size of your project's binaries.

> **Important:** If the application uses the WMA library, and you set the property to `false`, Mobile Designer cannot compile the application.

Default `true`

Java Classes

cpp.class.forname.inclusion.list

Specifies a list of classes that you want to be made available to be referenced at run time using Class.forName. Using this property to explicitly specify a list of classes prevents the application from unnecessarily linking to every known class whether the application uses the class or not.

Platforms All

Value A semicolon-separated list of classes, for example:

```
java.lang.Object;com.mypackage.MyCanvas
```

Default An empty String

cpp.class.forname.interfaces.inclusion.list

Identifies the names of the interfaces that your application uses. Using this property to explicitly specify the list of interfaces prevents the application from unnecessarily linking to every known interface whether the application uses the interface or not.

Platforms All

Value A semicolon-separated list of interfaces

Default An empty String

cpp.class.newinstance.inclusion.list

Controls the list of classes implemented in the Class.newInstance method.

Platforms All

Value A semicolon-separated list of classes

> **Caution:** List only classes that have default constructors, that is, no parameters. Compilation errors will occur if you specify a class that does not have a default constructor.

Default The value of the `cpp.class.forname.inclusion.list` property

cpp.no.extraexceptions

Specifies whether you want the application to perform the Java-required `NullPointer` and `ArrayIndexOutOfBounds` exception checks for every array access.

Platforms All

Value
- `true` if an application code-base is safe and does not need Java-required `NullPointer` and `ArrayIndexOutOfBounds` exception checks.
- `false` if you want an application to perform the Java-required `NullPointer` and `ArrayIndexOutOfBounds` exception checks.

Default `false`

cpp.no.lcduiforms

Specifies whether you want the application to include extra code in the libraries to support the liquid crystal display user interface (LCDUI) forms.

Platforms All

Value
- `true` if an application does *not* use LCDUI forms. Specifying `false` reduces the final size of your project's binaries.
- `false` if an application uses LCDUI forms.

Default `false`

csharp.class.forname.inclusion.list

Specifies a list of classes that you want to be made available to be referenced at run time using Class.forName. Using this property to explicitly specify a list of classes prevents the application from unnecessarily linking to every known class whether the application uses the class or not.

Platforms All

Value A semicolon-separated list of classes, for example:

```
java.lang.Object;com.mypackage.MyCanvas
```

Default An empty String

csharp.class.forname.interfaces.inclusion.list

Identifies the names of the interfaces that your application uses. Using this property to explicitly specify the list of interfaces prevents the application from unnecessarily linking to every known interface whether the application uses the interface or not.

Platforms All

Value A semicolon-separated list of interfaces

Default An empty String

csharp.class.newinstance.inclusion.list

Controls the list of classes implemented in the Class.newInstance method.

Platforms All

Value A semicolon-separated list of classes

> **Caution:** List only classes that have default constructors, that is, no parameters. Compilation errors will occur if you specify a class that does not have a default constructor.

Default The value of the `csharp.class.forname.inclusion.list` property

project.javac.encoding

Specifies the character encoding used in your application's Java source code.

Note: Mobile Designer also supports other Java supported character sets such as UTF-8 and SJIS for the Shift-JIS, Japanese character set.

Platforms All

Value Character encoding

Default `8859_1` for ISO 8859-1, Latin Alphabet No. 1

Makefile Additions

Note: Makefiles are now deprecated and will be removed in a later version of Mobile Designer. At this point, Mobile Designer will instead make use of the generated project file to build the mobile application (i.e. Xcode for iOS builds, VisualStudio for Windows Phone builds).

Use the Makefile Additions properties to link to third-party native-platform libraries and/or to include more complex native alterations to your applications.

Note: Unless indicated otherwise, Mobile Designer duplicates your settings in the corresponding Visual Studio or Xcode project.

project.cpp.additional.compiler.options

Specifies extra compiler flags or settings that you want Mobile Designer to include when compiling an application.

Note: Mobile Designer does not include the flags and settings in the Visual Studio or Xcode project.

Platforms All

Value Flags and settings as required by the specified makefile for the target platform

Default An empty String

project.cpp.additional.defines

Specifies `#define` statements that you want Mobile Designer to use when compiling an application.

Platforms All

Value A semicolon-separated list of `#define` statements

Default An empty String

project.cpp.additional.includes.path

Identifies the name of an Ant path that includes a list of folders or files that you want Mobile Designer to reference for all C++ `#include` statements in a project.

Platforms All

Value Name of the Ant path

Default An empty String

project.cpp.additional.libs.path

Identifies the name of an Ant path that includes a list of folders containing additional libraries used in a project.

Platforms All

Value Name of the Ant path

Default null

project.cpp.additional.libs.post

Identifies libraries to add to the end of the referenced library list in the makefile. The libraries you specify with this property are placed after the default libraries that Mobile Designer automatically includes.

Platforms All

Value A semicolon-separated list of libraries

Do not include file extensions or makefile-specific encoding. Mobile Designer will set the appropriate value for a platform. For example, if you set `project.cpp.additional.libs.post` to `mylib1;mylib2`, Mobile Designer might change the value to "-lmylib1 –lmylib2" on one platform, and "mylib1.lib mylib2.lib" on a different platform.

Default An empty String

project.cpp.additional.libs.pre

Identifies libraries to add to the beginning of the referenced library list in the makefile. The libraries you specify with this property are placed before the default libraries that Mobile Designer automatically includes.

Platforms All

Value A semicolon-separated list of libraries

When specifying the libraries, do not include file extensions or makefile-specific encoding.Mobile Designer will set the appropriate value for a platform. For example, if you set `project.cpp.additional.libs.pre` to `mylib1;mylib2`, Mobile Designer might change the value to "-lmylib1 –lmylib2" on one platform, and "mylib1.lib mylib2.lib" on a different platform.

Default An empty String

project.cpp.additional.linker.options

Specifies linker flags or settings that you want Mobile Designer to include when generating the final binary and linking the compiled code with the defined external C++ libraries.

Note: Mobile Designer does not include the flags and settings that you specify in the Microsoft Visual Studio or Apple Xcode project.

Platforms All

Value Flags and settings as required by the specified makefile for the target platform

Default An empty String

project.handset.custom.stubfolder

Specifies a stub makefile and/or project template to override those that Mobile Designer supplies.

Platforms iOS

Value Path to the copy of the stub makefile and/or project template you want to use

Default An empty String

Optimization

project.handset.bitmapsystemfont

Specifies whether the application uses system fonts and that you want Mobile Designer to embed a bitmap font resource in the final build of an application.

Note: Mobile Designer renders the system liquid crystal display user interface (LCDUI) fonts using a bitmap font rendering system.

Platforms All

Value

- `true` if an application uses system fonts and you want to embed a bitmap font resource.
- `false` if an application does not use system fonts. Specifying `false` saves resource space.

  Note: To use native fonts, see the project.runtime.render.system.font.using.native.font property.

Default The default is based on the platform:

Platform	Default
iOS	`true`
Other platforms	`false`

project.in.code.manifest

Specifies whether you want Mobile Designer to delete the META-INF folder, which determines how your application can access the manifest properties.

Platforms All

Value
- `true` to keep the META-INF folder. The application can manually access the manifest.mf file using the following path:

  /META-INF/MANIFEST.MF
- `false` if you want Mobile Designer to delete the META-INF folder. The application can access the manifest properties using the MIDlet.getAppProperty method.

Default `true`

project.runtime.http.connection.timeout.ms

Specifies the number of milliseconds an application waits for an HTTP or HTTPS connection before assuming a timeout has occurred.

Platforms Android
iOS
Windows 8
Windows RT
Windows Phone

Value Numeric value

Default The default value is based on the platform.

Platform	Default
Android iOS	`15000`
Windows 8 Windows RT Windows Phone	`30000`

Orientations

project.handset.landscape.mode

Specifies whether you want the application to operate the device's screen in landscape mode or portrait mode.

Platforms
Android
iOS

Value
- `true` if the application uses landscape mode.
- `false` if the application uses portrait mode.

Default `false`

project.handset.orientation.limiter

Specifies a bit field that indicates the device orientations that the application supports. The orientations are clockwise from standard portrait.

Platforms
iOS
Windows 8
Windows RT
Windows Phone 8

Value
Bit field

Set this bit...	To indicate this orientation is supported...
`0`	All orientations
`1`	0°
`2`	90°
`4`	180°
`8`	270°

Default `0`

Note: If an application supports multiple orientations, but not all, set all the bits that correspond to the supported orientations. For example, if an application supports 0° (bit 1) and 180° (bit 4), set the value to 5.

project.handset.portrait.mode.orientation

Specifies the degree of rotation required to force a device's screen display to portrait mode.

Platforms All

Value Value to indicate the orientation

Use this value...	For this orientation...
0	0°
1	90°
2	180°
3	270°

Default 0

Screen and Display Handling

project.runtime.callserially.stack.size

Specifies the maximum number of Display.callSerially method calls that an application can stack during its execution. If the specified value is exceeded, the call is ignored and a warning message is passed to the console.

Platforms All

Value 0 or positive number

Default 4

project.runtime.canvas.total

Specifies the maximum number of canvases an application can create at one time.

Note: The number of canvases an application uses at any one time impacts the management of repaints. Set the project.runtime.uses.threaded.repaint property to handle the load.

Platforms All

Value Numeric value

Default `1`

project.runtime.statusbar.visible

Specifies whether you want the application to leave room so that a device's status bar displays. The status bar is where a device displays information, such as battery strength, signal strength, and the time.

Platforms Android
iOS

Value
- `true` if you want the status bar visible.
- `false` if you do not want the status bar visible.

Default Depends on whether the application uses the Native user interface library (com.softwareag.mobile.runtime.nui) for its user interface:
- If an application uses the NativeUI library, the default is `true`.
- If an application does *not* use the NativeUI library, the default is `false`.

project.runtime.uses.threaded.repaint

Specifies whether you want the application to manage repaint calls on the application thread or use a second thread for repaint calls.

Note: To manage repaint actions, you might also need to specify the number of canvases allowed in your mobile application. For more information, see the project.runtime.canvas.total property.

Platforms All

Value
- `true` if you want the application to use a secondary thread for the repaint calls.

- `false` if you want the application to use the application thread for repaint calls.

Default `false`

Threading

project.handset.primary.thread.sleep.time.ms

Specifies the number of milliseconds for the primary thread sleep time that occurs on the device.

Note: Yield is the acknowledgment to the system that the thread is giving up its hold on the CPU. The value for yield is 0 (zero).

Platforms iOS

Value Numeric value

Default `5`

project.handset.thread.stacksize

Specifies the number of bytes to allocate for the stack when the application creates a thread.

Platforms All

Value Numeric value

Default The default is based on the platform:

Platform	Default
iOS	`524288`
Other platforms	`32768`

project.handset.wait.for.thread.termination

Specifies whether the application waits for running threads to terminate before exiting the application.

Platforms All

Value
- `true` causes the application to wait for threads to terminate before exiting. Your mobile application and all of its threads need to detect any incoming `destroyApp` calls and terminate gracefully.
- `false` if you do not want the application to wait for threads to terminate before exiting.

Default `false`

project.runtime.object.wait.sleep.time.ms

Specifies the number of milliseconds that you want a thread to sleep while waiting for an object to receive a wait notification.

Note: Yield is the acknowledgment to the system that the thread is giving up its hold on the CPU. The value for yield is 0 (zero).

Platforms All

Value Numeric value

Default `5`

project.runtime.sync.lock.sleep.time.ms

Specifies the number of milliseconds that you want a thread to sleep while waiting to attain ownership of a lock.

Note: Yield is the acknowledgment to the system that the thread is giving up its hold on the CPU. The value for yield is 0 (zero).

Platforms All

Value Numeric value

Default `0` (zero)

project.runtime.uses.user.threading

Specifies whether the application uses an internal user threading model or native threading model.

Platforms iOS (not the simulator)

Value
- `true` if an application uses an internal user threading model.

  Using an internal user defined threading model can help avoid potential thread limitations.
- `false` if an application uses a native threading model.

Default `false`

User Input

project.handset.threaded.inputs

Specifies whether the system should route touch screen events through the MIDlet's primary thread or through a secondary thread.

Platforms iOS

Value
- `true` to route touch screen events through the MIDlet's primary thread.
- `false` to route touch screen events through a secondary thread.

Default `false`

Android

android.apk

Specifies a file name that Mobile Designer uses to override the name for the Android application package file.

Platforms Android

Value File name for the Android application package file

Default File name as specified by the Multi Build dialog

android.backkey.valid

Specifies the action to take when the user presses the Back key, that is, whether to handle the Back key like all other keys or to terminate the application.

Platforms Android

Value
- `true` to use the Back key like other keys.
- `false` to terminate the application.

Default `true`

android.clean.source.folder

Specifies whether you want Mobile Designer to clean the source tree of files that were created when applying patches before Mobile Designer creates the final Android application package file (apk).

Platforms Android

Value
- `true` to clean the source tree. Specify `true` to prevent files created when applying patches from being incorporated into the final application package file.
- `false` to leave the source tree as is before creating the final application package file.

Default `true`

android.direct.to.surfaceview

Specifies whether you want the application to direct Canvas.paint method calls directly to the device's screen or to use *blitting* to copy the surface to the screen.

Platforms Android

Value
- `true` to direct Canvas.paint to render directly to the screen.
- `false` to direct Canvas.paint to render to an off-screen bitmap, which is then blitted to the screen.

Default `false`

android.manifest.permissions

Specifies permissions that you want Mobile Designer to include in the AndroidManifest.xml file.

Platforms Android

Value A semicolon-separated list of permissions

Default An empty String

android.manifest.xml

Specifies an override for the default AndroidManifest.xml file that Mobile Designer generates.

Platforms Android

Value The path to an alternate XML file to use instead of the default AndroidManifest.xml file

Default An empty String

android.min.sdk.version

Specifies the value that you want Mobile Designer to insert into the AndroidManifest.xml file for the `minSdkVersion` setting, which defines the Android application programming interface (API) level.

Platforms Android

Value Value for `minSdkVersion`

Default `4`

android.nativeui.navview.version

Specifies the rendering style for the native user interface view (`nUINavView`).

Platforms Android

Value
- `1` to specify the pop-up Menu navigation view from the Android SDK version 3.x or earlier.

- `2` to specify the always visible navigation view in the Android SDK version 4.x or later.

Default `2`

android.nativeui.view.header.version

Specifies the rendering style for the Android native user interface view display (`nUIViewDisplay`) header.

Platforms Android

Value
- `1` to specify the Android OS 2.3.4 and earlier, API level 10 or lower, thin grey header.
- `2` to specify the Android OS 3.0 and higher, API level 11 and higher, larger black header with app icon and back chevron.

Default `2`

android.orientation.forced

Specifies whether you want the application to force the screen orientation to portrait or landscape, or whether you want the application to allow the screen to automatically rotate based on the device's orientation.

Platforms Android

Value
- `portrait` to force the screen display to use portrait mode.
- `landscape` to force the screen display to use landscape mode.
- An empty string to allow the screen to automatically rotate.

Default An empty String

android.package

Specifies the name of the Java package that Mobile Designer uses for Android builds.

Platforms Android

Value Name of the Java package to use for Android builds

Default com.softwareag.mobile.runtime

android.package.name

Specifies a String to override the name for the Android application package (APK) file.

Platforms	Android
Value	String to use for the APK file name
Default	Value of the `android.package` property with the selected.jarname appended to it For example, if `android.package` is “com.softwareag.mobileruntime” and selected.jarname is “myproject”, the APK name would default to “com.softwareag.mobileruntime.myproject”.

android.target.sdk.version

Specifies the value that you want Mobile Designer to insert into the AndroidManifest.xml file for the `targetSdkVersion` setting. The `targetSdkVersion` setting indicates the highest Android application programming interface (API) level for which you have ensured your mobile application is backward compatible. In other words, you have ensured that your application runs as expected at the level you specify in `targetSdkVersion` down to the level you specify for the android.min.sdk.version property.

Newer versions of the Android platform include behavior that mimics the expected functionality of older versions to support backward compatibility. However, the newer Android platform only uses the backward compatible behavior required to support the API level specified in `targetSdkVersion`. For example, if a device is running Android level 15, but your application has been tested for level 11 and you want backward capability for level 11, set the `android.target.sdk.version` property to 11. In turn, Mobile Designer inserts the value 11 for the `targetSdkVersion` setting in the AndoidManifest.xml file. As a result, the Android level 15 platform mimics the expected behavior of Android level 11 platform when running your application.

Platforms	Android
Value	An integer that designates the Android API level that you want to use for the `<uses-sdk>` element's `targetSdkVersion` setting in the AndroidManifest.xml file.
Default	Value of the android.min.sdk.version property

android.uses.surfaceview

Specifies whether you want the application to use the Android SurfaceView rendering model for 2D graphics.

> **Note:** Applications use the Android SurfaceView rendering model for 3D graphics by default.

Platforms Android

Value
- `true` to enable using the SurfaceView rendering model for 2D graphics.

  > **Note:** Although automatically enabled for 3D graphics, setting this to `true` for 2D graphics will switch the rendering model to SurfaceView from the normal View.

- `false` to disable using the SurfaceView rendering model for 2D graphics.

Default Value of the project.runtime.uses.3d.graphics property

project.additional.libs.path

Specifies the name of an Ant path that includes a list of folders or direct file locations where required native-shared libraries are located.

Platforms Android

Value Name of an Ant path

Default `project.cpp.additional.libs.path`

project.android.sdk.version.override

Specifies the version of the SDK that you want to use for compiling.

Platforms Android

Value Version of the version of the SDK you want to use for compiling

Default `21`

iOS

ios.app.delegate.name

Specifies the name of the application delegate that you want the application to use.

Platforms iOS

Value Name of an alternative delegate application

Default `xyzApp`

ios.background.music

Specifies whether the application allows users to continue playing their audio in the background while the application is running.

Platforms iOS

Value
- `true` to allow the user to play music, which is not a part of the application, in the background while the application is running.

  Specifying `true` also ensures that the audio supports the iPhone's Ring/Silent switch setting.
- `false` if you do not want to allow a user to play music while the application is running.

Default `false`

ios.deployment.target

Specifies the minimum iOS firmware version that the application supports.

Platforms iOS

Value Version number

When specifying a version, use the number only, such as `6.0`.

Default `6.0`

ios.extra.frameworks

Injects a reference to one or more third-party frameworks into the makefile and Xcode project files.

Platforms iOS

Value Path to the framework, for example, `"/Users/mycompany/Desktop/MyFramework.framework"`.

To specify multiple frameworks, separate each path using the character defined in Ant path.separator property. In the following example, path.separator is set to a colon (:).

```
"/Users/mycompany/Desktop/MyFramework.framework:/Users/mycompany/Desktop/Another.framework"
```

Note: You must include the ".framework" portion when specifying a path.

Default An empty string

ios.in.code.default

Specifies whether you want to use the Default.png image within your application code.

Note: iOS applications have a Default.png that defines the initial splash screen that displays when the application loads. By default, the Default.png is a black image with the Software AG logo on it. You can override this default image for your application by adding your own Default.png file in the Resource Handler.

Platforms iOS

Value
- `true` if you want to use the Default.png image within your application code.
- `false` if you do not need to use the Default.png image within your application.

Default `true`

ios.in.code.icon

Specifies whether you want to use the icon.png image within your application code.

Platforms iOS

Value
- `true` if you want to use the icon.png image within your application code.
- `false` if you do not need to use the icon.png image within your application.

Default `false`

ios.info.plist.output.format

Specifies the format you want Mobile Designer to use for the final packaged Info.plist.

Platforms iOS

Value
- `binary` to maintain the Info.plist file in binary format.
- `xml` to maintain the Info.plist file in XML format.

Default `binary`

ios.retained.png.list

Specifies a list of portable network graphic (PNG) files that you want Mobile Designer to retain a .png file rather than generated _png files.

For information about _png files, see the ios.in.code.default property.

Platforms iOS

Value A semicolon-separated list of PNG files

Default An empty String

ios.sdk.version

Specifies the version of the iOS SDK that you want Mobile Designer to use when compiling the application.

Platforms iOS

Value Version number of the iOS SDK

Default Latest version of the SDK

Mobile Designer looks for the installed iOS SDK versions and selects the version with the highest version number

ios.use.retina.display

Specifies whether the application supports the high-resolution retina display of an iOS device.

Platforms iOS

Value
- `true` if the application supports the high-resolution retina display.
- `false` if the application does not support the high-resolution retina display.

Default `false`

ios.use.root.view.controller

Specifies whether the application uses the iOS rootViewController property.

Platforms iOS

Value
- `true` to use the rootViewController property.

  When an application uses the Native User Interface (NativeUI) library (com.softwareag.mobile.runtime.nui) for its user interface, using this setting allows the application to better support rotation of the device. That is, the application is better able to display overlays and NativeUI extensions, for example, the built-in camera and volume selector.
- `false` if you do not want to use the rootViewController property.

  Use this setting when the application does not use the NativeUI library, when the application does not support rotation, or if you want to maintain backwards compatibility for an application you developed using Mobile Designer 9.6 or earlier.

Default Depends on whether the application uses the NativeUI library (com.softwareag.mobile.runtime.nui) for its user interface:
- If an application uses the NativeUI library, the default is `true`.
- If an application does *not* use the NativeUI library, the default is `false`.

project.handset.nativehook

Specifies whether the application requires a callback method that executes from the core user interface thread.

Platforms iOS

Value

- `true` if the application requires a callback method from the core user interface thread. When you specify `true`, add the following function to your code:

  ```
  void objcNativeHook(void);
  ```

  Calls to the `objcThreadHook()` function will trigger the `objcNativeHook(void)` function, which the application calls using a core user interface thread.
- `false` if the application does not require a callback from the core user interface thread.

Default `false`

Windows RT and Windows 8

winrt.application.identity.string

Specifies the value Mobile Designer sets for the `Identity Name` field inside the Package.appxmanifest file, which Mobile Designer generates during the build process.

Platforms Windows RT
Windows 8

Value Application name that you want Mobile Designer to use for the `Identity Name` field in the package manifest file.

Default *vendor .appname*

where:

- *vendor* is the vendor named specified in the project.jad.vendor.name SDK property. If the value contains spaces, Mobile Designer replaces each space with a hyphen (-).
- *appname* is the text name of the application specified in the project.jar.name project property.

Cross-Product Integration Properties

The cross-product integration properties configure how Mobile Designer works with other products in the webMethods product suite.

mobilesupportclient.runtime.dir

Specifies whether to include the Mobile Support Client library in a mobile project. The methods in this library facilitate data synchronization between mobile devices and back-end databases by initiating synchronization requests with webMethods Mobile Support.

For more information about webMethods Mobile Support, see *Developing Data Synchronization Solutions with webMethods Mobile Support*. For more information about the Mobile Support Client library, see *webMethods Mobile Support Client Java API Reference*.

Note: The Mobile Support Client library requires the com.softwareag.mobile.runtime.database classes. As a result, when you set a value for the `mobilesupportclient.runtime.dir` property, the project.handset.uses.Database is automatically set to `true` to indicate that the mobile application uses the com.softwareag.mobile.runtime.database classes.

Platforms All

Values Root directory where the Mobile Support Client library mdlibrary.properties file and src folder reside.

Example:

```
<property name="mobilesupportclient.runtime.dir"
          value="c:/SoftwareAG/Mobile/SupportClient" />
```

Default None.

Device-Specific Properties

The device-specific properties define information for specific devices in your project. Store these properties in your project's targets folder in the target device file for the device to which the property pertains, *device_name* .xml.

project.handset.*device_name* .langgroups

Specifies one or more language groups that indicate the language(s) that your mobile application supports for the device indicated in the property name.

When specifying the property, replace *`device_name`* in the property name with the name of a specific device. The value for a specific device is the value that you selected

from the **Choose your handset** list in the Add Handset dialog when you added the device to the project. For example, for an Apple iPhone 5 phone, the property name is:

```
project.handset.IOS_Apple_iPhone5.langgroup
```

When you added the device to the project, Mobile Designer automatically added this property to the target device file in your project's targets folder.

project_folder /targets/*device_file* .xml

Mobile Designer set the value of the property to the language groups you specified in the **Language Groups** field of the Add Handset dialog. For more information, see "Adding a Device to a Project" on page 88.

You can update the property if you want to add, change, or remove language codes. You can specify the language groups you define with the project.langgroup.group_name property.

Platforms All

Value One or more language groups. To specify multiple language groups, create a semicolon-separated list.

Example

Suppose you use the project.langgroup.group_name property to define the following language groups:

```
<property name="project.langgroup.AMERICAN" value="en;fr;es"/>
<property name="project.langgroup.EUROPEAN"
value="en;fr;it;de;es"/>
<property name="project.langgroup.ASIAN" value="zh;ja"/>
```

To specify that the application supports the American and European language groups for the IOS_Apple_iPhone5 device, use the following:

```
<property name="project.handset.IOS_Apple_iPhone5.langgroup"
          value="AMERICAN;EUROPEAN"/>
```

Default No default.

This property must have a value.

project.manifest

Specifies the location of the manifest.mf file for a device.

Platforms All

Value Absolute path to the manifest file.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

Hook Point Properties

The hook point properties provide the names of Ant targets that you create and that you want Mobile Designer to execute when it is executing Ant targets that are provided with Mobile Designer, for example, the +Multi-Build Ant target. Using hookpoints allows you to customize Mobile Designer processes. For more information about hookpoints, see "Creating Custom Ant Scripts to Run at Predefined Hook Points" on page 149.

project.hookpoint.target.prebuildresources

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *before* it executes the Resource Handler while processing the executing the +Multi-Build, +Multi-Build-Last, +Activate-Handset, +Re-Activate-Handset, or +Run-Phoney-With-Re-Activation Ant tasks.

Platforms	All
Value	Name of an Ant Target you defined
Default	None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.postbuildresources

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *after* it executes the Resource Handler while processing the +Multi-Build, +Multi-Build-Last, +Activate-Handset, +Re-Activate-Handset, or +Run-Phoney-With-Re-Activation Ant task.

Platforms	All
Value	Name of an Ant Target you defined
Default	None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.precompilation

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *before* it performs platform-specific compilation for each build while processing the +Multi-Build or +Multi-Build-Last Ant task.

Platforms All

Value Name of an Ant Target you defined

Default None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.postcompilation

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *after* it performs platform-specific compilation for each build while processing the +Multi-Build or +Multi-Build-Last Ant task.

Platforms All

Value Name of an Ant Target you defined

Default None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.postcrosscompiler

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *after* the cross compiler has converted the project's source to your build's target language.

Note: Mobile Designer only calls this hook point for cross-compiled builds.

Platforms All

Value Name of an Ant Target you defined

Default None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.premakefilegeneration

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *before* it generates the makefile and platform-specific project. Examples of platform-specific projects are a Microsoft Visual Studio project for a Windows Phone platform, Apple Xcode project for the iOS platform, or an Eclipse project for platforms like Android.

Note: Mobile Designer only calls this hook point for cross-compiled builds.

Platforms All

Value Name of an Ant Target you defined

Default None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.prepatch

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *after* it applies the patches to your code.

Note: Mobile Designer only calls this hook point for cross-compiled builds.

Platforms All

Value Name of an Ant Target you defined

Default None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.postpackaging

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *after* it generates a platform-specific build bundle.

The output of the packaging process varies from platform to platform. For example, for Android builds, the output is in an application package (apk) file.

After Mobile Designer executes the Ant target specified by the `project.hookpoint.target.postpackaging` property, if appropriate for the platform, Mobile Designer performs any code signing.

Note: Mobile Designer only calls this hook point for cross-compiled builds.

Platforms All

Value Name of an Ant Target you defined

Default None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.postbuild

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *after* it packages and signs a build.

Platforms	All
Value	Name of an Ant Target you defined
Default	None. If you do not include this property, Mobile Designer does not invoke a hook point.

project.hookpoint.target.postmultibuild

Specifies an Ant target that you created. Mobile Designer calls the Ant target you specify *after* it creates all the builds you selected in the Multi-Build dialog.

Platforms	All
Value	Name of an Ant Target you defined
Default	None. If you do not include this property, Mobile Designer does not invoke a hook point.

Multi-Build Selection Properties

The Multi-Build selection properties contain values that you selected when building the project. Some of the properties are also set when you activate a device in the project.

Mobile Designer sets these properties based on the selections you make in the Multi Build or Activate Handset dialogs. The properties are *not* set in either the _defaults_.xml file or a target device file. The properties are not saved to any file. As a result, you cannot override the settings.

You can use the properties in your Ant scripts so that at run time it can determine information about the current build Mobile Designer is processing or the current device being activated.

selected.handset

Specifies the device for which Mobile Designer is building the application or the device Mobile Designer is activating. In other words, the value is the name of the device you selected in the Multi Build or Activate Handset dialog, for example, `IOS_Apple_iPhone5`. Mobile Designer sets this property.

Platforms All

Value Device you selected in the Multi Build or Activate Handset dialog

Default n/a

selected.jarname

Specifies the name of the JAR file that is listed in the **Filename** field of the Multi Build dialog, for example, `NativeUIDemo_iPhone5_DE`. This is the JAR file name that Mobile Designer uses for the build. Mobile Designer sets this property.

Platforms All

Value JAR file name Mobile Designer is using for the build of the project

Default n/a

selected.langgroup

Specifies the language you selected in the Multi Build or Activate Handset dialog. This is the language for which Mobile Designer is building the application or activating a device. Mobile Designer sets this property.

Platforms All

Value Language you selected in the Multi Build or Activate Handset dialog

Default n/a

selected.platform

Specifies the name of the platform that is listed in the **Platform** field of the Multi Build dialog, for example, `ios-app`. This is the platform for which Mobile Designer is creating the build. Mobile Designer sets this property.

Platforms All

Value Platform for which Mobile Designer is creating the build of the project

Default n/a

selected.target

Specifies the type of executable being built. Mobile Designer sets this property using the value from the **Target** field of the Multi Build dialog, for example, `release`.

Platforms All

Value Type of executable listed in the **Target** field in the Multi Build dialog

Default n/a

selected.version

Specifies the version number for the application that you specified in the Multi Build dialog, for example, `1.0.0`. Mobile Designer sets this property.

Platforms All

Value Version number you specified in the Multi Build dialog

Default n/a

Phoney Properties

The Phoney properties customize the use of Phoney for the project.

phoney.base.params

Specifies Phoney startup options. The startup options you specify with this property are used when you start Phoney via Software AG Designer.

Platforms All

Value One or more of the startup options. For a list of the startup options, see "Phoney Startup Options" on page 164.

Default `-r -i -s 1 -rs 1 -so -ai -p ${project.temp.dir.root}/_build_info_.txt`

Project Language Properties

The project definition properties define settings for your project.

project.langgroup.*group_name*

Defines a language group that specifies one or more languages that your application supports. When specifying the property, replace *group_name* with the name you want to give the language group. Set the value of the property to one or more language codes that indicate the language(s) in the group. You can use this property multiple times to define multiple language groups.

For example, you might want to set up three language groups to define languages for different territories, an American territory, European territory, and Asian territory. To do so, specify the following:

```
<property name="project.langgroup.AMERICAN" value="en;fr;es"/>
<property name="project.langgroup.EUROPEAN" value="en;fr;it;de;es"/>
<property name="project.langgroup.ASIAN" value="zh;ja"/>
```

When displaying the Multi Build dialog for the project, Mobile Designer lists each language group. You can then select from those language groups to identify the languages for which you want to create a build.

When specifying the project.handset.device_name.langgroups property to identify the language groups that a specific device supports, you set the value to one or more language groups you define using this `project.langgroup.`*group_name* property.

Platforms	All
Value	One or more of the following language codes to define the languages in the group. To specify multiple languages, use a semicolon-separated list.
Default	None.

Note: You must specify this property for a project.

Resource Handler Properties

The resource handler properties provide information about the project's resource handler. For more information about how you define the resource handler for a project, see "Defining Resources for a Mobile Application Project" on page 101.

debug.remember.resource.names

Specifies whether you want the Mobile Designer to record the names of the resources included in the build.

Platforms All

Value
- `true` if you want to record the names of the resources.

  When the property is set to `true`, the Mobile Designer run-time debug output references these names directly, instead of the ID numbers.

  Setting the property to `true` results in extra code and a larger data pool resulting from all the resource names that are stored in the final binary.
- `false` if you do *not* want to record the names of the resources.

  You should set the property to `false` when preparing a release build.

Default `false`

mobiledesigner.run.reshandler.with.beanshell

Deprecated. Specifies whether to use the BeanShell provided with Mobile Designer.

Platforms All

Value
- `true` if you want to use the BeanShell provided with Mobile Designer. You do not have to use this if you are *not* using Software AG Designer to develop your mobile applications.
- `false` if you do not want to use a BeanShell. The resource handler code is compiled, and then run with the normal Java executable.

Default `false`

project.audio.file.extensions

Specifies the file extension required for the audio files that a device supports.

Platforms All

Value File extension for the audio, for example `.mp3`, `.wav`, or `.mid`.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

project.audio.spec

Specifies the style of audio that a device supports.

Platforms All

Value Style of audio, for example `mp3`, `wav`, or `midi`.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

project.compiled.resources.info.format

Specifies the file format that you want Mobile Designer to use for the _compiled_resources file. Mobile Designer saves this file in a project's _temp_ directory.

Note: Set this property in the project's _default_.xml file, and *not* in a specific target device file, *target_name* .xml.

Platforms All

Value
- `txt` if you want Mobile Designer to create a text format compiled_resources file (_temp__compiled_resources.txt).
- `xml` if you want Mobile Designer to create an XML format compiled_resources file (_temp__compiled_resources.xml).

Default `txt`

project.jar.midlet.icon.spec

Specifies the icon(s) to use for the application's MIDlet-icon for a specific device.

Platforms All

Value Name of the portable network graphic (.png) file(s) to use for the application's MIDlet-icon.

Use this property to include the required icon(s) so that the icon for your application when it is installed on the target devices meets your company and/or application branding requirements. A best practice is to name the icons so that the name includes the size of the icon, for example icon-64x64-24bit.png. Store the image files in a subfolder that your resource handler can easily reference.

Default — The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

In the device profiles that Mobile Designer provides, the provided icons are Software AG and Mobile Designer icons.

project.java.reshandler.name

Specifies the Java package/class name of the resource handler class you created for your project. For more information about creating a resource handler for your project, see "Coding the Resource Handler" on page 102.

Platforms — All

Value — Name the Resource Handler class name.

Default — None.

Note: You must specify this property for a project.

project.reshandler.additional.libs.path

Specifies the Ant path to additional libraries that your project's resource handler requires. This is not an Ant property. Use the following format to specify the Ant path:

```
<path id="project.reshandler.additional.libs.path">
          <pathelement path="path"/>
</path>
```

For example:

```
<path id="project.reshandler.additional.libs.path">
          <pathelement path="${basedir}/reshandler/libs"/>
</path>
```

Platforms — All

Value — Path to the additional libraries for the project's resource handler

Default None.

Note: If you use additional libraries for the resource handler, you must specify this Ant path for a project.

project.reshandler.src.path

Specifies the path to the project's resource handler script and any associated classes. In other words, the path to the Java class you specify with the project.java.reshandler.name property.

This is not an Ant property. Use the following format to specify the Ant path:

```
<path id="project.reshandler.src.path">
          <pathelement path="path"/>
</path>
```

For example:

```
<path id="project.reshandler.additional.libs.path">
          <pathelement path="${basedir}/reshandler"/>
</path>
```

Platforms All

Value Path to the Java class specified by the project.java.reshandler.name property

Default None.

Note: You must specify this Ant path for a project.

project.resource.dir.root

Specifies the path to the top-level folder that contains the resources (audio files, image files, etc.) for your project. For example, you might have a resources folder that contains subfolders with the resources:

```
MyProject
    resources
        audio
        graphics
        icons
        text
```

For this example, set the property to point to the top-level folder, "${basedir}/resources". For more information about how to specify resources for your project, see "Setting Project Properties for the Resource Handler" on page 108.

Platforms All

Value Path to the top-level folder that contains the project resource files.

Default None.

Note: You must specify this property for a project.

Run-Time Classes Properties

The run-time classes properties customize how the project uses the run-time classes that Mobile Designer provides.

mobiledesigner.runtime.core.class.camera

Overrides the value that Mobile Designer sets for how mobile application uses a device's camera.

Note: The com.softwareag.mobile.runtime.media.CameraHandler class uses this property. For more information, see "Run-Time Media Classes" on page 76.

Platforms All

Value
- `none` if your application does not use the camera, or specific devices that your project supports do not support a camera.
- `jsr135` specifies your application uses the Mobile Media API (MMAPI) package for J2ME devices that support the camera.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.comms.httpconnection

Overrides the value that Mobile Designer sets for how the mobile application initiates and manages HTTP connections.

Note: The com.softwareag.mobile.runtime.comms.HttpConnectionHandler class uses this property. For more information, see "Run-Time Comms Classes" on page 75.

Platforms All

Value
- `none` if your application or specific devices that your project supports are not using HTTP connections.
- `httpstream` if your application is using HTTP connection streams.

Default — The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.comms.messageconnection

Overrides the value that Mobile Designer sets for whether the application uses SMS messaging.

Note: The com.softwareag.mobile.runtime.comms.MessageConnectionHandler class uses this property. For more information, see "Run-Time Comms Classes" on page 75.

Platforms — All

Value
- `none` if your application or specific devices that your project supports are not support SMS messaging.
- `wma` if your mobile application is using the J2ME Wireless Messaging API for SMS messaging.

Default — The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.datatypes

Overrides the value that Mobile Designer sets to determine which primitive data types can be read when parsing your resources with the run-time com.softwareag.mobile.runtime.storage.ResourceDataTypes class.

Note: The com.softwareag.mobile.runtime.storage.ResourceDataTypes class uses this property. For more information, see "Run-Time Storage Classes" on page 78.

Platforms — All

Value
- `cldc11` specifies your mobile application uses CLDC 1.1.

  CLDC 1.1 supports integer numbers, float and double.

Default — The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.graphics.canvas

Overrides the canvas that Mobile Designer sets for each device.

Note: The com.softwareag.mobile.runtime.core.CanvasBase class uses this property. For more information, see "Run-Time Canvas Classes" on page 73.

Platforms All

Value
- `midp1_canvas` for javax.microedition.lcdui.Canvas (Mobile Information Device Profile (MIDP)1.0 J2ME devices)
- `midp2_gamecanvas` for java.microedition.lcdui.game.GameCanvas (Mobile Information Device Profile (MIDP) 2.0 J2ME devices and most newer smartphones)
- `nokiaui_fullcanvas` for com.nokia.mid.ui.FullCanvas (older Nokia J2ME devices)

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.graphics.dimensions

Overrides the value that Mobile Designer sets for the screen height and width.

Note: The com.softwareag.mobile.runtime.core.CanvasDimensions class uses this property. For more information, see "Run-Time Canvas Classes" on page 73.

Platforms All

Value
- `dynamic` if the application responds to the device's orientation changing from portrait to landscape mode, or vice versa.
- `fixed` if the application has a fixed orientation either in portrait or landscape mode.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.graphics.image

Overrides the value that Mobile Designer sets for how your mobile application loads, draws, and manages images.

Note: The com.softwareag.mobile.runtime.media.ImageHandler class uses this property. For more information, see "Run-Time Media Classes" on page 76.

Platforms All

Value
- `midp1` provides the support for the Mobile Information Device Profile (MIDP) 1.0 for J2ME.

  MIDP1 devices cannot perform dynamic transformations. When using this value, Mobile Designer creates multiple versions of the images in memory for rendering at draw-time. Do not create unnecessary transformations because the images consume heap-space.
- `midp2` provides the support for the Mobile Information Device Profile (MIDP) 2.0 for J2ME.
- `nokiaui` is a Nokia UI API extension to MIDP 1.0.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.interrupts

Overrides the value that Mobile Designer sets for how your mobile application detects interrupts.

Note: The com.softwareag.mobile.runtime.core.CanvasInterrupts class uses this property. For more information, see "Run-Time Canvas Classes" on page 73.

Platforms All

Value
- `midp1_canvas` (this is a fixed value)

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.keysandtouch

Overrides the value that Mobile Designer sets for how mobile application detects keypress, touch, or pointer events.

Note: The com.softwareag.mobile.runtime.core.CanvasKeysandTouch class uses this property. For more information, see "Run-Time Canvas Classes" on page 73.

Platforms All

Value
- `midp1_canvas` (this is a fixed value)

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.serialize

Overrides the value that Mobile Designer sets for com.softwareag.mobile.runtime.serialize.Serializer class.

Note: The com.softwareag.mobile.runtime.serialize.Serializer class uses this property. For more information, see "Run-Time Serializer Class" on page 78.

Platforms All

Value
- `cldc11` specifies your mobile application uses CLDC 1.1.

  CLDC 1.1 supports integer numbers, float and double.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.softkeys

Overrides the value that Mobile Designer sets for soft-key labels.

Note: The com.softwareag.mobile.runtime.core.CanvasSoftKeys class uses this property. For more information, see "Run-Time Canvas Classes" on page 73.

Platforms All

Value
- `graphical`

  The graphical solution enables amending the soft-key texts for some devices, while on some devices the soft keys do not trigger key callback events.
- `lcdui`

  In the `lcdui` support, soft-key commands are created and monitored using the optimal setting for each device. `lcdui` uses the `CommandListener` for optimization.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.sound

Overrides the value that Mobile Designer sets for how your application controls sound and vibration functionality.

Note: The com.softwareag.mobile.runtime.media.AudioHandler class uses this property. For more information, see "Run-Time Media Classes" on page 76.

Platforms All

Value
- If your application does not use sound and vibration, specify `none` for the value.
- For smartphones, use `jsr135` for the value.
- For older legacy J2ME devices, specify one of the following values:
  - `nokiaui` specifies your application uses the audio handler in the Nokia user interface API.
  - `samsung` specifies your application uses the Samsung API com.samsung.util package to deliver audio to the device.
  - `sprint` specifies your application uses the audio handler in the com.sprintpcs.media package available on some Sprint Nextel J2ME devices to deliver audio.
  - `vscl` specifies your application uses the Vodafone Service Class Library.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.threading

Overrides the default mechanism that used to manage the primary thread.

Note: The com.softwareag.mobile.runtime.core.CanvasThreading class uses this property. For more information, see "Run-Time Canvas Classes" on page 73.

Platforms All

Value
- `thread` if you want the mobile application to use java.lang.Thread to manage the primary thread.
- `timertask` if you want the mobile application to use java.lang.TimerTask to manage the primary thread.

Default The default is based on the specific device and is set in the device's device profile. You can find the device profiles in the following location:

Mobile Designer_directory/Devices

mobiledesigner.runtime.core.class.ui

Specifies whether your application uses the CanvasNativeUI run-time class.

Platforms All

Value
- `nui` if your application uses the CanvasNativeUI class.
- `menu` if your application uses older technology that uses a pixel-pushing system for rendering and handling menu flow rather than using the CanvasNativeUI class.

Default `menu`

project.numeric.keys.emulate.directionals

Specifies how the keypress response works in your application.

Note: The com.softwareag.mobile.runtime.core.CanvasCore class uses this property. For more information, see "Run-Time Canvas Classes" on page 73.

Platforms All

Value
- `false` to enable each keypress to return its own key code, for example, pressing 2 on a standard keypad results in storing 2 in the pressed key.
- `4way` to enable the numeric keys instead of the directional keys and fire the key code and the associated action, for example pressing 2 on a standard keypad results in storing 2 and Up in the pressed key.
- `8way` to enable the diagonals, for example, pressing 1 on a standard keypad results in storing 1, Up, and Left in the pressed keys.

Note: Enabling `4way` or `8way` controls and referencing directions in an application makes the porting process easier because standard key variations are defined for directional control in Mobile Designer.

Default `4way`

project.runtime.uses.nativeui

Specifies whether your application uses the com.softwareag.mobile.runtime.core.CanvasNativeUI class. For more information about the CanvasNativeUI class, see *webMethods Mobile Designer Native User Interface Reference*.

Platforms All

Value
- `true` if your application uses the CanvasNativeUI class.
- `false` if your application does *not* use the CanvasNativeUI class.

Default `false`

Run-Time Code Compilation Properties

The run-time code compilation properties customize how Mobile Designer compiles your project.

project.runtime.additional.classes.path

Specifies the Ant path to additional precompiled classes to include when building the project. This is not an Ant property. Use the following format to specify the Ant path:

```
<path id="project.runtime.additional.classes.path">
          <pathelement path="path"/>
</path>
```

For example:

```
<path id="project.runtime.additional.classes.path">
          <pathelement path="${basedir}/src/classes"/>
</path>
```

Platforms All

Value Path to the folder that contains the additional precompiled classes.

Default None.

Note: If you use additional precompiled classes, you must specify this Ant path for a project.

project.runtime.additional.stubs.path

Specifies the Ant path to additional stubs that you want Mobile Designer to use when compiling the run-time source code. This is not an Ant property. Use the following format to specify the Ant path:

```
<path id="project.runtime.additional.stubs.path">
          <pathelement path="path"/>
</path>
```

For example:

```
<path id="project.runtime.additional.stubs.path">
          <pathelement path="${basedir}/stubs"/>
</path>
```

Important: This path is used in combination with default classpath entries that Mobile Designer defines. If you set this Ant path, be careful to avoid any duplicate class clashes.

Platforms All

Value Path to the folder that contains the stubs.

Default None.

Note: If you want to use additional stubs, you must specify this Ant path for a project.

project.runtime.project.src.path

Specifies the Ant path to the run-time code to include in the build of the project. This is not an Ant property. Use the following format to specify the Ant path:

```
<path id="project.runtime.project.src.path">
          <pathelement path="path"/>
</path>
```

For example:

```
<path id="project.runtime.project.src.path">
          <pathelement path="${basedir}/src"/>
</path>
```

Platforms All

Value Path to the folder that contains the run-time code.

Default None.

Note: You must specify this Ant path for a project.

Android Project Properties

The Android properties customize how Mobile Designer build applications for Android devices.

project.android.sdk.version.override

Overrides the default Android SDK that Mobile Designer uses for the project.

Platforms Android

Value API number of the SDK you want to use.

Default `21`

For more information about the sdk.properties, see "Configuring Mobile Designer for the Android SDK" on page 38.

B Ant Target Summary

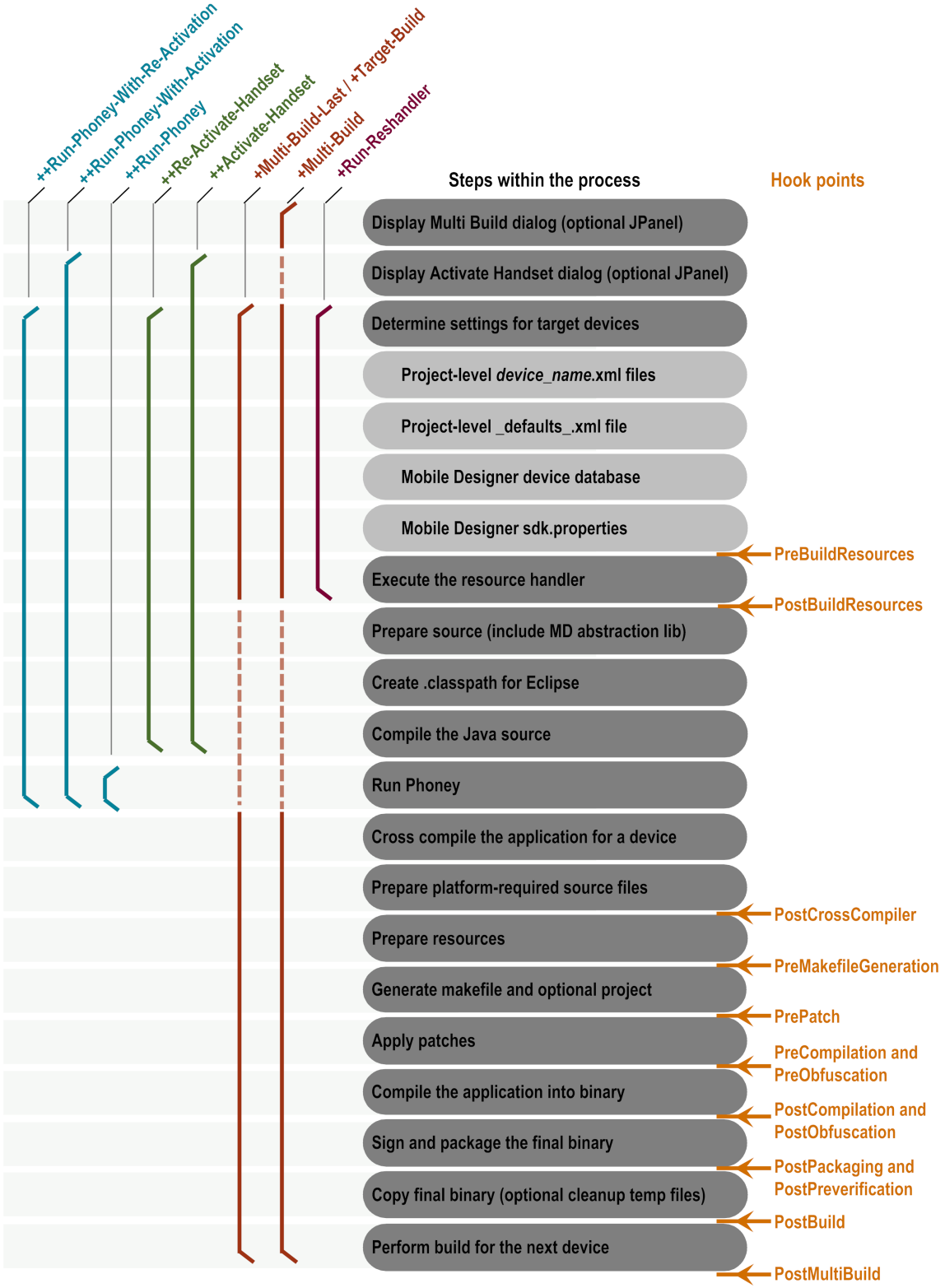
Ant Target Summary

This section provides a diagram that shows the Ant targets you can use to compile resources, build a mobile application project, activate a device, and use Phoney.

For each Ant target, the diagram indicate the steps Mobile Designer performs for an Ant target. For example, for the ++Run-Phoney Ant target, Mobile Designer only performs the “Run Phoney” step. Dashed lines indicate steps that Mobile Designer does not perform. For example, for the +Multi-Build Ant target, there is a dashed line for the “Display Activate Handset dialog (optional JPanel)” step because Mobile Designer does not perform this step when running the +Multi-Build Ant target. For more details about the steps, see one of the following:

- For details about compiling resources only, see "Compiling Resources Using the +Run-Reshandler Ant Target" on page 111.
- For details about the build process, see "Steps in the Multi-Build Process" on page 130.
- For details about the process to activate a device, see "Steps Performed to Activate Handsets" on page 175.
- For details about the actions taken when you run Phoney, see "Steps Performed for Phoney Ant Targets" on page 162.

Use this diagram to compare the actions performed for each Ant target.


++Run-Phoney-With-Re-Activation
++Run-Phoney-With-Activation
++Run-Phoney
++Re-Activate-Handset
++Activate-Handset
+Multi-Build-Last / +Target-Build
+Multi-Build
+Run-Reshandler
Steps within the process
Hook points
Display Multi Build dialog (optional JPanel)
Display Activate Handset dialog (optional JPanel)
Determine settings for target devices
Project-level device_name.xml files
Project-level _defaults_.xml file
Mobile Designer device database
Mobile Designer sdk.properties
PreBuildResources
Execute the resource handler
PostBuildResources
Prepare source (include MD abstraction lib)
Create .classpath for Eclipse
Compile the Java source
Run Phoney
Cross compile the application for a device
Prepare platform-required source files
PostCrossCompiler
Prepare resources
PreMakefileGeneration
Generate makefile and optional project
PrePatch
Apply patches
PreCompilation and PreObfuscation
Compile the application into binary
PostCompilation and PostObfuscation
Sign and package the final binary
PostPackaging and PostPreverification
Copy final binary (optional cleanup temp files)
PostBuild
Perform build for the next device
PostMultiBuild