

webMethods Mobile Development Help

Version 9.6

April 2014

This document applies to webMethods Mobile Development Version 9.6 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.

Copyright © 2014 Software AG, Darmstadt, Germany and/or Software AG USA Inc., Reston, VA, USA, and/or its subsidiaries and/or its affiliates and/or their licensors.

The name Software AG and all Software AG product names are either trademarks or registered trademarks of Software AG and/or Software AG USA Inc. and/or its subsidiaries and/or its affiliates and/or their licensors. Other company and product names mentioned herein may be trademarks of their respective owners.

Detailed information on trademarks and patents owned by Software AG and/or its subsidiaries is located at <http://documentation.softwareag.com/legal/>.

Use of this software is subject to adherence to Software AG's licensing conditions and terms. These terms are part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

This software may include portions of third-party products. For third-party copyright notices and license terms, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". This document is part of the product documentation, located at <http://documentation.softwareag.com/legal/> and/or in the root installation directory of the licensed product(s).

Table of Contents

About this Guide	7
Document Conventions	7
Documentation Installation	7
Online Information	7
1. Introduction to Mobile Development	9
About Mobile Development	10
Other Resources for Mobile Development	10
2. Introduction to the Mobile Development User Interface	13
Mobile Development Perspective	14
Opening the Mobile Development Perspective	15
Displaying a Mobile Project in the Outline Editor	15
Displaying a Window, View, or Dialog in the Outline Editor	16
Changing How Information is Displayed in the Outline Editor	17
Adding Objects to a Mobile Project	17
Removing Objects from a Mobile Project	18
Setting Properties in the Outline Editor	18
Using Mobile Designer Ant Targets	19
3. Creating and Building a Mobile Application	21
Creating a New Mobile Project	22
Using Mobile Administrator to Manage and Distribute Mobile Applications	24
Building the User Interface	25
Generating Sources for a Mobile Project	25
Java Sources that Mobile Development Generates	26
Text Resources that Mobile Development Creates for a Project	30
Adding the Mobile Application Logic	31
Defining Resources for the Mobile Project	31
Using the Default Resource Handler	32
Storing Resource Files for the Mobile Project	32
Storing Image Files for the UniversalResHandler	33
Extending the UniversalResHandler to Allow Storing Image Files in Custom Subfolders	35
Coding a Custom Resource Handler	38
Adding Devices to the Mobile Project	39
Removing Devices from the Mobile Project	39
Compiling Resources for a Device	40
Configuring the Orientations Setting for the Application	40
Managing Languages the Application Supports	41
Setting the Default Language for the Project	42
Specifying Values for Non-Default Language Text Resources	43

Adding Services to a Mobile Project	43
Generating and Building a Mobile Project	45
4. Building the User Interface for a Mobile Application	47
Basic Structure of the Application User Interface	48
Defining Panes for the Application Window	49
Adding Views to the Application's User Interface	53
Renaming a View	54
Adding Content to a View	55
Programmatically Populating a ListView	57
Using a Content Provider to Populate a ListView	60
About User-Initiated Events and Listeners	64
Adding Listeners for User-Initiated Events	65
Defining Dialogs	67
Using Templates to Define Custom Objects for a Mobile Project	68
Creating a Template for a Custom Object	68
Using a Template in the Mobile Application User Interface	70
5. User Interface Object Reference	71
User Interface Objects	72
Application Node Properties	72
Objects to Use for Windows	73
Window Properties	74
Objects to Use for Panes	74
HorizontalSplitter Properties	75
PaneConfiguration Properties	75
PaneDefinition Properties	76
VerticalSplitter Properties	76
Objects to Use for Views	76
ListView Properties	77
NavigationView Properties	79
View Properties	79
WebView Properties	80
Objects to Use for the Layout of the User Interface	81
Group Properties	81
Separator Properties	82
Spacer Properties	82
Objects to Use for Dialogs	82
AlertDialog Properties	83
AlertDialogButton Properties	83
Objects to Use for Tables	84
DynamicTablecell Properties	85
DynamicTablerow Properties	85
Table Properties	85
TableButton Properties	86

Tablecell Properties	86
Tablerow Properties	87
Objects to Use for User Interface Controls	87
Button Properties	90
ButtonGroup Properties	91
CheckBox Properties	91
Container Properties	92
DateEntry Properties	93
DropDownListEntry Properties	93
DynamicDisplayObject Properties	94
DynamicDisplayObjectArray Properties	94
DynamicDropdownlistEntryItems Properties	94
Entry Properties	95
Image Properties	95
NavButton Properties	96
Pagination Properties	97
ProgressAnim Properties	97
RadioButton Properties	98
SearchEntry Properties	99
StringDropdownlistEntry Properties	99
Textfield Properties	99
WebViewElement Properties	100
Objects to Use for Content Providers	101
DataBinding Properties	102
DynamicDataSource Properties	102
ContentProvider Properties	102
JSONDataSource Properties	103
Objects to Use for Event Listeners	103
Objects to Use for Event Actions	104
Back Properties	106
ChangePaneConfiguration Properties	106
Delegate Properties	107
OpenDialog Properties	107
ToggleVisibility Properties	107
Transition Properties	108
Objects to Use for Templates	108
ListViewElement Properties	109
Template Properties	109
TemplateReference Properties	110
6. Services Object Reference	111
Objects to Use for Services	112
Resources Properties	112
Resource Properties	113
Method Properties	113

Request Properties	114
Parameter Properties	114
Response Properties	115
7. Creating Application Logic	117
About Adding Application Logic	118
About the TransitionStackController	119
Logic for a View	120
Logic for a Dialog	121
Logic to Display and Close a Dialog	122
Logic for a Method Name Property	122
Logic to Programmatically Set a Property Value at Run Time	123
Logic to Respond to a Listener Event	124
Logic to Transition to Another View	125
Common Methods to Override in Generated Code for the Application	126
Common Methods to Override in the Generated Code for a View	127
8. Managing a Project	129
Renaming a Mobile Project	130
Renaming the Application	130
Changing the Package Name	131

About this Guide

This document contains information about using the Mobile Development plug-in available within Software AG Designer.

Document Conventions

Convention	Description
Bold	Identifies elements on a screen.
Narrowfont	Identifies storage locations for services on webMethods Integration Server, using the convention <i>folder.subfolder:service</i> .
UPPERCASE	Identifies keyboard keys. Keys you must press simultaneously are joined with a plus sign (+).
<i>Italic</i>	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 (...).

Documentation Installation

You can download the product documentation using the Software AG Installer. The documentation is downloaded to a central directory named `_documentation` in the main installation directory (SoftwareAG by default).

Online Information

You can find additional information about Software AG products at the locations listed below.

If you want to...	Go to...
<p>Access the latest version of product documentation.</p>	<p>Software AG Documentation website http://documentation.softwareag.com</p>
<p>Find information about product releases and tools that you can use to resolve problems.</p> <p>See the Knowledge Center to:</p> <ul style="list-style-type: none"> ■ Read technical articles and papers. ■ Download fixes and service packs (9.0 SP1 and earlier). ■ Learn about critical alerts. <p>See the Products area to:</p> <ul style="list-style-type: none"> ■ Download products. ■ Download certified samples. ■ Get information about product availability. ■ Access older versions of product documentation. ■ Submit feature/enhancement requests. 	<p>Empower Product Support website https://empower.softwareag.com</p>
<ul style="list-style-type: none"> ■ Access additional articles, demos, and tutorials. ■ Obtain technical information, useful resources, and online discussion forums, moderated by Software AG professionals, to help you do more with Software AG technology. ■ Use the online discussion forums to exchange best practices and chat with other experts. ■ Expand your knowledge about product documentation, code samples, articles, online seminars, and tutorials. ■ Link to external websites that discuss open standards and many web technology topics. ■ See how other customers are streamlining their operations with technology from Software AG. 	<p>Software AG Developer Community for webMethods http://communities.softwareag.com/</p>

1 Introduction to Mobile Development

■ About Mobile Development	10
■ Other Resources for Mobile Development	10

About Mobile Development

Software AG Designer provides a set of Mobile Development features that you can use to develop mobile applications. Use the Mobile Development perspective to display the views and editors needed to work with mobile applications.

Mobile Development uses the principles of *model view controller (MVC)* architecture, which separates the user interface from the business logic and data.

When using Mobile Development, you define the user interface in the Outline Editor that Mobile Development provides. In the Outline Editor you can also define additional information for the mobile project, such as the languages the application supports or to identify services that your application uses to obtain data.

Mobile Development generates Java code for the application. The generated code is based on the project you define in the Outline Editor. When generating the code, Mobile Development maintains code that it generates separate from the business logic code that you provide. Mobile Development generates code, for example, that displays the user interface you design and that can respond to user-initiated events, such as when a user presses a button.

Mobile Development also creates Java classes where you put your business logic. These Java classes are placed in a location known as the *user space*. After initially creating the Java classes, in the user space, Mobile Development does not make any further changes to the classes so that any code you add is not overwritten or changed.

Other Resources for Mobile Development

In addition to the information contained in the Mobile Development online help topics, you can also find information about working with mobile applications in the following locations:

- *Using webMethods Mobile Designer*. This publication describes how to:
 - Set up your environment for various mobile platforms (for example, Android, iOS, and Windows Phone) so that you can develop mobile applications for a platform.
 - Code resource handlers and mobile applications.
 - Build mobile applications.
 - Install applications on various platforms.
- *webMethods Mobile Designer Native User Interface Reference*. This publication provides general information about how to build the user interface for a mobile application. Additionally, it provides details about the Mobile Designer native user interface that you can use to create user interfaces for mobile applications. The native user interface objects described in this publications correlate to the user interface elements you can add to a mobile application using the Outline Editor in the Mobile Development perspective.

- *webMethods Mobile Designer Java API Reference*. This publication describes the Java classes that Mobile Designer provides and that you can use when coding mobile applications.
- *webMethods Mobile Administrator API Reference*. This reference provides information about how you can access webMethods Mobile Administrator through the REST API.

These resources are available in the `_documentation` directory of your Software AG installation. The `_documentation` directory is installed as a separate component in the Software AG Installer and might not be present. You can install the directory using the Installer or you can download these files from the from the public Software AG Documentation website, <http://documentation.softwareag.com>, or the Empower Product Support Website at <https://empower.softwareag.com> (login required).

2 Introduction to the Mobile Development User Interface

■ Mobile Development Perspective	14
■ Opening the Mobile Development Perspective	15
■ Displaying a Mobile Project in the Outline Editor	15
■ Displaying a Window, View, or Dialog in the Outline Editor	16
■ Changing How Information is Displayed in the Outline Editor	17
■ Adding Objects to a Mobile Project	17
■ Removing Objects from a Mobile Project	18
■ Setting Properties in the Outline Editor	18
■ Using Mobile Designer Ant Targets	19

Mobile Development Perspective

The Mobile Development perspective contains the views and editors needed to work with mobile applications.

- Package Explorer** The Package Explorer is a standard Eclipse view. It shows a Java-specific view of your projects, including mobile projects.
- The Package Explorer tree structure contains a top-level node for each mobile project. The name of the top-level node matches the name of the mobile project.
- The Package Explorer allows access to all information in a project, including the application code, resource handler code, application resources, properties files, and information about the devices a project supports.
- Mobile Explorer view** The Mobile Explorer view is a Mobile Development-specific view. It contains a subset of the information in the Package Explorer. Mobile Explorer view displays information *only* for mobile projects. For each mobile project, the Mobile Explorer view displays:
- Root application
 - Single, main window for the project
 - Each view defined for the project
 - Each dialog defined for the project
- Use the Mobile Explorer view to navigate to the project information you want to view and work with in the Outline Editor. For example, if you want to work on one of the views in the project, you can navigate to that view in the Mobile Explorer view and display it in the Outline Editor so that you can edit the view.
- Outline Editor** The Outline Editor is a Mobile Development-specific editor that shows an outline of a mobile project. The Outline Editor has two sections:
- **Model** section, which displays the tree structure, or outline, of the mobile project. It lists, for example, the window, views, and dialogs in the project. Child nodes of a window, view, or dialog lists the user interface elements, such as buttons or text entry fields, that the window, view, or dialog contains. Additionally, the outline of the project lists the languages that the project supports.
 - **Properties** section, which displays the properties for the node that is selected in the **Model** section of the Outline Editor. Use the **Properties** section to view and edit properties.

Ant view

The Ant view is a standard Eclipse view. It shows Ant scripts that you can use for a mobile project.

Mobile Designer provides several Ant scripts that you use to perform various tasks for a mobile project. For example, you use an Ant task to build a project. The build Ant tasks compile your application code and package the application so that you can install it on a mobile device.

Opening the Mobile Development Perspective

Software AG Designer provides a Mobile Development perspective that contains the views and editors needed to work with mobile applications.

To open the Mobile Development perspective

- 1 In Software AG Designer, select **Window > Open Perspective > Other**.
- 2 In the Open Perspective dialog box, select **Mobile Development**.
- 3 Click **OK**.

Software AG Designer switches to the Mobile Development perspective.

Displaying a Mobile Project in the Outline Editor

You can use the Outline Editor to view an outline structure of your mobile project, update the user interface for the application, and specify languages that your application supports.

You can open a mobile project from the Mobile Explorer view or the Package Explorer.

To display a mobile project in the Outline Editor

- To display a mobile project from the Mobile Explorer view:
 - a Expand the project in the Mobile Explorer view.

The top-level child node of the project represents the root application for the project.
 - b Either double-click the root application node or select the root application node and press **ENTER**.
- To open a mobile project from the Package Explorer.

- a Locate the project in the Package Explorer.
- b Expand the project to locate the root application node in the model folder, for example, `model > application_name.aml`, where `application_name.aml` is the node that represents the root application for the project.
- c Do one of the following to open the project in the Outline Editor:
 - Double-click the root application node.
 - Select the root application node and press ENTER.
 - Right-click the root application node and select `Open With > Mobile Application Editor`.

Displaying a Window, View, or Dialog in the Outline Editor

If you want to work on a mobile project's main window or work on a specific view or dialog in a mobile project, you can display information for that window, view, or dialog in the Outline Editor. By doing so, you can concentrate on just the single item on which you want to work rather than displaying the entire mobile project in the Outline Editor.

You can open a mobile project from the Outline Editor or the Mobile Explorer view.

To display a single window, view, or dialog in the Outline Editor

- From the Outline Editor:
 - a Expand the outline to locate the window, view, or dialog with which you want to work.

Note: If you cannot locate the element (window, view, dialog) in the outline, the Outline Editor might be displaying only a portion of the project that does not include the element you want. In this case, use the instructions below to display the window, view, or dialog from the Mobile Explorer view.



- b Double-click the node for the window, view, or dialog.
- From the Mobile Explorer view:
 - a Expand the project and locate the window, view, or dialog with which you want to work.
 - b Either double-click the node for the window, view, or dialog or select the node and press ENTER.

Changing How Information is Displayed in the Outline Editor

The Outline Editor contains the **Model** section that displays the tree structure (or outline) of the project, and the **Properties** section that displays the properties for the node that is selected in the project's outline. You can display the **Model** and **Properties** sections in the following orientations:

- Horizontally, one on top of the other
- Vertically, side-by-side

To change how the sections are displayed in the Outline Editor

- To display the **Model** and **Properties** sections horizontally, one on top of the other, click  **Horizontal orientation**. This tool is located in the upper, right of the Outline Editor.
- To display the **Model** and **Properties** sections vertically, side-by-side, click  **Vertical orientation**. This tool is located in the upper, right of the Outline Editor.

Adding Objects to a Mobile Project

In the Outline Editor, you add the following types of objects to a mobile project.

- User interface objects, for example:
 - Views and dialogs
 - User interface controls, such as, buttons, check boxes, tables, search fields, and text entry fields,
- Languages that the application supports
- Services that you want to use in your mobile application. For example, you might to add a service that you use to obtain data that your application displays.

To add objects to a mobile project

- 1 Ensure the mobile project or specific window, view, or dialog to which you want to add an item is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#) or [“Displaying a Window, View, or Dialog in the Outline Editor” on page 16](#).
- 2 In the **Model** section of the Outline Editor, expand the outline so that you view the parent node where you want to add a child object.
- 3 To add a child object, right-click the parent node and select **New Child > *child_object***, where *child_object* is the name of the child object you want to add.

The **New Child** list contains only objects that are valid children of the selected parent node.

Tip! After adding a new node, you can edit the properties for the new node. For more information, see [“Setting Properties in the Outline Editor” on page 18](#).

To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Removing Objects from a Mobile Project

In the Outline Editor, you can remove objects from a mobile project.

To remove objects from a mobile project

- 1 Ensure the mobile project or specific window, view, or dialog from which you want to remove an item is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#) or [“Displaying a Window, View, or Dialog in the Outline Editor” on page 16](#).
- 2 In the **Model** section of the Outline Editor, expand the outline so that you can view the node you want to remove.
- 3 To remove a node, right-click the node and select **Delete**.
Alternatively, you can select the node and press the **DELETE** key.

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Setting Properties in the Outline Editor


After you add a new node to the outline, you should set properties for the new node. Additionally, you can update the properties later if you need to change the settings.

To set the properties for a node

- 1 Ensure the mobile project or specific window, view, or dialog for which you want to work with properties is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#) or [“Displaying a Window, View, or Dialog in the Outline Editor” on page 16](#).
- 2 In the **Model** section of the Outline Editor, expand the outline so that you view the node for which you want to set properties.
- 3 Select the node for which you want to set properties.

- 4 In the **Properties** section of the Outline Editor, fill in the properties for the selected node.

For more information about the properties, see [“User Interface Objects” on page 72](#).

Note: If the  **Content Assist Available** icon is displayed next to a field, click into the field and press CTRL+SPACE to view the types of information you can specify for a property. The content assist shows valid values and/or syntax you can use to specify a valid value. If the content assist lists `@{myMethodName}` or `@{my.package.class.static.method}`, you can specify the name of a method to execute at run time to supply the value for the property. For more information, see [“Logic to Programmatically Set a Property Value at Run Time” on page 123](#).

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Using Mobile Designer Ant Targets

Mobile Designer provides several Ant targets that you use to build and run your project in the Mobile Designer utility, *Phoney*, which is a phone simulator that is not platform-specific that you can use to test your applications.

For more information about the Ant targets that are available and the actions the Ant targets perform, see *Using webMethods Mobile Designer*.

To run an Ant target for a mobile project

- 1 In the Project Explorer, locate the project for which you want to execute an Ant target.
- 2 Expand the project and drag its build.xml file to the Ant view.
- 3 In the Ant view, double-click the Ant target that you want to run.

3 Creating and Building a Mobile Application

■ Creating a New Mobile Project	22
■ Using Mobile Administrator to Manage and Distribute Mobile Applications	24
■ Building the User Interface	25
■ Generating Sources for a Mobile Project	25
■ Text Resources that Mobile Development Creates for a Project	30
■ Adding the Mobile Application Logic	31
■ Defining Resources for the Mobile Project	31
■ Adding Devices to the Mobile Project	39
■ Removing Devices from the Mobile Project	39
■ Compiling Resources for a Device	40
■ Configuring the Orientations Setting for the Application	40
■ Managing Languages the Application Supports	41
■ Setting the Default Language for the Project	42
■ Specifying Values for Non-Default Language Text Resources	43
■ Adding Services to a Mobile Project	43
■ Generating and Building a Mobile Project	45

Creating a New Mobile Project

Mobile Development provides the New Mobile Development Project wizard that you can use to create a new mobile project. When you create a mobile project, Mobile Development automatically adds the following to your mobile project:

- Adds your system language as a language that your application supports. You can add additional languages and/or remove languages after the project is created. For more information, see [“Managing Languages the Application Supports” on page 41](#).
- Adds several universal devices that your application supports. You can add additional devices and/or remove devices after the project is created. For more information, see [“Adding Devices to the Mobile Project” on page 39](#) and [“Removing Devices from the Mobile Project” on page 39](#).

Additionally, when you create a project, in the New Mobile Development Project wizard, you can provide settings to use Mobile Administrator with your project. Before you can use Mobile Administrator, you must perform required setup. For more information, see [“Using Mobile Administrator to Manage and Distribute Mobile Applications” on page 24](#).

To create a new mobile project

- 1 Open the New Mobile Development Project wizard by selecting one of the following:
 - File > New > Mobile Project
 - File > New > Project > Software AG > Mobile Development > Mobile Project
- 2 Click Next.
- 3 Specify the following settings for the mobile project:
 - a In the Project Name field, type a name for the new project.
 - b In the Application Name field, type the name you want to assign the application you are creating.

Mobile Development uses the name you specify internally, for example, as part of the name of the `application_nameAppControllerImpl.java` Java class that it creates.

Note: You can rename the application at a later time. However, if you have added custom code to `application_nameAppControllerImpl.java` you need to take further actions. For more information, see [“Renaming the Application” on page 130](#).

- c In the Package Name field, type a package name. When specifying the name, be sure to only use characters valid in a Java package name.

Mobile Development uses the name you supply as part of package names for Java classes in the `gen/src` and `src` folders of your project. For example, if you specify `com.mycompany.myproject`, the `gen/src` folder contains the `com.mycompany.myproject` package.

- d Indicate whether you want to save the project in the default location:
 - To use the default location, select the **Use default location** check box.
 - To specify an alternate location, clear the **Use the default location** check box, click **Browse**, and browse to and select the location where you want to save the project.
- e Click **Next**.
- 4 If you want to manage and distribute your application via webMethods Mobile Administrator, continue with the next step. Otherwise, click **Finish**.
- 5 If you want to use Mobile Administrator to manage and distribute your application, do the following.

Important! To use Mobile Administrator for a mobile project, you must perform setup in Mobile Administrator. For more information, see [“Using Mobile Administrator to Manage and Distribute Mobile Applications”](#) on page 24.

- a Select the **Use Mobile Administrator** check box.
- b In the **URL** field, type the Mobile Administrator instance you want to use to manage and distribute the application.
- c In the **Access Token** field, specify 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.
- d Click **Login**.
- e For **Application** do one of the following:
 - If you want to use an existing Mobile Administrator application, select **Existing**.
 - If the Mobile Administrator application you want to use does not yet exist and you plan to create a new Mobile Administrator application, select **New**.
- 6 Complete the Mobile Administrator information based on whether you are using an existing application or will be creating a new application:
 - If you are using an existing Mobile Administrator application, select the identifier of the application you want to use.
 - If you will be creating a new Mobile Administrator application, perform the following steps:

- i In the **Identifier** field, type the name of the identifier you want to use for a new project.
 - ii For an Android project, select the **Android** check box and in the **Bundle ID** field specify the bundle ID you want to use for the final binary.
 - iii For an iOS project, select the **iOS** check box and in the **Bundle ID** field specify the bundle ID you want to use for the final binary.
- 7 Click **Finish**.

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 Designer provides a Mobile Administrator plug-in that allows you to use Mobile Administrator for applications you create using Mobile Development.

Before You Can Configure a Mobile Project to Use Mobile Administrator

Before you can configure a mobile project to use the Mobile Administrator, perform the following required setup in Mobile Administrator.

- Create a Mobile Administrator project that you will associate with your mobile project. You need one Mobile Administrator project for each mobile project.
- Use an existing Mobile Administrator user account or define a new one, and assign the user account to the Mobile Administrator project.
- Ensure the Mobile Administrator user account, at a minimum, has the following permissions for the Mobile Administrator project:
 - View and Download Stable Versions
 - Manage Build Jobs
- Ensure the Mobile Administrator user account has the global **Can Manage Site** permission.

You can set this permission in Mobile Administrator on the **Details** tab for a user.

- Determine whether you want to use an access token or basic authentication with the Mobile Administrator user account.

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.

Configuring a Mobile Project to Use Mobile Administrator

You can configure a mobile project to use Mobile Administrator when you initially create the project using the New Mobile Development Project wizard. When you specify information for Mobile Administrator in the wizard, Mobile Development performs all the necessary configuration tasks for your mobile project.

If you do not specify information for Mobile Administrator in the New Mobile Development Project wizard and later decide you want to use Mobile Administrator for your project, you must configure the mobile project manually. For information about how to manually set up your mobile project to use the Mobile Administrator plug-in, see information about using Mobile Administrator to distribute mobile applications in *Using webMethods Mobile Designer*.

Building the User Interface

The following lists the tasks to perform to build the user interface for a mobile application.

- Understand the basic structure of the user interface, for information, see [“Basic Structure of the Application User Interface” on page 48](#).
- Define the configuration of panes to use for the application’s window. For more information, see [“Defining Panes for the Application Window” on page 49](#).
- Define the different screens that the application displays. The screens are referred to as *views*. For more information, see [“Adding Views to the Application’s User Interface” on page 53](#) and [“Adding Content to a View” on page 55](#).
- Add listeners that wait for user-initiated events when a user interacts with controls you add to the view and take an action based on the user-initiated event. For more information, see [“About User-Initiated Events and Listeners” on page 64](#) and [“Adding Listeners for User-Initiated Events” on page 65](#).
- Define templates if you want to customize and reuse user interface structures. For more information, see [“Using Templates to Define Custom Objects for a Mobile Project” on page 68](#).

Generating Sources for a Mobile Project

To incorporate the changes you make to the mobile project’s model you need to generate the sources. When you generate sources, Mobile Development generates Java classes for the mobile application. For a description of the Java classes that Mobile Development generates, see [“Java Sources that Mobile Development Generates” on page 26](#).

You should generate sources after you update a project, for example, by adding additional user interface objects to the project. You can generate sources from the Outline Editor or the Package Explorer.

Note: Mobile Development also create .txt files for each language that the mobile project supports. For more information, see [“Text Resources that Mobile Development Creates for a Project”](#) on page 30.

To generate sources

- From the Outline Editor:
 - a Display your project in the Outline Editor. For more information, see [“Displaying a Mobile Project in the Outline Editor”](#) on page 15.
 - b In the Outline Editor, right click anywhere and select **Generate Source Code > Application Model**.
- From the Package Explorer:
 - a Locate the project in the Package Explorer.
 - b Right-click the project node or any file in the project and select **Generate Source Code > Application Model**.

Note: Selecting **Generate Source Code > Application Model** generates the source code for the mobile project based on the model you define in the Outline Editor. If you select **Generate Source Code > Application Model and API**, Mobile Development also generates the Mobile Development API in the `gen/api-src` folder.

Java Sources that Mobile Development Generates

When you generate sources for your mobile project by using **Generate Source Code > Application Model** in the Outline Editor, Mobile Development generates Java classes in the following folders:

- **gen/src folder** contains Java classes that are specific to your mobile project and are based on the model you develop in the Outline Editor.

All the Java in the `gen/src` folder is generated. Mobile Development regenerates the Java classes in this folder each time you generate sources for your mobile project. As a result, the Java classes reflect the changes you make to your model, for example, if you add or remove user interface objects.

- **src folder**, also known as the *user space*, contains Java classes that you update to provide the business logic for your application.

Mobile Development generates each Java class in the `src` folder *only* one time. If the class already exists when you generate sources, Mobile Development does not overwrite it. Additionally, to preserve logic you might have added to generated logic,

Mobile Development also does *not* delete the Java classes, for example, if you rename or delete a corresponding item in the model. You must delete unneeded Java classes manually for the project to compile.

Note: If you use **Generate Source Code > Application Model and API**, Mobile Development also creates Java classes in the `gen/api-src` folder. The names of the Java packages in this folder start with `com.software.mobile.runtime.toolkit`. These packages contain Java classes for the Mobile Development API.

Caution! Do not make changes to the Java classes in the `gen/src` or `gen/api-src` folders. These folders contain classes that Mobile Development automatically generates and changes you make will be lost.

Model-Specific Java Code in the `gen/src` Folder

When you generate sources for your mobile project, Mobile Development generates the following packages based on the model that you defined in the Outline Editor. In the names of the following packages, *package_name* is the package name that you specified for your mobile project.

Package Name in the <code>gen/src</code> folder	Description
<i>package_name</i>	This package contains general model-based Java classes.
<i>package_name.i18n</i>	This package contains Java classes for language support to load languages that you indicated your mobile application supports. You specify languages your application supports by adding the languages to the model. For more information, see “Managing Languages the Application Supports” on page 41.
<i>package_name.services.rest</i>	This package contains Java classes that correspond to the services that you add to your mobile project in the Outline Editor. For more information, see “Adding Services to a Mobile Project” on page 43.
<i>package_name.ui</i>	This package contains Java classes that correspond to the user interface that you designed in the Outline Editor. This includes Java classes for each view in your user interface along with its associated abstract controller.
<i>package_name.ui.dialog</i>	This package contains Java classes that correspond to the dialogs that you designed in the Outline Editor.

<u>Package Name in the gen/src folder</u>	<u>Description</u>
<i>package_name</i> .ui.templates	This package contains Java classes that correspond to the templates you defined in the Outline Editor, if any. For more information about using templates, see “Using Templates to Define Custom Objects for a Mobile Project” on page 68.
<i>package_name</i> .utils	This package contains a helper class that provides services, such as, determining whether the application is running on a tablet or the orientation of the device, whether portrait or landscape.

Model-Specific Java Code in the src Folder

When you generate sources for your project, Mobile Development generates the following packages based on the model that you defined in the Outline Editor. In the names of the following packages, *package_name* is the package name that you specified for your mobile project.

Package Name in the src folder	Description
<i>package_name</i> .ui.controller.impl	<p>This package contains Java classes that correspond to the user interface that you designed in the Outline Editor. Mobile Development generates the classes a single time. You add your application logic to these Java classes. The Java classes in this package are:</p> <ul style="list-style-type: none"> <li data-bbox="695 512 1192 541">■ <i>application_name</i>AppControllerImpl.java <p data-bbox="740 562 1393 735">In the name of the Java class, <i>application_name</i> is the name you assigned to the application. Mobile Development generates one <i>application_name</i>AppControllerImpl.java class for your mobile project.</p> <p data-bbox="740 756 1382 928">Add the logic to this Java class that you want the application to execute when the application starts and when the user rotates the device, changing its orientation. This is also a good location for code that is not related to a specific view.</p> <li data-bbox="695 949 1073 978">■ <i>view_name</i>ControllerImpl.java <p data-bbox="740 999 1393 1171">In the name of the Java class, <i>view_name</i> is the name of a view you defined in the Outline Editor. Mobile Development generates one <i>view_name</i>ControllerImpl.java class for each view in your model.</p> <p data-bbox="740 1192 1382 1398">Add logic specific to a view to this Java class. You can add custom code here that extends the generated abstract view controller methods that Mobile Development generates in the <i>Abstractview_name</i>Controller.java files, which reside in the gen/src folder in the <i>package_name</i>.ui package.</p> <p data-bbox="695 1419 1373 1514">For more information about the types of logic to add these Java classes, see “About Adding Application Logic” on page 118</p>

<u>Package Name in the src folder</u>	<u>Description</u>
<i>package_name.ui.dialog</i>	<p>This package contains Java classes that correspond to the dialogs that you added to the user interface in the Outline Editor. Mobile Development generates the classes a single time.</p> <p>For each dialog you define in the Outline Editor, Mobile Development generates a <i>dialog_name.java</i> class, where <i>dialog_name</i> is the name you assigned the dialog in the Outline Editor.</p> <p>Mobile Development generates the Java classes for dialogs a single time. You add logic to customize the user interface object to the generated Java classes.</p>
<i>package_name.ui.templates</i>	<p>This package contains Java classes that correspond to the templates you defined in the Outline Editor. You use templates to customize user interface objects that Mobile Development provides. For more information, see “Using Templates to Define Custom Objects for a Mobile Project” on page 68.</p> <p>For each template you define in the Outline Editor, Mobile Development generates a <i>template_name.java</i> class, where <i>template_name</i> is the name you assigned the template in the Outline Editor.</p> <p>Mobile Development generates the Java classes for templates a single time. You can add your logic for the dialogs to these Java classes. For more information, see “Creating a Template for a Custom Object” on page 68.</p>

Text Resources that Mobile Development Creates for a Project

In addition to generated Java source, Mobile Development generates .txt files for each language that the mobile project supports.

When you create the project using the New Mobile Development Project wizard, as described in [“Creating a New Mobile Project” on page 22](#), Mobile Development generates .txt files in the project’s resources/text folder. Mobile Development updates the .txt files each time you save the project. Mobile Development generates one .txt file for each language that the mobile project supports. The following shows the naming convention for the .txt files:

core.language_code.txt

The *language_code* in the file name corresponds to the language code you specified for the **Short Name** property when you added the language to the mobile project.

Each `core.language_code.txt` file contains lines for the text strings that you use in a mobile project's views. For example, the file contains a line for the view's Header Text property. If you add a Textfield object to a view, the file contains a line for the Textfield object's Text property.

Note: Mobile Development only creates text resource entries for properties that take a text string for a value *when* you provide a value for the property. Additionally, you must name the element to which the property belongs. You name a property using the element's Name property. If you specify a value for a property, but do not name the element, Mobile Development generates the plain String value instead of creating a reference to the text resource.

When generating the `core.language_code.txt` files, Mobile Development only includes the values in the .txt file that is associated with the project's default language. For example, if the default language uses the language code "en", the `core.en.txt` file might have the following line:

```
MASTERVIEW_HEADER_TEXT=Master View
```

If a mobile project also includes a language with the language code "de", but "de" is *not* the default language, the corresponding line in the `core.de.txt` file is:

```
MASTERVIEW_HEADER_TEXT=
```

It is your responsibility to provide the appropriate translations for the strings for the `core.language_code.txt` files of the languages that are not the default language.

Adding the Mobile Application Logic

To add the business logic for your application, add your custom code to the *user space*, that is, the Java classes that Mobile Development generates in the mobile project's src folder.

Caution! Do not add logic to the Java classes in the `gen/src` or `gen/api-src` folders. When you generate sources or when you generate sources and API for a mobile project, Mobile Development regenerates all the Java classes in those folders. Changes you make will be lost.

For more information, see ["About Adding Application Logic" on page 118](#).

Defining Resources for the Mobile Project

Each project requires its own resource handler. The resource handler defines all the resources to include with your mobile application, such as graphics, text, icons, and sounds. You can use either the default resource handler that Mobile Development provides or code your own resource handler.

Using the Default Resource Handler

Mobile Development provides a default resource handler named `package_name.UniversalResHandler.java`, which is in the project's reshandler folder.

If you want to use the resource handler, you do not need to add any code to `UniversalResHandler`.

To use the default resource handler

- 1 Ensure the settings for your mobile project are set to use the default resource handler.
 - a Ensure the mobile project is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
 - b Select the top-level child node of the project, which is the root application node.
 - c Ensure the **Res Handler** property is set to `UniversalResHandler`.
- 2 Save the resources for your mobile application, for example, audio files and icons. For instructions, see [“Storing Resource Files for the Mobile Project” on page 32](#) and [“Storing Image Files for the UniversalResHandler” on page 33](#).

Storing Resource Files for the Mobile Project

Whether you use the default resource handler that Mobile Development provides or a custom resource handler you code, you need to save the files that contain the resources that your application uses in your project.

Store your resource files within the subfolders of the mobile project's resources folder. The resources folder has subfolders for the different types of resources.

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

```
MyProject
  resources
    graphics
    icons
    text
    www
```

If you want to use a different subfolder than the ones provided, for example, if you want to use a subfolder named `audio` to save sound files, add the custom subfolders to the resource folder and code a custom resource handler for your application. For more information about using custom resource handlers, see [“Coding a Custom Resource Handler” on page 38](#).

If you are using the default, `UniversalResHandler` resource handler, the following table describes the types of assets you should save in each of the resources subfolders.

resources subfolder	Store this type of assets in the subfolder
graphics	Image files that the mobile application uses. These are image files that are larger than an icon. The UniversalResHandler resource handler requires a specific folder structure under the resources/graphics folder. For more information, see “Storing Image Files for the UniversalResHandler” on page 33 .
icons	Small image files that the mobile application uses as icons.
text	Text files that contain Strings that the mobile application uses.
www	HTML files that contain web content that the mobile application uses.

Storing Image Files for the UniversalResHandler

When using the UniversalResHandler resource handler, you need to store image files in a specific folder structure. The required folder structure is automatically created for a project when you create the project using the Mobile Development New Mobile Development Project wizard, as described in [“Creating a New Mobile Project” on page 22](#).

The structure allows you to supply different image files for different platforms. For example, if a mobile application uses an image file named myimage.png, you might need one version of the image file for an Android device and a different version for an iOS device. The folder structure allows you to save both, and at run time the application selects the correct image file based on the device on which the mobile application is running.

The folder structure has four platform-specific folders and a single general-purpose folder. Use the general-purpose folder to save image files that can be used for devices running on any platform. The following shows an example for a project named "MyProject":

```
MyProject
  resources
    graphics
      Android
      BlackBerry
      general
      iOS
      WinPhone
    icons
    text
    www
```

Additionally, each of the main image folders (Android, BlackBerry, iOS, WinPhone, and general) contain subfolders themselves. This allows you to supply different image files for different devices within a platform. For example, if a mobile application uses an

image file named `myimage.png`, you might need one version of the image file for an iOS non-retina device and a different version for an iOS retina device. The following table describes the subfolder structure for each main image folder:

Main image folder	Description of its subfolder structure
Android	<p>The subfolders are based on screen density of Android device on which the mobile application runs. The following shows the subfolder structure:</p> <pre> Android drawable-hdpi drawable-ldpi drawable-mdpi drawable-xhdpi drawable-xxhdpi </pre>
BlackBerry	<p>The subfolders are based on the screen width (in pixels) of the BlackBerry device on which the mobile application runs. The following shows the subfolder structure:</p> <pre> BlackBerry w160 w240 w320 w360 w480 </pre>
iOS	<p>The subfolders are based on the display property of the iOS device, either non-retina or retina. The following shows the subfolder structure:</p> <pre> iOS NonRetina Retina </pre>
WinPhone	<p>The subfolders are based on the screen resolution of the Windows Phone device and whether background color of the view in which the image is displayed is dark or light. The following shows the subfolder structure:</p> <pre> WinPhone Dark 1080p 720p WVGA WXGA Light 1080p 720p WVGA WXGA </pre>

Main image folder	Description of its subfolder structure
general	<p>The subfolders are based on the width (in pixels) of the container in which the image is displayed at run time. For example, the container might a view or table cell. The following shows the subfolder structure:</p> <pre> general w200 w400 w600 w800 </pre>

At run time, the application searches the folders in the following order to locate the version of the image to use:

- 1 **Platform-specific subfolder.** For example, if the application is running on an iOS retina device, the application first attempts to locate the image in the `resources/graphics/iOS/Retina/` folder.
- 2 **Platform-specific root folder.** Continuing with the example, if the image was not found in the platform-specific subfolder, the application next looks for the image in the platform-specific root folder, which is `resources/graphics/iOS/` folder.
- 3 **General subfolder.** Continuing with the example, if the image was not found in the platform-specific root folder, the application next looks for the image in the appropriate general subfolder. For example, if the image is to be displayed in a table cell that is 150 pixels, the application looks for the image file in the `resources/graphics/general/w200/` folder.
- 4 **General folder.** Continuing with the example, if the image was not found in the general subfolder, the application next looks for the image in the general root folder, which is `resources/graphics/general/` folder.

After searching for the image file, if the application does not locate an image file to use, it returns a placeholder image file. The placeholder image file is a point with size 1x1 pixel.

Extending the UniversalResHandler to Allow Storing Image Files in Custom Subfolders

If you want to use the basic functionality of the `UniversalResHandler` resource handler, but want to use additional subfolders for the image files, you can create a custom resource handler that extends the `UniversalResHandler` resource handler. For example, suppose in addition to the standard iOS platform subfolders, which are `NonRetina` and `Retina`, you also want to use `RetinaliPhone4` and `RetinaliPhone5`. In this case, you can create a custom resource handler that extends the `UniversalResHandler` resource handler and includes the logic to handle the new subfolders.

To extend the UniversalResHandler to support additional image subfolders

- 1 Create a new java class, for example, MyUniversalResHandler, in the same folder where UniversalResHandler resides.
- 2 In the new java class for the custom resource handler, add your custom logic.

The custom logic should:

- Extend UniversalResHandler.java.
- Perform a super call to provide the behavior of the UniversalResHandler resource handler.
- Determine the type device and if the device is one for which you have a custom folder, provide logic for that custom folder.

The following shows an example resource handler named MyUniversalResHandler that extends the UniversalResHandler resource handler. This custom resource handler accommodates storing image files in the following graphic subfolders. These subfolders are in addition to the subfolders that the UniversalResHandler resource handler supports.

- resources/graphics/iOS/RetinaiPhone4/
- resources/graphics/iOS/RetinaiPhone5/
- resources/graphics/general/w1000/

```
// Use the UniversalResHandler's package
package my_application_package;

public class MyUniversalResHandler extends UniversalResHandler {

    @Override
    public void projectResourceScript() {

        // This call provides the UniversalResHandler behavior as
        // the default behavior.
        super.projectResourceScript();

        rh.setResourceReadSubdirectory("graphics");
        // get current handset name
        String selectedHandset = rh.getProperty("selected.handset");

        if (selectedHandset.startsWith("iOS")) {
            // Logic for iOS devices only

            // Processes the "iOS/RetinaiPhone4" folder to
            // make all images in the "RetinaiPhone4" folder
            // available at the run time.
            addResourceFolder("iOS", "RetinaiPhone4");

            // Processes the "iOS/RetinaiPhone5" folder to
            // make all images in the "RetinaiPhone5" folder
            // available at the run time.
            addResourceFolder("iOS", "RetinaiPhone5");
        }
    }
}
```

```

        // Processes "general/w1000" folder to make all
        // images in the "general/w1000" folder available
        // at the runtime.
        addResourceFolder("general", "w1000", false);
    }
}

```

- 3 Update the mobile project's properties so that the mobile project uses your custom resource handler.
 - a Ensure the mobile project is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor”](#) on page 15.
 - b Select the top-level child node of the project, which is the root application node.
 - c Set the **Res Handler** property to the name of your custom resource handler, *package_name.custom_resource_handler_name.java*, for example *com.softwareag.mobile.myproject.MyUniversalResHandler.java*.
- 4 In the *application_nameAppControllerImpl.java*, override the methods the application uses to obtain image files.

The method you override depends on where you add the new subfolders that need to be searched for image files. The following tables lists the methods to override:

If you add a subfolder to this resources/graphics folder...	Override this method(s)
Android	getAndroidGraphicsFolder
BlackBerry	getBlackBerryGraphicsFolder
iOS	getIOGraphicsFolder
WinPhone	getWinPhoneGraphicFolder and getWinPhoneThemeFolder
general	getGeneralGraphicsFolder

Continuing with the example started in step 1 of this procedure, the following code sample shows how to override the `getIOGraphicsFolder` method so that at run time the application searches following additional folders for image files:

- `resources/graphics/iOS/RetinaIPhone4/`
- `resources/graphics/iOS/RetinaIPhone5/`

```

protected String getIOGraphicsFolders(int currentScreenPPI,
                                     int viewBackgroundColor) {
    int height = Math.max(CanvasController.CURRENT_SCREEN_WIDTH,
                         CanvasController.CURRENT_SCREEN_HEIGHT);
    if (currentScreenPPI >= 200) {
        if (height == 960) {
            return "RetinaIPhone4/"; // !!! Slash at the end is important!!!
        } else if (height == 1136) {
            return "RetinaIPhone5/"; // !!! Slash at the end is important!!!
        }
    }
}

```

```
    }  
  }  
  // otherwise, return default folder  
  return super.getIOSGraphicsFolders(currentScreenPPI, viewBackgroundColor);  
}
```

Again, continuing with the example started in step 1 of this procedure, the following code sample shows how to override the `getGeneralGraphicsFolder` method so that at run time the application searches the `resources/graphics/general/w1000/` folder for image files:

```
protected String getGeneralGraphicsFolder(int viewBackgroundColor,  
                                         int containerWidth) {  
    if ( containerWidth >= 1000) {  
        return "w1000/"; // !!! Slash at the end is important!!!  
    }  
    // otherwise, return default folder  
    return super.getGeneralGraphicsFolder(viewBackgroundColor,  
                                         containerWidth);  
}
```

Coding a Custom Resource Handler

To use a custom resource handler for your mobile application

- 1 Code your resource handler. For information, see information about defining resources for mobile applications in the *Using webMethods Mobile Designer*.

Note: It is recommended that you save your custom resource handler in your mobile project's reshandler folder.

- 2 Ensure the mobile project is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 3 Select the top-level child node of the project, which is the root application node.
- 4 Set the **Res Handler** property to identify the fully-qualified name of the custom resource handler you coded.
- 5 Save the resources for your mobile application, for example, audio files and icons. For instructions, see [“Storing Resource Files for the Mobile Project” on page 32](#).

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Adding Devices to the Mobile Project

When you create a project, Mobile Development adds several universal devices to your project. If needed, you can add additional devices to the mobile project later using the Mobile Designer Add-Handset Ant target. The following describes how to add a device using the Add-Handset Ant target.

Note: If you need to remove devices from your project, see [“Removing Devices from the Mobile Project” on page 39](#).

To add devices to a mobile project using the Add-Handset Ant target

- 1 Open the mobile project in the Outline Editor if it is not already open. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the Project Explorer, expand the project, and drag the build.xml file to the Ant view.
- 3 In the Ant view, double-click the **Add-Handset Ant** target and fill in the required information.

For more information about adding devices to project how to use the Add-Handset Ant target, see *Using webMethods Mobile Designer*.

Removing Devices from the Mobile Project

When you create a project, Mobile Development adds several universal devices to your project. You can add additional devices using the procedure described in [“Adding Devices to the Mobile Project” on page 39](#).

If you later decide you no longer want your application to support a device, you can remove it.

To remove a device from a mobile project

- 1 In the Project Explorer, expand the project so that you can view the project’s targets folder, and expand the targets folder.

The targets folder contains one .xml file for each device the application supports.
- 2 Delete the .xml file that corresponds to the device you want to remove from the mobile project.

Important! Do not remove the `_defaults_.xml` file.

Compiling Resources for a Device

Use the +Run-Reshander Ant target to compile the resources for the current device.

You should compile resources for a device:

- After you change or add language resources, such as text or header text.
- After you change or add new image resources.
- After you add parameters to the `_defaults.xml` file.

Alternatively, you can use the ++Activate-Handset Ant target, which allows you to select the device that you want to activate. For information about using the ++Activate-Handset Ant target, see *Using webMethods Mobile Designer*.

To compile resources for a device

- 1 In the Project Explorer view, expand the mobile project, and drag the build.xml file to the Ant view.
- 2 In the Ant view, double-click Run-Reshander.

The Ant target compiles the resources for the current device.

Configuring the Orientations Setting for the Application

An application's orientation setting indicates whether the user interface for the application displays in portrait mode, landscape mode, or rotates from portrait mode to landscape or vice versa as the user rotates the device.

To configure the orientation setting for a mobile application

- 1 Ensure the mobile project is displayed in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the Model section of the Outline Editor, expand the project so that you can view the top-level child node that represents the root application for the project.
- 3 Select the root application node.
- 4 In the Properties section of the screen, select the orientation you want to use in the Orientation property.

Select PerHandset you want to set the orientation settings for each device a project supports rather than use a single global orientation setting for all devices. When you use PerHandset, the orientation setting for a device is made in the XML file for the device in the project's targets folder. In this case, you are responsible for setting the correct orientation property for each device. For more information about project properties, see *Using webMethods Mobile Designer*.

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Managing Languages the Application Supports

When you create your project, Mobile Development adds your system language to your mobile project as a language your application supports. If needed, you can add or remove languages your application supports.

Note: If you want to change the default language, see [“Setting the Default Language for the Project” on page 42](#)

To add or remove languages that your application supports

- 1 Ensure the mobile project for which you want to manage languages is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the **Model** section of the Outline Editor, expand your mobile project’s **Languages** container node.
- 3 To add a language, do the following:
 - a Right-click the **Languages** container node and select **New Child > Language**.
 - b Select the **Language** node you just added.
 - c In the **Properties** section of the Outline Editor, specify the following properties:

For this property...	Specify...
Directionality	Direction to use for the language. Select one of the following: <ul style="list-style-type: none"> ■ L2R for left-to-right ■ R2L for right-to-left
Short Name	Abbreviation for the language, for example, “en”. Use the two-character language code defined by the ISO-639 standard.

- 4 To remove a language, in the **Model** section of the Outline Editor, right-click the language you want to remove and select **Delete**.

Alternatively, you can select the language and press the **DELETE** key.

Tip! To update the information that Mobile Development generates for the project so that your changes are represented in the generated text resources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Setting the Default Language for the Project

Mobile Development can create localized mobile applications. You designate one language that your project supports as the default. Your application uses the default language when no specific language is selected.

When you generate sources for a mobile project, Mobile Development generates text resource files for the text fields in the mobile project. Although Mobile Development maintains a text resource file for each language in the mobile project, it only includes text values in the text resource file for the default language. For more information, see [“Text Resources that Mobile Development Creates for a Project” on page 30](#). For information about working with languages that are not the default, see [“Specifying Values for Non-Default Language Text Resources” on page 43](#).

Note: When you switch the default language to another language and then generate sources for the project, Mobile Development does not clear or update values in the text resource file of the former default language. Mobile Development *only* updates and/or adds values to the text resource file associated with the default language.

To set the default language for the project

- 1 Ensure the mobile project is displayed in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the **Model** section of the Outline Editor, expand the project so that you can view the top-level child node that represents the root application for the project.
- 3 Select the root application node.
- 4 In the **Properties** section of the screen, select the language you want to use from the list in the **Default Language** property.

This list is populated with all the languages that your application supports. In other words, languages you have added to the **Languages** container node in the model.

Tip! To update the information that Mobile Development generates for the project so that your changes are represented in the generated text resources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Specifying Values for Non-Default Language Text Resources

When generating text resource files for a mobile project, Mobile Development only includes the values for the text strings in the text resource file that is associated with the default language. It is your responsibility to translate the values for other languages and specify the values in the text resource files for those languages. For more information about the text resource files, see [“Text Resources that Mobile Development Creates for a Project” on page 30](#).

To specify values for non-default language text resources

- 1 Locate the project in the Package Explorer.
- 2 Expand the project to locate the `resources/text` folder.
- 3 Expand the text folder.
- 4 Open a `core.language_code.txt` file for a non-default language, where `language_code` is the language code you specified for the **Short Name** property when you added the language to the mobile project.

Tip! You might find it helpful to also open the `core.language_code.txt` file for the default language so that you can see the values you need to translate.

- 5 For each line in the file, fill in the translated value for each text field.

Caution! Do not edit values for the default language in this manner because when you save the mobile project, Mobile Development regenerates the `core.language_code.txt` file for the default languages, and your changes will be lost. To change values for the default language, edit the associated values in the Outline Editor.

- 6 Save the file.

Repeat this procedure for each non-default language that the mobile project supports.

Adding Services to a Mobile Project

You can use REST services as data sources for a mobile application. An application can execute REST services to obtain data to display in the application’s user interface. Because REST services typically return multiple data elements, it is common to use a `ListView` object to display the data you obtain from a REST service. For more information, see [“Using a Content Provider to Populate a ListView” on page 60](#).

To add REST services to a mobile project

- 1 Ensure the mobile project is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the **Model** section of the Outline Editor, expand the project so that you view the **Services** container node.

If the model does *not* have a **Services** container node, add one by right-clicking the root application and selecting **New Child > Services**.
- 3 Right-click the **Services** node and select **New Child > Resources**.
- 4 Select the **Resources** node, and in the **Properties** section of the Outline Editor set the properties for the **Resources** node. For more information, see [“Resources Properties” on page 112](#).
- 5 Right-click the **Resources** node and select **New Child > Resource**.
- 6 Select the **Resource** node, and in the **Properties** section of the Outline Editor set the properties for the **Resource** node. For more information, see [“Resource Properties” on page 113](#).
- 7 Right-click the **Resource** node and select one of the following:
 - **New Child > Method** to specify the service you want to use. The mobile application queries the REST service by calling the method you specify.
 - **New Child > Resource** if you want to add additional **Resource** objects to specify subpaths. If you add another **Resource** node, repeat the previous step to specify the properties for the **Resource** node and this step to add a child node.
- 8 When you add a **Method** child node, select the node, and in the **Properties** section of the Outline Editor set the properties for the **Method** node. For more information, see [“Method Properties” on page 113](#).

Note: Mobile Development automatically adds two child nodes for the **Method** node. The child nodes are **Request** and **Response**.

- 9 Select the **Request** node, and in the **Properties** section of the Outline Editor set the properties for the **Request** node. For more information, see [“Request Properties” on page 114](#).
- 10 If the REST service requires input parameters, perform the following steps for each input parameter:
 - a Right-click the **Request** node and select **New Child > Parameter**.
 - b Select the **Parameter** node, and in the **Properties** section of the Outline Editor set the properties for the **Parameter** node. For more information, see [“Parameter Properties” on page 114](#).

- 11 Select the **Response** node, and in the **Properties** section of the Outline Editor set the properties for the **Response** node. For more information, see “[Response Properties](#)” on [page 115](#).
- 12 If this is the first service you added to the model, add the json library to the project. The json library is required to avoid compile errors. You can obtain the json library in one of the following:
 - Download the source code from <https://github.com/upictec/org.json.me/>.
 - Copy the source code from the `_LibraryJSON_` sample project that is installed with Mobile Designer. You can find the `_LibraryJSON_` sample project in the following location:

Mobile Designer_directory/Samples

Adding the json library is required because when you add a service to a model and generate the source for the application model and API, Mobile Development adds `com.software.mobile.runtime.rest` package in the `src-api` folder. The classes in this package depend on the json library.

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see “[Generating Sources for a Mobile Project](#)” on [page 25](#). If this is the first service you added, use **Generate Source Code > Application Model and API** to generate the `com.software.mobile.runtime.rest` package in the `src-api` folder.

Generating and Building a Mobile Project

To create a build of a mobile project, you can generate project source files and build these source files to create one or more final binaries that are installable on devices.

To generate and build a mobile project

- 1 Open the mobile project in the Outline Editor if it is not already open. For instructions, see “[Displaying a Mobile Project in the Outline Editor](#)” on [page 15](#).
- 2 If you unsaved changes, select **File > Save** to save your project.
- 3 In the Outline Editor, right-click and select **Generate Source Code > Application Model**.
Generating the project transforms the model into Java source code in the mobile project.
- 4 To build the project:
 - 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 one of the following Ant targets to build the project.
 - **+Multi-Build**

- **+Multi-Build-Last**
- **+Target-Build**

For more information about how to use these Ant targets, see *Using webMethods Mobile Designer*.

4 Building the User Interface for a Mobile Application

■ Basic Structure of the Application User Interface	48
■ Defining Panes for the Application Window	49
■ Adding Views to the Application's User Interface	53
■ Renaming a View	54
■ Adding Content to a View	55
■ Programmatically Populating a ListView	57
■ Using a Content Provider to Populate a ListView	60
■ About User-Initiated Events and Listeners	64
■ Adding Listeners for User-Initiated Events	65
■ Defining Dialogs	67
■ Using Templates to Define Custom Objects for a Mobile Project	68

Basic Structure of the Application User Interface

The user interface is made up of a window, panes, views, and content within the views. Additionally, you can define dialogs.

Main Window for the Application

When using Mobile Development to design the user interface, your application contains a single main window for your application. The window defines the visible bounds of the display to use for an application.

When you create a mobile project, Mobile Development defines the application's main window for you.

Panes for the Window

You divide the main window into one or more panes. When creating an application for a small hand-held device, such as a mobile phone, you might want to use a single pane or maybe two, one for a navigation area and the other for a main area. When creating an application for a larger device, such as a tablet, you might want to use more panes. For more information, see [“Defining Panes for the Application Window” on page 49](#)

Views to Place in Panes

You define views that the application displays in the panes of the application's window. For information about how to define a view, see [“Adding Views to the Application's User Interface” on page 53](#). For information about the types of views you can add, see [“Objects to Use for Views” on page 76](#).

Contents of Views

Inside a view, you place the content you want the application to display. For example, you can add text fields, buttons, check boxes, etc. For more information, see [“Adding Content to a View” on page 55](#).

Dialogs

Define alert dialogs if you need small pop-ups that display over a view. Use dialogs to:

- Present information to the user.
- Interact with the user by presenting a simple question, for example, a question requiring a “yes” or “no” answer.

An application can display one dialog at a time. For more information, see [“Defining Dialogs” on page 67](#).

Defining Panes for the Application Window

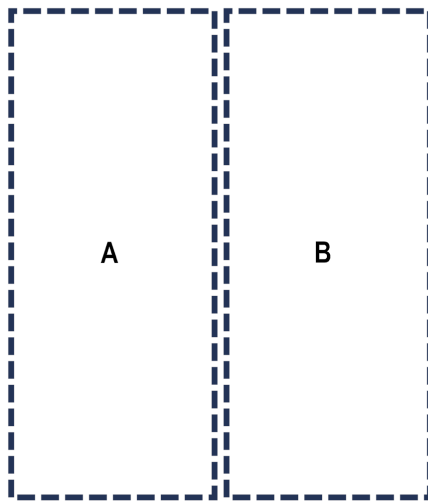
You need to define panes and pane configurations for the user interface of a mobile application. *Panes* are subsections of an application's window. The application displays views within panes. A *pane configuration* indicates how to lay out the panes within the window.

You define panes using the `PaneDefinition` object. You define a pane configuration using the `PaneConfiguration` object.

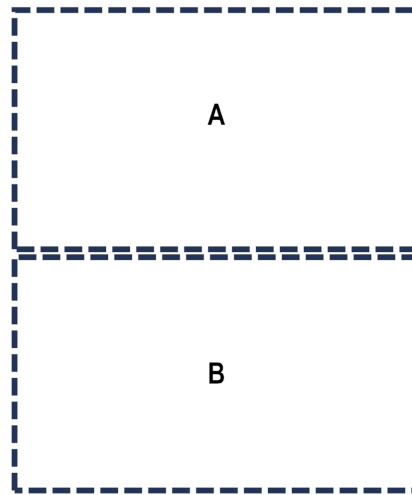
The simplest layout is a single pane. To have a single pane, add a single `PaneDefinition` child object to the `PaneConfiguration` object. The `PaneDefinition` object indicates the pane to display and the view that you initially want the application to display in the pane.



Another simple layout is to use two panes, either vertically (side-by-side) or horizontally (one on top of the other). To define this type of configuration, rather than adding the `PaneDefinition` child object directly to the `PaneConfiguration` object, you first add either a `VerticalSplitter` object or a `HorizontalSplitter` object to the `PaneConfiguration` object. You can then add two `PaneDefinition` child objects to the splitter object. The order you list the `PaneDefinition` objects is the order the panes display in the window. For example, if you list pane A followed by pane B, in a vertical arrangement pane A is on the left and in a horizontal arrangement pane A is on the top.

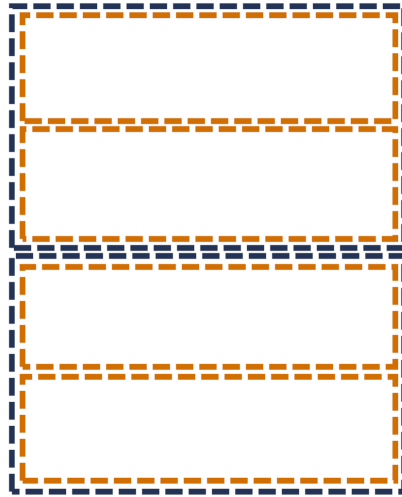


PaneConfiguration
VerticalSplitter
Left PaneDefinition
Right PaneDefinition



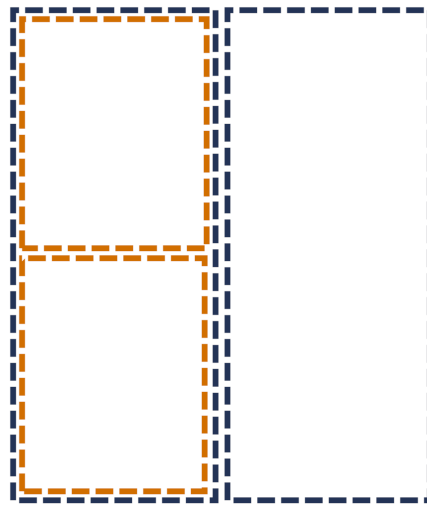
PaneConfiguration
HorizontalSplitter
Top PaneDefinition
Bottom PaneDefinition

If you need a more complex arrangement of panes for an application's user interface, you can nest `VerticalSplitter` and `HorizontalSplitter` objects under parent splitter objects. For example, the following shows a layout with four panes arranged horizontally.



PaneConfiguration
HorizontalSplitter
 Top HorizontalSplitter
 Top PaneDefinition
 Bottom PaneDefinition
 Bottom HorizontalSplitter
 Top PaneDefinition
 Bottom PaneDefinition

The following shows another example that has three panes, with two panes displayed horizontally on the left and a single pane on the right.



PaneConfiguration
 VerticalSplitter
 Left HorizontalSplitter
 Top PaneDefinition
 Bottom PaneDefinition
 Right PaneDefinition

Keep the following usage notes in mind when working with panes:

- When you create a new project, by default, the application's window is named "MainWindow" and has a pane configuration made up of two panes named "MasterPane" and "DetailPane". The pane configuration is defined to arrange the panes vertical, with the MasterPane on the left and the DetailPane on the right.

If you do not want to use the default panes and configuration, you can delete them.

- You can add as many panes as you want.
- You can define multiple pane configurations and have the application switch pane configurations, as needed.

For example, you might only want a single pane for an application's login panels, but switch to a multi-pane setup after the user logs in.

- You can use the same named panes in multiple pane configurations.
- A pane configuration can include one or more panes.

For smaller devices, such as phones, you might only use a single pane or maybe two, one for the navigation and one for the main view. For larger devices, such as tablets, you might want to use additional panes.

- The order you list the **PaneDefinition** child objects within a **PaneConfiguration** parent object is the order the panes display in the window.

- By default, when you use a `HorizontalSplitter`, the split creates two equal sections, one on top of the other. However, you can define the size for *one* of the sections, and the other section uses the remaining space. To set the absolute size of a pane, set the `HorizontalSplitter` object's `Height` property. For more information, see [“HorizontalSplitter Properties” on page 75](#).

Note: An exception to the default behavior is when you use a `HorizontalSplitter` with a `NavigationView` in the bottom pane. In this case, the size of the bottom pane is set to the height required for the `NavigationView`. The top pane uses the remaining space.

- By default, when you use a `VerticalSplitter`, the split creates two equal side-by-side sections. However, you can define the size for *one* of the sections, and the other section uses the remaining space. To set the absolute size of a pane, set the `VerticalSplitter` object's `Width` property. For more information, see [“VerticalSplitter Properties” on page 76](#).

Adding Views to the Application's User Interface

To add a view to the user interface, you add a `ListView`, `NavigationView`, `View`, or `WebView` object to the model. You can then reference the view in your model to display it in a pane or transition to it when a user-initiated event occurs.

To add a view to the user interface

- 1 Ensure the mobile project is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the **Model** section of the Outline Editor, expand the outline so that you view the **Views** node.
- 3 Right-click the **Views** node and select **New Child > *child_object***, where *child_object* is the name of the type of view you want to add.

For a description of the types of views you can add, see [“Objects to Use for Views” on page 76](#).

- 4 Set the properties for the view.

For more information, see [“Setting Properties in the Outline Editor” on page 18](#). For descriptions of the projects, see:

- [“ListView Properties” on page 77](#)
- [“NavigationView Properties” on page 79](#)
- [“View Properties” on page 79](#)
- [“WebView Properties” on page 80](#)

- 5 To use the view in the user interface, you can do one or more of the following:
 - To display the view when using a pane configuration, specify the view in the **Start View** property of the **PaneDefinition** object.
 - To transition to the view when a user-initiated event occurs, set the **View** property of the specific **Transition** event action object. For more information, see [“Objects to Use for Event Actions” on page 104](#) and [“Transition Properties” on page 108](#).

You can also add code to your application logic to programmatically transition to the view. For more information, see [“Logic to Transition to Another View” on page 125](#).

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Renaming a View

You specify a name for a view by setting the view’s **Name** property. When you generate sources for the mobile project, the `view_nameControllerImpl.java` Java class that Mobile Development generates in the `src` folder includes the view name in the name of the Java class. After generating sources, if you want to change the name of the view, use the following procedure.

To rename a view

- 1 Ensure the view is displayed in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#) or [“Displaying a Window, View, or Dialog in the Outline Editor” on page 16](#).
- 2 In the **Model** section of the Outline Editor, select the view node that you want to rename.
- 3 Type the new name for the view in the **Name** property, which is displayed in the **Properties** section of the Outline Editor.
- 4 Save the mobile project and generate sources for the mobile project. For more information, see [“Generating Sources for a Mobile Project” on page 25](#).

Mobile Development generates a new `new_view_nameControllerImpl.java` Java class for the view where `new_view_name` is the new name you assigned to the view.

Mobile Development does *not* remove the `old_view_nameControllerImpl.java` Java class, where `old_view_name` is the previous name of the view. Mobile Development retains this file in the event that you previously added custom code to the `old_view_nameControllerImpl.java` Java class.

- 5 Update the `new_view_nameControllerImpl.java` class with any custom code that you added to the `old_view_nameControllerImpl.java` Java class.

- a In the Package Explorer or Navigator view, locate the `src > package > ui > controller > impl` folder, which contains both the `new_view_nameControllerImpl.java` and `old_view_nameControllerImpl.java` Java classes.
- b Open both Java classes and copy all custom code from the `old_view_nameControllerImpl.java` to `new_view_nameControllerImpl.java`.
- c Save both files.
- d Delete the `old_view_nameControllerImpl.java` Java class.

Adding Content to a View

What You Can Add to a View

To define a view's user interface, in the Outline Editor you add user interface objects to the model as child objects of the view object. For descriptions of the objects you can add to views, see the following:

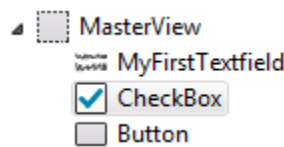
- [“Objects to Use for the Layout of the User Interface” on page 81](#)
- [“Objects to Use for Tables” on page 84](#)
- [“Objects to Use for User Interface Controls” on page 87](#)
- [“Objects to Use for Content Providers” on page 101](#)
- [“Objects to Use for Event Listeners” on page 103](#)
- [“Objects to Use for Event Actions” on page 104](#)

The user interface objects that are valid in a view is based on the specific type of view, that is, whether you are adding the object to a `ListView`, `NavigationView`, `View`, or `WebView`. For example, the only valid object that you can add to a `NavigationView` is a `NavButton` object. When using the Outline Editor to build a view's user interface, the Outline Editor only lists objects that are valid for each type of view.

The following sections provide general information about adding content to views. For information specifically about adding content to a `ListView`, see [“Programmatically Populating a ListView” on page 57](#) and [“Using a Content Provider to Populate a ListView” on page 60](#).

Order of Objects You Add to the View

The order of the child objects under a view object dictates the order the objects will display in a view. For example, assume the following is defined for a view:



The following shows what the user interface might look like when the application executes:



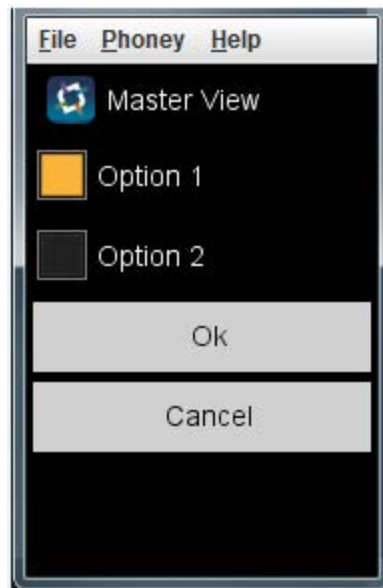
Nesting Objects in a View

Some objects allow you to nest child objects under them. When the view is displayed, the child objects are displayed inside their parent object. If you nest multiple objects, they display in the parent object in the order you list them in the model in the Outline Editor.

For example, if you use a **Container** object, you can nest child objects under the **Container** object.



The result is that in the view's user interface, the child objects you place under the **Container** object display within the container in the user interface. The order of the child objects are the order in which the child objects are displayed within the **Container** object.



Programmatically Populating a ListView

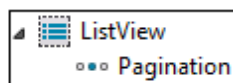
Use a `ListView` object to display a list of items. You can populate a `ListView` by adding logic to the view's controller.

Note: Rather than adding logic to the view's controller to populate the `ListView` object, you can use a content provider to populate a `ListView`. For example, you might populate the `ListView` with the response from a REST service. For more information, see [“Using a Content Provider to Populate a ListView” on page 60](#).

Objects to Add to the Project Model

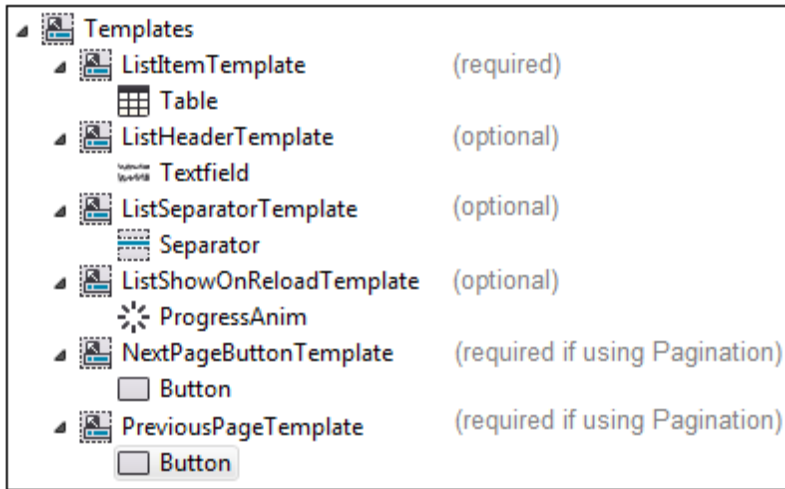
This section describes the objects you add to the model if you want to add logic to the view's controller to populate the `ListView` object.

- In the `UserInterface > Views` section of the model, you need to add the following objects.



Object	Description
<code>ListView</code>	Required. Defines the <code>ListView</code> .
<code>Pagination</code>	Optional. Specifies how many list items to display per page and identifies templates for objects that the user selects to view the next or previous page of results.

- In the `UserInterface > Templates` section of the model, you add templates that indicate how to display the data within the `ListView`.



Template for...	Description
List item	<p>Required. You must create a template that defines how to display a single item from the data source.</p> <p>Typically, you create a template for a <code>Table</code> or <code>TableButton</code> object.</p> <p>When you add logic to the controller for the view, you invoke the Java class for this template to display an item in the view.</p>
List header	<p>Optional. You can create a template to provide a header for the <code>ListView</code>. For example, you might create a template for an object like a <code>Textfield</code> or an <code>Image</code>.</p> <p>If you want to provide a header for the <code>ListView</code>, specify the template in the <code>ListView</code> object's <code>List View Header</code> property.</p>
List separator	<p>Optional. You can create a template for an object that you want to display between each list item in the <code>ListView</code>. For example, you might create a template for an object like a <code>Separator</code>, <code>Spacer</code>, or <code>Image</code> object.</p> <p>If you want to provide a separator for the list of data, specify the template in the <code>ListView</code> object's <code>Separator</code> property.</p>
Control to show when reloading data	<p>Optional. You can create a template for an object that the application will display when the application is accessing the data source to refresh the data. For example, you might create a template for an object like a <code>ProgressAnim</code>, <code>Image</code>, or <code>Textfield</code> object.</p> <p>If you want the application to display an object when refreshing the data, specify the template in the <code>ListView</code> object's <code>Show On Reload</code> property.</p>

Template for...	Description
Control to display the next page of results	<p>Conditionally required. Create a template for an object, for example, a Button object, that a user selects at run time to display the next page of results.</p> <p>This template is required if you are using the Pagination object. You specify this template in the Pagination object's Next Page Template property.</p>
Control to display the previous page of results	<p>Conditionally required. Create a template for an object, for example, a Button object, that a user selects at run time to display the previous page of results.</p> <p>This template is required if you are using the Pagination object. You specify this template in the Pagination object's Previous Page Template property.</p>

Logic in the Controller for the View

When you generate sources for a mobile project, Mobile Development generates a Java class named `view_nameControllerImpl.java` in the `src.package_name.ui.controller.impl` package. For example, if you assigned the view the name "MyListView" and the package name "my.company.com", Mobile Development generates `MyListViewControllerImpl.java` in the `src.my.company.com.ui.controller.impl` package.

To provide logic to populate the **ListView**, you override the following methods:

- `getNumberOfRows()`. At run time, the application invokes this method to determine the total number of list items to display.

Add logic to this method to determine the number of list item results to display, for example:

```
public int getNumberOfRows(ListView listView) {
    Vector my_items = getMyData();
    return my_items.size();
}
```

- `getCell`. At run time, the application invokes this method to obtain a list item to display in the **ListView**.

Add logic to this method that returns a single list item to display, for example:

```
public nUIDisplayObject getCell(ListView listView, int rowIndex){
    Vector myItems = getData();
    final ListItemTemplate item = new ListItemTemplate();
    item.initializeWithData(myItems[rowIndex]);
    item.setIndex(rowIndex + 1);
    return item;
}
```

- `onRowSelect()`. At run time, the application invokes this method when a user selects a row in the list of results. Optionally add logic to this method if you want to take some action when a list item is selected. For example, you might want to transition to another view or open a dialog.

```
public void onRowSelect(ListView listView, int rowIndex) {
    Vector myItems = getData();
    getTransitionStackController().pushViewController(new
        ItemDetailViewImpl(myItems[rowIndex]));
}
```

Using a Content Provider to Populate a ListView

Use a `ListView` object to display a list of items. You can populate a `ListView` object using a `ContentProvider` object to retrieve data from a data source. Mobile Development supports the following types of data sources:

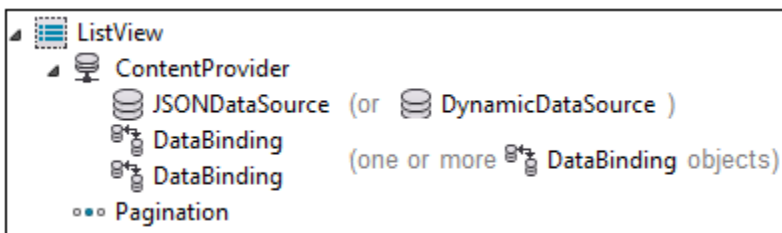
- Use a `DynamicDataSource` object if you want to specify a method that the application executes to obtain the data to display in the `ListView` object. You are responsible for providing the logic for the method. The method you code must return an instance of `IListViewDataSource`, which is in the `gen/api-src` folder in the `com.softwareag.mobile.runtime.toolkit.delegates` package.
- Use a `JSONDataSource` object to execute a REST service to obtain JSON-format data to display in the `ListView` object.

Note: Instead of using a content provider, you can add logic to the view's controller to populate the `ListView` object. For more information, see [“Programmatically Populating a ListView” on page 57](#).

Objects to Add to the Project Model

To obtain data from a data source to populate a `ListView` object, you need to add several objects to the project's model.

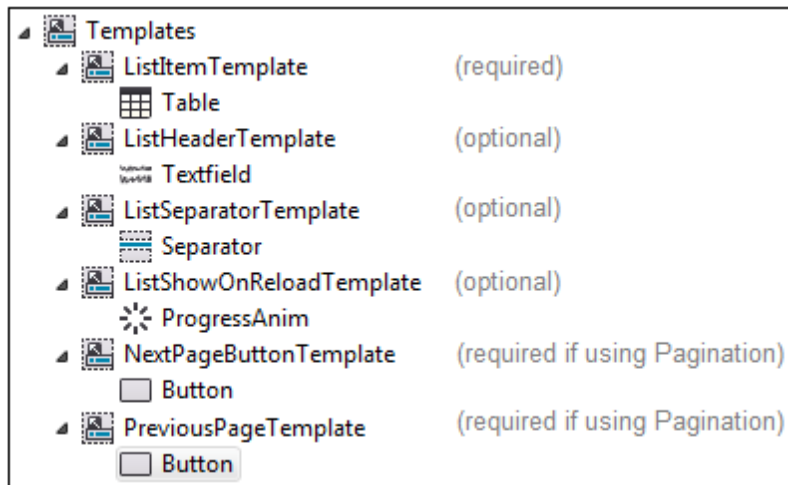
- In the `UserInterface > Views` section of the model, you need to add the following objects.



Object	Description
<code>ListView</code>	Required. Defines the view that you want to populate with data from a data source.

Object	Description
ContentProvider	Required. Identifies the data source that indicates from where to obtain the data and the template to use to present the data.
DynamicDataSource or JSONDataSource	Required. Provides details about the specific method to invoke to obtain the data. If you use a JSONDataSource object, you must define the service you want to use in the Services section of the model.
DataBinding	Required. Identifies an element in the data you obtained from the data source and binds it to a specific control defined within the template you are using to display the returned data. Use one DataBinding object for each element you want to bind to a control.
Pagination	Optional. Specifies how many list items to display per page and identifies templates for objects that the user selects to view the next or previous page of results.

- In the **UserInterface > Templates** section of the model, you add templates that indicate how to display the data within the **ListView**.



Template for...	Description
List item	Required. You must create a template that defines how to display a single item from the data source. Typically, you create a template for a Table or TableButton object. You reference this template in the ContentProvider object's Template property.

Template for...	Description
List header	<p>Optional. You can create a template to provide a header for the <code>ListView</code>. For example, you might create a template for an object like a <code>Textfield</code> or an <code>Image</code>.</p> <p>If you want to provide a header for the <code>ListView</code>, specify the template in the <code>ListView</code> object's <code>List View Header</code> property.</p>
List separator	<p>Optional. You can create a template for an object that you want to display between each list item in the <code>ListView</code>. For example, you might create a template for an object like a <code>Separator</code>, <code>Spacer</code>, or <code>Image</code> object.</p> <p>If you want to provide a separator for the list of data, specify the template in the <code>ListView</code> object's <code>Separator</code> property.</p>
Control to show when reloading data	<p>Optional. You can create a template for an object that the application will display when the application is accessing the data source to refresh the data. For example, you might create a template for an object like a <code>ProgressAnim</code>, <code>Image</code>, or <code>Textfield</code> object.</p> <p>If you want the application to display an object when refreshing the data, specify the template in the <code>ListView</code> object's <code>Show On Reload</code> property.</p>
Control to display the next page of results	<p>Conditionally required. Create a template for an object, for example, a <code>Button</code> object, that a user selects at run time to display the next page of results.</p> <p>This template is required if you are using the <code>Pagination</code> object. You specify this template in the <code>Pagination</code> object's <code>Next Page Template</code> property.</p>
Control to display the previous page of results	<p>Conditionally required. Create a template for an object, for example, a <code>Button</code> object, that a user selects at run time to display the previous page of results.</p> <p>This template is required if you are using the <code>Pagination</code> object. You specify this template in the <code>Pagination</code> object's <code>Previous Page Template</code> property.</p>

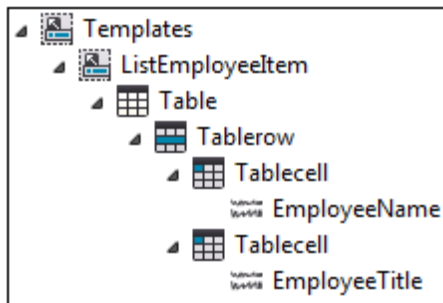
- In the `Services` section of the model, if you want to display data from a REST service in the `ListView` object, define the service. For more information, [“Adding Services to a Mobile Project” on page 43](#).

Example

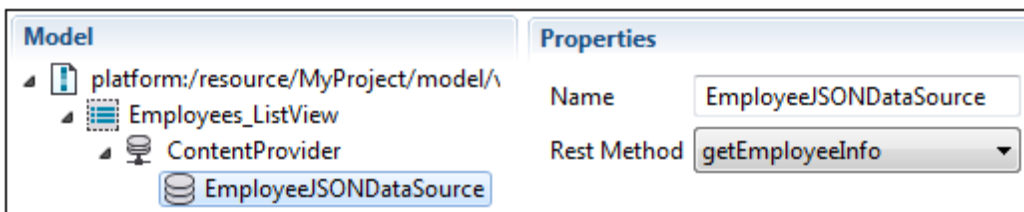
Consider you have a REST service that has a `getEmployeeInfo` method that returns information about employees. The following illustrates the type of information that the REST service returns.

Employees	EmployeeID	19	
	name	Leanna Jones	
	startDate	1995-01-04	
	department	Research	
	title	Manager	
	EmployeeID	30	
	name	Zane Smith	
	startDate	1997-06-20	
	department	Research	
	title	Engineer	

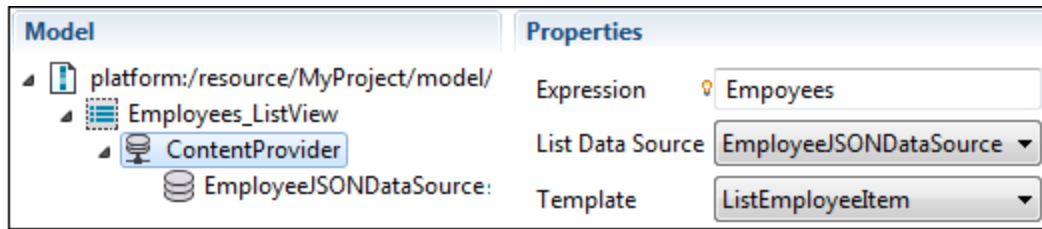
Set up a template to define how to display the response data in the `ListView`. In this example, the `ListView` will display an employees name and title. Define a template for a `Table` object that defines a row with two table cells. Each table cell contains a `Textfield` object. One cell is to display an employee name and the other to display an employee title.



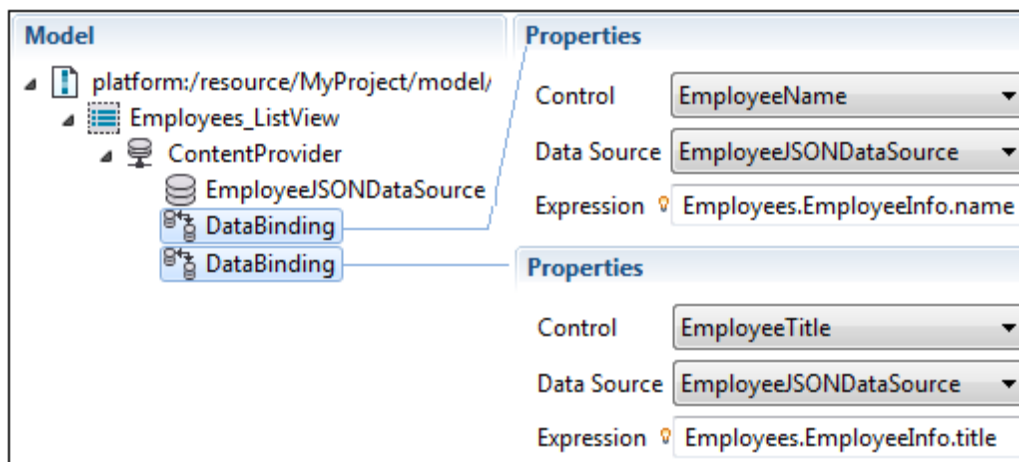
Add a `ListView` object to the model and a child `ContentProvider` object. Then add a child `JSONDataSource` object to the `ContentProvider` object. Set the properties for the `JSONDataSource` object to indicate that the data source is the `getEmployeeInfo` method of a REST service that is defined in the `Services` section of the model.



Set the properties for the `ContentProvider` object to indicate that the data source is the “`EmployeeJSONDataSource`” that is defined with the `JSONDataSource` object and that the data to display is within the “`Employees`” part of the response. Additionally, set the template to use to format the output to the “`ListEmployeeItem`” template defined in the `Templates` section of the model.



Add a **DataBinding** object to bind the employee name from the response, `Employees.EmployeeInfo.name`, to the “EmployeeName” **Textfield** object in a table cell in the “ListEmployeeItem” template. Then add a second **DataBinding** object to bind the employee title from the response, `Employees.EmployeeInfo.title`, to the “EmployeeTitle” **Textfield** object in a table cell in the “ListEmployeeItem” template.



About User-Initiated Events and Listeners

A *user-initiated event* is when a user interacts with a control in the application’s user interface, for example, when a user presses a button, types text in a text field, selects a check box, etc.

You can add listeners to your model so that when a user-initiated event occurs for a control, your application can respond by taking an appropriate action. For more information about how to add listeners, see [“Adding Listeners for User-Initiated Events” on page 65](#).

Types of Listeners

Mobile Development supports the following types of listeners:

- **GainFocusListener** that listens for when a user selects an object so that the user interface object gains focus.
- **LoseFocusListener** that listens for when a user interface object loses focus because the user stops selecting an object when the user selects another user interface control.

- **PostEditListener** that listens for when a user edits an object, for example an entry field, and generates an event after the object is edited.
- **PreEditListener** that listens for when a user edits an object, for example an entry field, and generates an event when the user first selects the object for editing.
- **TriggerListener** that listens for when a user uses an object, for example, presses a button.

When you add the objects for event listeners to your application, Mobile Development generates the code to listen for the user-initiated events. You do not need to add custom logic to listen for the events.

How the Application Responds to a User-Initiated Event

When you add listeners to your model, at run time if the listener detects the associated user-initiated event, the application fires an event. For example, if you add a **TriggerListener** object to a **Button** object, at run time when the user presses the button, the application fires an event.

In addition to specifying listeners in the model, you can also define how the application responds to the event, in other words, the action the application takes when the event occurs. Mobile Development provides user interface objects for the following actions:

- **Back** action to transition to the previous view.
- **ChangePaneConfiguration** action to change the configuration of panes in the application's window.
- **Delegate** action to execute a method that you code.
- **OpenDialog** action to open an alert dialog that you have defined in your model.
- **ToggleVisibility** action to make a user interface object that you have defined in your model either visible or hidden. If the object is currently visible, the action hides the object. If the object is currently hidden, the action makes the object visible.
- **Transition** action to transition to another view that you have defined in your model.

For more information about these objects, see [“Objects to Use for Event Actions” on page 104](#).

Adding Listeners for User-Initiated Events

To add a listener for user-initiated events, add a listener object as a child of the user interface control for which you want to listen. For example, if you want to listen for when a user presses a button, add the listener object as a child of the **Button** object.

After adding the listener object, add an event action object as a child of the listener. For example, if you want to transition back to the previous view when a user presses the **Button** object, you add a **TriggerListener** event listener object as a child of the **Button** object and the **Back** event action object as a child of the **TriggerListener** object.

To add event listeners and associated actions

- 1 Ensure the view to which you want to add event listeners is displayed in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#) or [“Displaying a Window, View, or Dialog in the Outline Editor” on page 16](#).
- 2 In the Model section of the Outline Editor, right-click the user interface control for which you want to add a listener and select **New Child > *event_listener_object***, where *event_listener_object* is one of the following event listener objects:
 - GainFocusListener
 - LoseFocusListener
 - PostEditListener
 - PreEditListener
 - TriggerListener

For more information about these objects, see [“About User-Initiated Events and Listeners” on page 64](#) and [“Objects to Use for Event Listeners” on page 103](#).

Note: The event listener objects have no properties that you need to set.

- 3 Right-click the event listener object you added and select **New Child > *event_action_object***, where *event_action_object* is one of the following event action objects:
 - Back
 - ChangePaneConfiguration
 - Delegate
 - OpenDialog
 - ToggleVisibility
 - Transition

For more information about these objects, see [“About User-Initiated Events and Listeners” on page 64](#) and [“Objects to Use for Event Actions” on page 104](#).

- 4 Select the event action object that you added.
- 5 In the Properties section of the screen, set properties for the event action object.

For more information about these objects, see [“Setting Properties in the Outline Editor” on page 18](#) and [“Objects to Use for Event Actions” on page 104](#).

Note: Based on the event action you are using, you might also need to add application logic for the action. For more information, see [“Logic to Respond to a Listener Event” on page 124](#).

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project” on page 25](#).

Defining Dialogs

You can define alert dialogs for a mobile application. A dialog is a pop-up window that displays over a view. An application can only have dialog open at a time.

To add a dialog, you add an `AlertDialog` object to the `Dialogs` container in the Outline Editor. When you add an `AlertDialog` object to the user interface, Mobile Development automatically adds an `AlertDialogButton` as a child object. An `AlertDialog` object *requires* at least one child `AlertDialogButton` object.

To define a dialog for a mobile application

- 1 Ensure the mobile project is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the `Model` section of the Outline Editor, expand the outline so that you view the `Dialogs` node.
- 3 Right-click the `Dialogs` node and select `New Child > AlertDialog`.
Mobile Development adds an `AlertDialogButton` child object as well.
- 4 Select the `AlertDialog` object, and in the `Properties` section of the Outline Editor set the properties for the dialog. For more information, see [“AlertDialog Properties” on page 83](#).
Use the `Text` property to specify the text you want displayed in the dialog.
- 5 Select the `AlertDialogButton` object that Mobile Development added for you, and in the `Properties` section of the Outline Editor set the properties for the button. For more information, see [“AlertDialogButton Properties” on page 83](#).
- 6 If you want the dialog to contain an additional button, right-click the `AlertDialog` object and select `New Child > AlertDialogButton` to add the button. Then select the button and set the properties. Repeat this step for each additional button you want in the dialog.

- 7 To use the dialog in the user interface, you can do one or more of the following:
- To display the view in response to a user-initiated event, for example, when a user selects a check box, specify the dialog for a **OpenDialog** event action object. For more information, see [“Objects to Use for Event Actions”](#) on page 104, [“Adding Listeners for User-Initiated Events”](#) on page 65, and [“OpenDialog Properties”](#) on page 107.
 - For information about how to add code to open a dialog, see [“Logic to Display and Close a Dialog”](#) on page 122.
 - For information about the code you can add to a dialog, see [“Logic for a Dialog”](#) on page 121.

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project”](#) on page 25.

Using Templates to Define Custom Objects for a Mobile Project

You can add templates to your mobile project to add customizations to the following user interface objects that Mobile Development provides.

- Button
- ColumnLayout
- DateEntry
- Entry
- GridLayout
- Image
- ListViewElement
- ProgressAnim
- SearchEntry
- Separator
- Spacer
- Table
- TableButton
- TextField
- WebViewElement

For more information about creating a template, see [“Creating a Template for a Custom Object”](#) on page 68.

After you create a template for a user interface object, you can use it in your mobile application user interface. For more information, see [“Using a Template in the Mobile Application User Interface”](#) on page 70.

Creating a Template for a Custom Object

Create a template if you want to customize a user interface object for your mobile project.

To create a template

- 1 Ensure the mobile project to which you want to add a template is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the **Model** section of the Outline Editor, expand the outline so that you view the **UserInterface > Templates** node.
- 3 Right-click the **Templates** node and select **New Child > Template**.
Mobile Development adds a **Template** child node.
- 4 Select the new **Template** node.
- 5 In the **Properties** section of the Outline Editor, specify a Java class name in the **Class Name** property.
For example, if you want to customize the **Button** user interface object, you might specify `MyButtonTemplate`.
Mobile Development renames the **Template** node to the name you specified in the **Class Name** property.
- 6 Right-click the template node you just added and select **New Child > *object***, where *object* is the type of object you want to customize.
For example, if you want to customize the **Button** object, select **New Child > Button**.
- 7 Select the new node you added and in the **Properties** section of the Outline Editor, fill in the properties. For more information about properties, see [“Template Properties” on page 109](#).
- 8 Save the mobile project and generate sources for the mobile project. For more information, see [“Generating Sources for a Mobile Project” on page 25](#).

Mobile Development generates the following Java classes for the template:

- `Abstracttemplate_name.java` in the `gen/src` folder in the `package_name.ui.templates` package

This class contains the standard logic to handle the user interface object you are customizing with the template.

Important! Do *not* update this Java class. Mobile Development regenerates it each time you generate sources and any changes you make will be overwritten.

- `template_name.java` in the `src` folder in the `package_name.ui.templates` package
You update the `templat_name.java` class to customize the user interface object.

For the generated Java classes:

- *template_name* is the Java class name you specified for the **Class Name** property of the template node.
 - *package_name* is the package name you specified for your mobile project.
- 9 Add the logic to customize the user interface object to the *template_name.java* class.

Using a Template in the Mobile Application User Interface

After you create a template to customize a user interface object, you can use the object in the user interface of your mobile application.

The following procedure describe how to use a template by using the `TemplateReference` object. You can also use templates to customize a `ListView`. For more information, see [“Programmatically Populating a ListView” on page 57](#) and [“Using a Content Provider to Populate a ListView” on page 60](#).

To use a template in a mobile application user interface

- 1 Ensure the mobile project or specific window, view, or dialog to which you want to add the template is open in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor” on page 15](#) or [“Displaying a Mobile Project in the Outline Editor” on page 15](#).
- 2 In the **Model** section of the Outline Editor, expand the **UserInterface** part of the outline so that you view the location where you want to add the template.
- 3 Right-click the node in which you want to use the template and select **New Child > TemplateReference**.

Note: If `TemplateReference` is not listed in the right-click menu, it is not valid where you want to use the template.

- 4 Select the `TemplateReference` node.
- 5 In the **Properties** section of the Outline Editor, specify the following properties:

For this property...	Specify...
Name	Name for your own reference purpose. This name does not appear in the application’s user interface.
Template	Template that you want to use. The list includes the templates that you have added to your project.

Tip! To update the Java classes that Mobile Development generates for the project so that your changes are represented in the generated sources, save the project and regenerate sources. For instructions, see [“Generating Sources for a Mobile Project”](#) on page 25.

5 User Interface Object Reference

■ User Interface Objects	72
■ Application Node Properties	72
■ Objects to Use for Windows	73
■ Objects to Use for Panes	74
■ Objects to Use for Views	76
■ Objects to Use for the Layout of the User Interface	81
■ Objects to Use for Dialogs	82
■ Objects to Use for Tables	84
■ Objects to Use for User Interface Controls	87
■ Objects to Use for Content Providers	101
■ Objects to Use for Event Listeners	103
■ Objects to Use for Event Actions	104
■ Objects to Use for Templates	108

User Interface Objects

The following table lists the objects you can define for your application and where you can find a description of the user interface objects and a description of the properties to set for each object.

For information about...	See...
Application node	“Application Node Properties” on page 72
Windows	“Objects to Use for Windows” on page 73
Panes	“Objects to Use for Panes” on page 74
Views	“Objects to Use for Views” on page 76
Layout	“Objects to Use for the Layout of the User Interface” on page 81
Dialogs	“Objects to Use for Dialogs” on page 82
Tables	“Objects to Use for Tables” on page 84
Controls	“Objects to Use for User Interface Controls” on page 87
Content Providers	“Objects to Use for Content Providers” on page 101
Event Listeners	“Objects to Use for Event Listeners” on page 103
Event Actions	“Objects to Use for Event Actions” on page 104
Templates	“Objects to Use for Templates” on page 108

Application Node Properties

The following table provides descriptions of the properties you can set for the mobile project’s root application node.

Property	Description
Bundle Id	Package name for your mobile project. You initially define the package name for a mobile project when you create the project using the New Mobile Development Project wizard. You can use this property to change the package name. For more information, see “Changing the Package Name” on page 131 .
Default Language	Default language for the application. For more information, see “Setting the Default Language for the Project” on page 42 .

Property	Description
Name	<p>Name of the mobile application. This is an internal application name that Mobile Development uses.</p> <p>You initially define the application name when you create the project using the New Mobile Development Project wizard. You can use this property to change the name. For more information, see “Renaming the Application” on page 130.</p>
Orientation	<p>Whether you want the application to display in portrait mode, landscape mode, or rotate as a user turns the device.</p> <p>You initially define the orientation setting when you create the project using the New Mobile Development Project wizard. You can use this property to reconfigure the setting. For more information, see “Configuring the Orientations Setting for the Application” on page 40.</p>
Res Handler	<p>Name of the resource handler for the mobile application project. By default, the mobile application uses the default application that Mobile Development provides, which is UniversalResHandler.</p> <p>If you want to use a custom resource handler, use this property to specify its fully-qualified name.</p> <p>For more information, see “Defining Resources for the Mobile Project” on page 31.</p>
Use Mobile Administrator	<p>Whether you want to use Mobile Administrator to distribute the final binary for the application. For more information about how to set up your mobile project to use Mobile Administrator, see “Using Mobile Administrator to Manage and Distribute Mobile Applications” on page 24.</p>

Objects to Use for Windows

The following table provides a description of the user interface object you use for the application’s window.

Object	Description
Window	<p>Defines the application’s window.</p> <p>For information about setting properties for the Window object, see “Window Properties” on page 74.</p>

Window Properties

Property	Description
Name	Name you assign the application's window. This name does not appear in the application's user interface.
Start Pane Configuration	Name of the pane configuration that you want to use when the window is initially displayed. Specify the name of a <code>PaneConfiguration</code> object that you previously defined for the mobile project.

Objects to Use for Panes

The following table provides descriptions of the user interface objects you use to define panes for an application's window. For more about using panes, see [“Defining Panes for the Application Window” on page 49](#).

Object	Description
HorizontalSplitter	<p>Indicates that you want to display two panes horizontally, one on top of the other.</p> <p>For information about setting properties for the <code>HorizontalSplitter</code> object, see “HorizontalSplitter Properties” on page 75.</p>
PaneConfiguration	<p>Specifies the name of a configuration of panes.</p> <p>Add <code>HorizontalSplitter</code>, <code>VerticalSplitter</code>, and/or <code>PaneDefinition</code> child objects to define how to place panes in the application's <code>Window</code> object when using this pane configuration.</p> <p>For information about setting properties for the <code>PaneConfiguration</code> object, see “PaneConfiguration Properties” on page 75.</p>
PaneDefinition	<p>Specifies the following for a single pane in a pane configuration:</p> <ul style="list-style-type: none">■ Name of the pane.■ Name of a view that you want initially displayed in the pane.■ Flag indicating whether the view is visible or not. <p>For information about setting properties for the <code>PaneDefinition</code> object, see “PaneDefinition Properties” on page 76.</p>

Object	Description
VerticalSplitter	<p>Indicates that you want to display two panes vertically, side by side.</p> <p>For information about setting properties for the VerticalSplitter object, see “VerticalSplitter Properties” on page 76.</p>

HorizontalSplitter Properties

Property	Description
Height	<p>Absolute size to use for the height of one of panes, either the top or bottom pane. The other pane uses the remaining space available. You can specify the height using either a percentage value or the number of pixels.</p> <ul style="list-style-type: none"> ■ To set the absolute size to use for the top pane, type the value. For example: <ul style="list-style-type: none"> ■ To use 320 pixels for the top pane, specify: 320 ■ To use 38 percent for the top pane, specify: 38% ■ To set the absolute size for the bottom pane, type a comma followed by the value. For example: <ul style="list-style-type: none"> ■ To use 320 pixels for the bottom pane, specify: , 320 ■ To use 38 percent for the bottom pane, specify: , 38% <p>If you do not specify a value, the split creates two equal sections.</p> <hr/> <p>Note: An exception to the default behavior is when you use a HorizontalSplitter with a NavView in the bottom pane. In this case, the size of the bottom pane is set to the height required for the NavView. The top pane uses the remaining space.</p> <hr/>

PaneConfiguration Properties

Property	Description
Name	Name you assign to the pane configuration.

PaneDefinition Properties

Property	Description
Name	Name of the pane.
Start View	Name of a view that you want initially displayed in the pane. This can be a name that you previously defined for a <code>View</code> , <code>ListView</code> , <code>NavigationView</code> , or <code>WebView</code> object.
Visible	Whether the pane is visible or hidden.

VerticalSplitter Properties

Property	Description
Width	<p>Absolute size to use for the width of one of panes, either the left or right pane. The other pane uses the remaining space available. You can specify the width using either a percentage value or the number of pixels.</p> <ul style="list-style-type: none">■ To set the absolute size to use for the left pane, type the value. For example:<ul style="list-style-type: none">■ To use 320 pixels for the left pane, specify: 320■ To use 38 percent for the left pane, specify: 38%■ To set the absolute size for the right pane, type a comma followed by the value. For example:<ul style="list-style-type: none">■ To use 320 pixels for the right pane, specify: , 320■ To use 38 percent for the right pane, specify: , 38% <p>If you do not specify a value, the split creates two equal sections.</p>

Objects to Use for Views

The following table provides descriptions of the types of views that you can use in an application's user interface.

Object	Description
ListView	<p>Defines a view that displays a list of data obtained from a specified data source.</p> <p>Add a ContentProvider child object to the ListView object to define the content you want to list in the view.</p> <p>For information about setting properties for a ListView, see “ListView Properties” on page 77.</p>
NavigationView	<p>Defines a view that you want to use for navigation in your application.</p> <p>The navigation view has different formats based on the platform. For example, for some platforms the navigation view might display as a menu bar that is always visible and uses both icons and text. For other platforms, the navigation view might have hidden menu items that are displayed only when a user presses a button.</p> <p>For information about setting properties for a NavigationView object, see “NavigationView Properties” on page 79.</p>
View	<p>Defines a general purpose view for your application.</p> <p>For information about setting properties for a View object, see “View Properties” on page 79.</p>
WebView	<p>Defines a view in which you want to display Web content.</p> <p>For information about setting properties for a WebView object, see “WebView Properties” on page 80.</p>

ListView Properties

Property	Description
Back Button Text	<p>Text to display on the Back button.</p> <p>If you do not set this property, the default is to display the Header Text property value of the previous view, which will be displayed if the user presses the Back button.</p>
Background Color	Background color of the view.
Background Image	Image to display as the background for the view.
Header Background Color	Background color of the header area of the view.
Header Image	Image to display as the header of the view.

<u>Property</u>	<u>Description</u>
Header Text	Text you want displayed in the header area of the view. Leave this property blank if you do not want text in the header.
Hide Back Button	Whether you want the Back button in the view to be displayed or hidden.
HScrollable	Whether you want to allow horizontal scrolling in the view.
Inner Height	Usable height of the view in which you can insert child objects.
Inner Width	Usable width of the view in which you can insert child objects.
Inner X	Distance from the view's left edge to where child elements are drawn.
Inner Y	Distance from the view's top edge to where child elements are drawn.
Inner YSpacing	Vertical distance between each element in the view.
List View Header	Template that defines an object to display as the header for the list of data displayed in the ListView. Specify a template that you previously defined. The template should customize an object like a <code>Textfield</code> or <code>Image</code> object.
Name	Name that you assign the view. This name does not appear in the application's user interface.
Popup Dismiss Text	For the iOS platform, the text to use on a label that closes an open keyboard or drop-down list.
Separator	Template that defines an object to display between the each list item displayed in the ListView. Specify a template that you previous defined. The template should customize an object like a <code>Separator</code> object.
Show On Reload	Template that defines an object to display while the application is obtaining data to refresh the list of items in the view. Specify a template that you previous defined. The template should customize an object like a <code>ProgressAnim</code> , <code>Image</code> or <code>Textfield</code> object.
VScrollable	Whether you want to allow vertical scrolling in the view.

NavView Properties

Property	Description
Name	Name you assign the view. This name does not appear in the application's user interface.

View Properties

Property	Description
Back Button Text	Text to display on the Back button. If you do not set this property, the default is to display the Header Text property value of the previous view, which will be displayed if the user presses the Back button.
Background Color	Background color of the view.
Background Image	Image to display as the background for the view.
Header Background Color	Background color of the header area of the view.
Header Image	Image to display as the header of the view.
Header Text	Text you want displayed in the header area of the view. Leave this property blank if you do not want text in the header.
Hide Back Button	Whether you want the Back button in the view to be displayed or hidden.
HScrollable	Whether you want to allow horizontal scrolling in the view.
Inner Height	Usable height of the view in which you can insert child objects.
Inner Width	Usable width of the view in which you can insert child objects.
Inner X	Distance from the view's left edge to where child elements are drawn.
Inner Y	Distance from the view's top edge to where child elements are drawn.
Name	Name you assign the view. This name does not appear in the application's user interface.
Popup Dismiss Text	For the iOS platform, the text to use on a label that closes an open keyboard or drop-down list.
VScrollable	Whether you want to allow vertical scrolling in the view.

WebView Properties

<u>Property</u>	<u>Description</u>
Back Button Text	Text to display on the Back button. If you do not set this property, the default is to display the Header Text property value of the previous view, which will be displayed if the user presses the Back button.
Background Color	Background color of the view.
Background Image	Image to display as the background for the view.
File	File that contains the Web content to display. The file should be in the project's resources\www folder.
Header Background Color	Background color of the header area of the view.
Header Image	Image to display as the header of the view.
Header Text	Text you want displayed in the header area of the view. Leave this property blank if you do not want text in the header.
Hide Back Button	Whether you want the Back button in the view to be displayed or hidden.
HScrollable	Whether you want to allow horizontal scrolling in the view.
Inner Height	Usable height of the view in which you can insert child objects.
Inner Width	Usable width of the view in which you can insert child objects.
Inner X	Distance from the view's left edge to where child elements are drawn.
Inner Y	Distance from the view's top edge to where child elements are drawn.
Name	Name you assign the view. This name does not appear in the application's user interface.
Popup Dismiss Text	For the iOS platform, the text to use on a label that closes an open keyboard or drop-down list.
Url	URL to the web page to load into the view.
VScrollable	Whether you want to allow vertical scrolling in the view.

Objects to Use for the Layout of the User Interface

The following table provides descriptions of user interface objects that you can use to define the layout of user interface objects within a view.

Object	Description
Group	<p>Creates a container that holds a group of user interface objects.</p> <p>To specify user interface objects to include in the group, add the objects as children of the Group object.</p> <p>Use the Group object's Visible property to indicate whether you want the group of user interface objects visible or hidden.</p> <p>For information about setting properties for the Group object, see “Group Properties” on page 81.</p>
RadioButtonGroup	<p>Creates a container that holds a group of radio buttons.</p> <p>To specify the radio buttons to include in the group, add RadioButton objects as children of the RadioButtonGroup object.</p> <p>The RadioButtonGroup object does not have any properties.</p>
Separator	<p>Displays a horizontal line that you can use to separate blocks of content.</p> <p>For information about setting properties for the Separator object, see “Separator Properties” on page 82.</p>
Spacer	<p>Displays blank space that you can use to create extra padding between user interface objects.</p> <p>For information about setting properties for the Spacer object, see “Spacer Properties” on page 82.</p>

Group Properties

Property	Description
Name	Name you assign the group in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Visible	Whether you want the user interface objects in the group to be visible.

Separator Properties

<u>Property</u>	<u>Description</u>
Color	Color of the separator line.
Height	Height of the separator line.
Name	Name you assign the separator in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the separator line's left edge to its parent object's inner X position.
Position Y	Distance from the separator line's top edge to its parent object's inner Y position.
Visible	Whether you want the separator line to be visible.
Width	Width of the separator line.

Spacer Properties

<u>Property</u>	<u>Description</u>
Height	Height of the spacer object.
Name	Name you assign the spacer object in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the spacer object's left edge to its parent object's inner X position.
Position Y	Distance from the spacer object's top edge to its parent object's inner Y position.
Visible	Whether you want the spacer object to be visible.
Width	Width of the spacer object.

Objects to Use for Dialogs

The following table provides descriptions of the user interface object that you can use to create alert dialogs.

Object	Description
AlertDialog	<p>Displays a small pop-up that you can use to:</p> <ul style="list-style-type: none"> ■ Present information to the user. ■ Interact with the user by presenting a simple question, for example, a question requiring a “yes” or “no” answer. <p>You must add at least one AlertDialogButton child object for the AlertDialog object.</p> <p>For information about setting properties for the AlertDialog object, see “AlertDialog Properties” on page 83.</p>
AlertDialogButton	<p>Displays a button to include in an AlertDialog.</p> <p>For information about setting properties for the AlertDialogButton object, see “AlertDialogButton Properties” on page 83.</p>

AlertDialog Properties

Property	Description
Class Name	<p>Name of the class to generate for the alert dialog and to which you can add logic for the alert dialog. You must specify this property.</p> <p>The generated class extends the NativeUI class for the element, which is <code>com.softwareag.mobile.runtime.nui.nUIAlertDialog</code>.</p> <p>You can specify the same value for Class Name for AlertDialog objects. Mobile Development generates only one class.</p> <p>For more information, see “Logic for a Dialog” on page 121.</p>
Header Text	<p>Text you want displayed in the header area of the dialog.</p> <p>Leave this property blank if you do not want text in the header.</p>
Text	<p>Text to display in the dialog. You must specify this property.</p>

AlertDialogButton Properties

Property	Description
Id	<p>Identifier you assign to the button. You must specify this property.</p>

Property	Description
Text	Text you want displayed on the button.

Objects to Use for Tables

The following table provides descriptions of user interface objects that you can use to define tables that you want to display in an application's view.

Object	Description
DynamicTablecell	<p>Specifies a method that executes at run time to populate a table cell. A <code>DynamicTablecell</code> object is the child of a <code>TableRow</code> object.</p> <p>For information about setting properties for the <code>DynamicTablecell</code> object, see “DynamicTablecell Properties” on page 85.</p>
DynamicTablerow	<p>Specifies a method that executes at run time to dynamically define the layout for the table and populate the table. A <code>DynamicTablerow</code> object is the child of a <code>Table</code> object.</p> <p>For information about setting properties for the <code>DynamicTablerow</code> object, see “DynamicTablerow Properties” on page 85.</p>
Table	<p>Displays a table.</p> <p>To specify the rows in the table, add one or more <code>TableRow</code> objects or a single <code>DynamicTablerow</code> object as children of the <code>Table</code> object.</p> <p>For information about setting properties for the <code>Table</code> object, see “Table Properties” on page 85.</p>
TableButton	<p>Displays a table that contains other objects and that acts as a button.</p> <p>For information about setting properties for the <code>TableButton</code> object, see “TableButton Properties” on page 86.</p>
TableCell	<p>Adds a cell to a table row. A <code>TableCell</code> object is the child of a <code>TableRow</code> object.</p> <p>To specify user interface objects that you want to display in the table cell, add the objects as children of the <code>TableCell</code> object.</p> <p>For information about setting properties for the <code>TableCell</code> object, see “Tablecell Properties” on page 86.</p>

Object	Description
TableRow	<p>Adds a single row to a table. A <code>TableRow</code> object is the child of a <code>Table</code> object.</p> <p>A child of a <code>Table</code> object. Use to add a single row to the table.</p> <p>To specify the contents of the table row, add one or more <code>TableCell</code> objects or a single <code>DynamicTableCell</code> object as children of the <code>TableRow</code> object.</p> <p>For information about setting properties for the <code>TableRow</code> object, see “TableRow Properties” on page 87.</p>

DynamicTableCell Properties

Property	Description
Method Name	<p>Name of a method you code to populate the table cell. For information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see “Logic for a Method Name Property” on page 122.</p>

DynamicTableRow Properties

Property	Description
Method Name	<p>Name of a method you code to populate the table. For information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see “Logic for a Method Name Property” on page 122.</p>

Table Properties

Property	Description
Border Color	Color of the table’s border.
Border Thickness	Thickness of the table’s border.
Inner Height	Usable height of the table in which you can insert child objects.
Inner Width	Usable width of the table in which you can insert child objects.

<u>Property</u>	<u>Description</u>
Inner X	Distance from the table's left edge to where child elements are drawn.
Inner Y	Distance from the table's top edge to where child elements are drawn.
Name	Name you assign the table in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the table's left edge to its parent object's inner X position.
Position Y	Distance from the table's top edge to its parent object's inner Y position.
Rel Widths	Relative widths of the columns in the table. For example, if you specify 25, 25, 50, the table has three columns where the first two each use 25% of the width and remaining column uses 50% of the width.
Spacing Height	Distance between the table rows.
Spacing Width	Distance between the table columns.
Visible	Whether the table is visible or hidden.
Width	Width of the table.

TableButton Properties

<u>Property</u>	<u>Description</u>
Background Color Highlight	Color of the table button when the table button has focus.
Name	Name you assign the table button in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Visible	Whether the table button is visible or hidden.

Tablecell Properties

<u>Property</u>	<u>Description</u>
Background Color	Color of the table cell.
HAlign	Horizontal alignment of the contents in the cell.

Property	Description
HSpan	Number of columns you want the cell to span. For example, when HSpan is set to 2, the cell spans two columns.
Inner Height	Usable height of the cell in which you can insert child objects.
Inner Width	Usable width of the cell in which you can insert child objects.
Inner X	Distance from the cell's left edge to where child elements are drawn.
Inner Y	Distance from the cell's top edge to where child elements are drawn.
VAlign	Vertical alignment of the contents in the cell.
VSpan	Number of rows you want a cell to span. For example, when VSpan is set to 2, the cell spans two rows.

Tablerow Properties

Property	Description
Background Color	Color of the table row.
Height	Height of the table row.

Objects to Use for User Interface Controls

The following table provides descriptions of user interface controls that you can display in an application's view.

Object	Description
Button	Displays a single button that contains a text label. For information about setting properties for the Button object, see "Button Properties" on page 90 .
Button Group	Creates a container that holds a group of buttons. To specify the buttons to include in the group, add Button objects as children of the ButtonGroup object. For information about setting properties for the Button Group object, see "ButtonGroup Properties" on page 91 .

Object	Description
Checkbox	<p>Displays a check box.</p> <p>For information about setting properties for the <code>Checkbox</code> object, see “Checkbox Properties” on page 91.</p>
Container	<p>Creates a container that holds other user interface objects.</p> <p>To specify user interface objects that you want to display in the container, add the objects as children of the <code>Container</code> object.</p> <p>You can set the <code>Container</code> object’s properties to allow scrolling. For example, you might use a container to hold long pieces of text that exceed the viewable area, allowing the user to scroll through the text.</p> <p>For information about setting properties for the <code>Container</code> object, see “Container Properties” on page 92.</p>
DateEntry	<p>Displays a date or time selector control.</p> <p>For information about setting properties for the <code>DateEntry</code> object, see “DateEntry Properties” on page 93.</p>
DropDownListEntry	<p>Displays a drop-down list that contains selection items.</p> <p>To define the items in the drop-down, add one or more <code>StringDropDownListEntryItem</code> objects or a single <code>DynamicDropDownListEntryItem</code> object as children of the <code>DropDownListEntry</code> object.</p> <p>For information about setting properties for the <code>DropDownListEntry</code> object, see “DropDownListEntry Properties” on page 93.</p>
DynamicDisplayObject	<p>Name of a method you code to display a user interface object. For information about setting properties for the <code>DynamicDisplayObject</code> object, see “DynamicDisplayObject Properties” on page 94.</p>
DynamicDisplayObjectArray	<p>Name of a method you code to display an array of user interface objects. For information about setting properties for the <code>DynamicDisplayObject</code> object, see “DynamicDisplayObject Properties” on page 94.</p> <p>For information about setting properties for the <code>DynamicDisplayObjectArray</code> object, see “DynamicDisplayObjectArray Properties” on page 94.</p>

Object	Description
DynamicDropDownListEntryItem	<p>Specifies a method that executes at run time to provide the list of entries to display in the drop-down list. A <code>DynamicDropDownListEntryItem</code> object is the child of a <code>DropDownListEntry</code> object.</p> <p>For information about setting properties for the <code>DynamicDropDownListEntryItem</code> object, see “DynamicDropdownlistEntryItems Properties” on page 94.</p>
Entry	<p>Displays a text entry box.</p> <p>You can set the <code>Entry</code> object’s <code>Input Type</code> property to:</p> <ul style="list-style-type: none"> ■ Restrict the user input to alphanumeric characters or only numbers. ■ Mask the field’s contents, making the field suitable for a user to enter passwords or personal identifier numbers (PINs). <p>For information about setting properties for the <code>Entry</code> object, see “Entry Properties” on page 95.</p>
Image	<p>Displays an image.</p> <p>To specify the image that you want to display, set the <code>Image</code> object’s <code>Image</code> property.</p> <p>You can use an <code>Image</code> object as a button if you add a <code>TriggerListener</code> object as a child object.</p> <p>For information about setting properties for the <code>Image</code> object, see “Image Properties” on page 95.</p>
NavButton	<p>Displays a button that an application uses for navigation.</p> <p>For information about setting properties for the <code>NavButton</code> object, see “NavButton Properties” on page 96.</p>
Pagination	<p>Adds objects that a user selects to display the next or previous page of list items in a <code>ListView</code>.</p> <p>For information about setting properties for the <code>Pagination</code> object, see “Pagination Properties” on page 97.</p>

<u>Object</u>	<u>Description</u>
ProgressAnim	<p>Displays an animated status indicator that indicates background activity is in progress.</p> <p>For information about setting properties for the ProgressAnim object, see “ProgressAnim Properties” on page 97.</p>
RadioButton	<p>Displays a single radio button that uses two states, selected or cleared.</p> <p>For information about setting properties for the RadioButton object, see “RadioButton Properties” on page 98.</p>
SearchEntry	<p>Displays a search entry field.</p> <p>For information about setting properties for the SearchEntry object, see “SearchEntry Properties” on page 99.</p>
StringDropDownListEntryItem	<p>Adds a single entry to a drop-down list. A StringDropDownListEntryItem object is the child of a DropDownListEntry object.</p> <p>For information about setting properties for the StringDropDownListEntryItem object, see “StringDropdownlistEntry Properties” on page 99.</p>
Textfield	<p>Displays plain text in a label or for a block of text.</p> <p>For information about setting properties for the Textfield object, see “Textfield Properties” on page 99.</p>
WebViewElement	<p>Use to display rich Web content from a local file or by specifying the URL of the content to display.</p> <p>For information about setting properties for the WebViewElement object, see “WebViewElement Properties” on page 100.</p>

Button Properties

<u>Property</u>	<u>Description</u>
Create On Condition	Whether to create the button at run time.
Font Color	Color of the text on the button.
Font Size	Size of the font to use for the text on the button.

Property	Description
Font Style	How the text should be formatted, for example, bold, italic, or underlined.
HAlign	Horizontal alignment of the text on the button.
Name	Name you assign the button in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the button's left edge to its parent object's inner X position.
Position Y	Distance from the button's top edge to its parent object's inner Y position.
Text	Text to display on the button.
Visible	Whether the button is visible or hidden.
Width	Width of the button.

ButtonGroup Properties

Property	Description
Name	Name you assign the button group in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.

CheckBox Properties

Property	Description
Create On Condition	Whether to create the check box at run time.
Font Color	Color of the text for the check box.
Font Size	Size of the font to use for the check box text.
Font Style	How the text should be formatted, for example, bold, italic, or underlined.
HAlign	Horizontal alignment of the text.
Name	Name you assign the check box in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the check box's left edge to its parent object's inner X position.

<u>Property</u>	<u>Description</u>
Position Y	Distance from the check box's top edge to its parent object's inner Y position.
Text	Text to display for the check box.
Visible	Whether the check box is visible or hidden.
Width	Width of the check box.

Container Properties

<u>Property</u>	<u>Description</u>
Create On Condition	Whether to create the container at run time.
Height	Height of the container.
HScrollable	Whether you want to allow horizontal scrolling in the container.
Inner Height	Usable height of the container in which you can insert child objects.
Inner Width	Usable width of the container in which you can insert child objects.
Inner X	Distance from the container's left edge to where child elements are drawn.
Inner Y	Distance from the container's top edge to where child elements are drawn.
Name	Name you assign the container in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the container's left edge to its parent object's inner X position.
Position Y	Distance from the container's top edge to its parent object's inner Y position.
Visible	Whether the container is visible or hidden.
VScrollable	Whether you want to allow vertical scrolling in the container.
Width	Width of the container.

DateEntry Properties

Property	Description
Create On Condition	Whether to create the date or time selector control at run time.
Date Format	Format in which to display the date or time.
Name	Name you assign the date or time selector control in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the date or time selector control's left edge to its parent object's inner X position.
Position Y	Distance from the date or time selector control's top edge to its parent object's inner Y position.
Visible	Whether the date or time selector control is visible or hidden.
Width	Width of the date or time selector control.

DropDownListEntry Properties

Property	Description
Create On Condition	Whether to create the drop-down list at run time.
Font Size	Size of the font to use for the text for the drop-down list.
Font Style	How the text should be formatted, for example, bold, italic, or underlined.
Name	Name you assign the drop-down list in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the drop-down list's left edge to its parent object's inner X position.
Position Y	Distance from the drop-down list's top edge to its parent object's inner Y position.
Visible	Whether the drop-down list is visible or hidden.
Width	Width of the drop-down list.

DynamicDisplayObject Properties

<u>Property</u>	<u>Description</u>
Create On Condition	Whether to create the user interface object at run time.
Method Name	Name of a method you code to display a user interface object For information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see “Logic for a Method Name Property” on page 122.
Name	Name you assign the user interface object in the Outline Editor for your own reference purpose. This name does not appear in the application’s user interface.
Visible	Whether the user interface object is visible or hidden.

DynamicDisplayObjectArray Properties

<u>Property</u>	<u>Description</u>
Create On Condition	Whether to create the array of user interface objects at run time.
Method Name	Name of a method you code to display an array of user interface objects. For information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see “Logic for a Method Name Property” on page 122.
Name	Name you assign the array of user interface objects in the Outline Editor for your own reference purpose. This name does not appear in the application’s user interface.
Visible	Whether the array of user interface objects is visible or hidden.

DynamicDropdownlistEntryItems Properties

<u>Property</u>	<u>Description</u>
Method Name	Name of a method you code to populate the entries in the drop-down list. For more information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see “Logic for a Method Name Property” on page 122.

Entry Properties

Property	Description
Context Key	Name of a context key you want to reference using syntax such as <code>\${CONTEXT_key}</code> . By default, Mobile Development saves the value you enter in the control to a context that is available within the lifetime of the application for the key you specify.
Create On Condition	Whether to create the text entry box at run time.
Font Size	Size of the font to use for the text for the text entry box.
Font Style	How the text should be formatted, for example, bold, italic, or underlined.
Hint	Text you want displayed when a user hovers over text entry box to provide information about what a user can specify in the entry field. Leave this property blank if you do not want to provide hint text.
Input Type	Type of input a user can supply in the text entry box. For example, you might specify <code>text</code> , <code>textPassword</code> , or <code>number</code> .
Lines	Number of lines to display in the text entry box. This is the number of lines into which a user can type information.
Name	Name you assign the text entry box in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the text entry box's left edge to its parent object's inner X position.
Position Y	Distance from the text entry box's top edge to its parent object's inner Y position.
Text	Text to display in the text entry box.
Visible	Whether the text entry box is visible or hidden.
Width	Width of the text entry box.

Image Properties

Property	Description
Create On Condition	Whether to display the image at run time.
HAlign	Horizontal alignment of the image.

Property	Description
Image	Image to display. <hr/> Note: The image must be one of your application resources. For more information about resources, see “Defining Resources for the Mobile Project” on page 31 . <hr/>
Name	Name you assign the image in the Outline Editor for your own reference purpose. This name does not appear in the application’s user interface.
Position X	Distance from the image’s left edge to its parent object’s inner X position.
Position Y	Distance from the image’s top edge to its parent object’s inner Y position.
Scale to Parent Height	Whether you want the image to be scaled vertically to match the height of its parent object. Select the Select to Parent Height check box if you want the image scaled to the parent object’s height.
Scale to Parent Width	Whether you want the image to be scaled horizontally to match the width of its parent object. Select the Select to Parent Width check box if you want the image scaled to the parent object’s width.
Visible	Whether the image is visible or hidden.

NavButton Properties

Property	Description
Create On Condition	Whether to create the button at run time.
Icon	Icon image to display on the button. <hr/> Note: The icon must be one of your application resources. For more information about resources, see “Defining Resources for the Mobile Project” on page 31 . <hr/>
Name	Name you assign the button in the Outline Editor for your own reference purpose. This name does not appear in the application’s user interface.
Text	Text to display on the button.

Property	Description
Type	Type of button. The types are: <ul style="list-style-type: none"> ■ BACK This type is placed on the left of the view header. ■ DEFAULT This type is placed on the right of the view header.
Visible	Whether the button is visible or hidden.

Pagination Properties

Property	Description
Max Number Per Page	Maximum number of list items to display on a single page of a ListView.
Next Page Template	<p>Template that defines an object to display at the bottom of the list of items in a ListView if more list items are available on subsequent pages. A user selects the object to display the next set of results in a ListView.</p> <p>Specify a template that you previously defined. The template should customize an object like a Button object.</p>
Previous Page Template	<p>Template that defines an object to display at the top of the list of items in a ListView if more list items are available on previous pages. A user selects the object to display the previous set of results in a ListView.</p> <p>Specify a template that you previously defined. The template should customize an object like a Button object.</p>

Important! The templates that you specify for the **Next Page Template** and **Previous Page Template** properties must be for an object that a user can trigger. Do *not* use a template for an object like a **Textfield**, which a user use to trigger an action.

ProgressAnim Properties

Property	Description
Create On Condition	Whether to create the object at run time.

<u>Property</u>	<u>Description</u>
Name	Name you assign the object in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the object's left edge to its parent object's inner X position.
Position Y	Distance from the object's top edge to its parent object's inner Y position.
Visible	Whether the object is visible or hidden.

RadioButton Properties

<u>Property</u>	<u>Description</u>
Create On Condition	Whether to create the radio button at run time.
Font Color	Color of the text displayed for the radio button.
Font Size	Size of the font to use for the text displayed for the radio button.
Font Style	How the text should be formatted, for example, bold, italic, or underlined.
HAlign	Horizontal alignment of the radio button.
Name	Name you assign the radio button in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the radio button's left edge to its parent object's inner X position.
Position Y	Distance from the radio button's top edge to its parent object's inner Y position.
Text	Text to display for the radio button.
Visible	Whether the radio button is visible or hidden.
Width	Width of the radio button.

SearchEntry Properties

Property	Description
Context Key	Name of a context key you want to reference using syntax such as <code>#{CONTEXT_key}</code> . By default, Mobile Development saves the value you enter in the control to a context that is available within the lifetime of the application for the key you specify.
Create On Condition	Whether to create the search entry box at run time.
Hint	Text you want displayed when a user hovers over search entry box to provide information about what a user can specify in the entry field. Leave this property blank if you do not want to provide hint text.
Name	Name you assign the search entry box in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the search entry box's left edge to its parent object's inner X position.
Position Y	Distance from the search entry box's top edge to its parent object's inner Y position.
Visible	Whether the search entry box is visible or hidden.
Width	Width of the search entry box.

StringDropDownlistEntry Properties

Property	Description
Item	Single item to display in the parent drop-down list object.

Textfield Properties

Property	Description
Create On Condition	Whether to display the text field at run time.
Font Color	Color of the text.
Font Size	Size of the font to use for the text.

<u>Property</u>	<u>Description</u>
Font Style	How the text should be formatted, for example, bold, italic, or underlined.
HAlign	Horizontal alignment of the text.
Max Lines	Maximum number of lines of text to display in the text field when line-wrapped. The value you specify for Max Lines must be greater than or equal to the value you specify for Min Lines .
Min Lines	Minimum number of lines of text to display in the text field when line-wrapped. The value you specify for Min Lines must be less than or equal to the value you specify for Max Lines .
Name	Name you assign the text field in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Position X	Distance from the text field's left edge to its parent object's inner X position.
Position Y	Distance from the text field's top edge to its parent object's inner Y position.
Render Type	Whether to display the text as plain text or hyperlinked.
Text	Text to display.
Visible	Whether the text field is visible or hidden.
Width	Width of the text field.

WebViewElement Properties

<u>Property</u>	<u>Description</u>
Create On Condition	Whether to display the Web content at run time.
File	File that contains the Web content to display.
Height	Height to use for the object containing the Web content.
HScrollable	Whether you want to allow horizontal scrolling of the Web content.
Name	Name you assign the object in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.

Property	Description
Position X	Distance from the object's left edge to its parent object's inner X position.
Position Y	Distance from the object's top edge to its parent object's inner Y position.
Url	URL to the Web content to display.
Visible	Whether the object containing the Web content is visible or hidden.
VScrollable	Whether you want to allow vertical scrolling of the Web content
Width	Width to use for the object containing the Web content.

Objects to Use for Content Providers

The following table provides descriptions of the user interface objects that you use to specify the content to display in a `ListView` object.

Object	Description
<code>DataBinding</code>	<p>Defines how to bind data from the data source to an object in the user interface. A <code>DataBinding</code> object is the child of a <code>ContentProvider</code> object.</p> <p>For information about setting properties for the <code>DataBinding</code> object, see “DataBinding Properties” on page 102.</p>
<code>DynamicDataSource</code>	<p>Identifies a method that the application executes to obtain the data to display in the <code>ListView</code> object. You are responsible for providing the logic for the method.</p> <p>For information about setting properties for the <code>DynamicDataSource</code> object, see “DynamicDataSource Properties” on page 102.</p>
<code>ContentProvider</code>	<p>Specifies the content you want listed in a <code>ListView</code> object.</p> <p>For information about setting properties for the <code>ContentProvider</code> object, see “ContentProvider Properties” on page 102.</p>
<code>JSONDataSource</code>	<p>Specifies a service to invoke to provide the data to display in a <code>ListView</code> object. A <code>JSONDataSource</code> object is the child of a <code>ContentProvider</code> object.</p> <p>For information about setting properties for the <code>JSONDataSource</code> object, see “JSONDataSource Properties” on page 103.</p>

DataBinding Properties

Property	Description
Control	Name of the user interface object to which to bind data from the data source.
Data Source	Name of the <code>DynamicDataSource</code> or <code>JSONDataSource</code> object to use to obtain the data to bind to the control.
Expression	Expression that identifies the data to bind to the control specified by the <code>Control</code> property.

DynamicDataSource Properties

Property	Description
Method Name	Name of a method you code to populate a <code>ListView</code> . For information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see “Logic for a Method Name Property” on page 122 . The method you code must return an instance of <code>IListViewDatasource</code> , which is in the <code>gen/api-src</code> folder in the <code>com.softwareag.mobile.runtime.toolkit.delegates</code> package.
Name	Name you assign the data source in the Outline Editor for your own reference purpose.

ContentProvider Properties

Property	Description
Expression	Expression that identifies the data to obtain from the data source defined by the <code>List Data Source</code> property.
List Data Source	Name of a <code>DynamicDataSource</code> or <code>JSONDataSource</code> object that you defined as a child of the <code>ContentProvider</code> object.

Property	Description
Template	<p>Name of a Template object that you previously defined.</p> <p>At run time, the data obtained from the data source is formatted and displayed in this user interface object. Specify a Template object that is based on a user interface object that can display the list of information, such as a Table object.</p> <p>For more information about templates, see “Using Templates to Define Custom Objects for a Mobile Project” on page 68 and “Using a Content Provider to Populate a ListView” on page 60.</p>

JSONDataSource Properties

Property	Description
Name	Name you assign the object in the Outline Editor for your own reference purpose. This name does not appear in the application’s user interface.
Rest Method	<p>Method to invoke to provide the data for the ListView object.</p> <p>Select a method that you have defined in the model. You define the method using a Method object that is a child of the Resource object in the Services part of the model.</p>

Objects to Use for Event Listeners

The following table provides descriptions of user interface objects that you can include in the model to add listeners to an application. The objects generate a user-initiated event when the user performs the action, such as when a user presses a button.

Object	Description
GainFocusListener	<p>Generates a user-initiated event when a user interface object gains focus.</p> <p>For example, if you want to generate a user-initiated event when an entry field gains focus because the user selects the entry field, add a GainFocusListener object as a child of the Entry object.</p> <p>The GainFocusListener object does not have any properties.</p>

Object	Description
LoseFocusListener	<p>Generates a user-initiated event when a user interface object loses focus.</p> <p>For example, if you want to generate a user-initiated event when an entry field loses focus because the user stops selecting the entry field by selecting another user interface control, add a <code>LoseFocusListener</code> object as a child of the <code>Entry</code> object.</p> <p>The <code>LoseFocusListener</code> object does not have any properties.</p>
PostEditListener	<p>Generates a user-initiated event after a user edits an object.</p> <hr/> <p>Note: In the case of keyboard entry, a new event is generated for each character typed or deleted.</p> <hr/> <p>For example, if you want to generate a user-initiated event after a user types an entry in a <code>SearchEntry</code> object, add a <code>PostEditListener</code> object as a child of the <code>SearchEntry</code> object.</p> <p>The <code>PostEditListener</code> object does not have any properties.</p>
PreEditListener	<p>Generates a user-initiated event when a user is about to edit an object.</p> <p>For example, if you want to generate a user-initiated event when a user is about to type text in a <code>SearchEntry</code> object, add a <code>PreEditListener</code> object as a child of the <code>SearchEntry</code> object.</p> <p>The <code>PreEditListener</code> object does not have any properties.</p>
TriggerListener	<p>Generates a user-initiated event after a user uses an object.</p> <p>For example, if you want to generate a user-initiated event after a user presses a <code>Button</code> object, add a <code>TriggerListener</code> object as a child of the <code>Button</code> object.</p> <hr/> <p>Note: If you add a <code>TriggerListener</code> as the child of an <code>Image</code> object, the image acts as a button.</p> <hr/> <p>The <code>TriggerListener</code> object does not have any properties.</p>

Objects to Use for Event Actions

The following table provides descriptions of user interface objects that you include in the model to specify the action you want an application to take when a user-initiated event occurs.

Object	Description
Back	<p>The application displays the previous view when the associated user-initiated event occurs.</p> <p>For information about setting properties for the Back object, see “Back Properties” on page 106.</p> <hr/> <p>Note: This action is only supported if the application uses the Mobile Development <code>TransitionStackController</code>. If the application override the <code>TransitionStackController</code>, this event action will not work properly. For more information about the <code>TransitionStackController</code>, see “About the TransitionStackController” on page 119.</p>
ChangePaneConfiguration	<p>The application switches to a pane configuration you specify when the associated user-initiated event occurs.</p> <p>For information about setting properties for the ChangePaneConfiguration object, see “ChangePaneConfiguration Properties” on page 106.</p>
Delegate	<p>The application executes a method you specify when the associated user-initiated event occurs.</p> <p>For information about setting properties for the Delegate object, see “Delegate Properties” on page 107.</p>
OpenDialog	<p>The application displays the AlertDialog you specify when the associated user-initiated event occurs.</p> <p>For information about setting properties for the OpenDialog object, see “OpenDialog Properties” on page 107.</p>
ToggleVisibility	<p>The application switches between making a user interface object that you specify either visible or hidden.</p> <ul style="list-style-type: none"> ■ If the object is currently visible, the application hides it. ■ If the object is currently hidden, the application makes it visible. <p>An example of using the ToggleVisibility action might be to show a button if a user enters text in a text entry field.</p> <p>For information about setting properties for the ToggleVisibility object, see “ToggleVisibility Properties” on page 107.</p>

Object	Description
Transition	<p>The application transitions to the new view you specify when the associated user-initiated event occurs.</p> <p>For information about setting properties for the Transition object, see “Transition Properties” on page 108.</p>

Back Properties

Property	Description
Back To Root	<p>Back behavior that you want the application to take when the user-initiated event occurs.</p> <ul style="list-style-type: none">■ Clear the check box if you want the application to return to the previous view.■ Select the check box if you want the application to display the first view. <p>When you select the check box, the TransitionStackController opens the first pushed view. For more information about the TransitionStackController, see “About the TransitionStackController” on page 119.</p>

ChangePaneConfiguration Properties

Property	Description
Method Name	<p>Name of a method that performs the change to the new pane configuration. Specifying the Method Name property is optional.</p> <p>When you specify Method Name, Mobile Development generates a method and places code to change the pane configuration in the method you specify. You can then add additional code to the method. For more information, see “Logic for a Method Name Property” on page 122.</p>
On Condition	<p>Whether to perform the change pane configuration action at run time.</p>
Pane Configuration	<p>Pane configuration to which the application switches when the user-initiated event occurs. Specify the name of a PaneConfiguration object you previously defined in the model.</p>

Delegate Properties

<u>Property</u>	<u>Description</u>
Method Name	<p>Name of a method you code and that the application executes when the user-initiated event occurs.</p> <p>For information about the Java sources that Mobile Development generates for the method and how to provide logic for the method, see “Logic for a Method Name Property” on page 122.</p>

OpenDialog Properties

<u>Property</u>	<u>Description</u>
Dialog	<p>Dialog to display when the user-initiated event occurs. Specify the name of a <code>AlertDialog</code> object you previously defined in the model.</p>

ToggleVisibility Properties

<u>Property</u>	<u>Description</u>
Control	<p>The user interface object that you want to make visible or hide in response to the user-initiated event.</p> <hr/> <p>Note: If you identify a control that is within a template, the control within the template is made visible or hidden. If the template is instantiated multiple times, the control in all instantiated templates are made visible or hidden at the same time.</p> <p>If you nest templates inside templates, only identify controls in the top-level template.</p> <hr/>

Transition Properties

Property	Description
Method Name	<p>Name of a method that performs the transition to the new view. Specifying the <code>Method Name</code> property is optional.</p> <p>When you specify <code>Method Name</code>, Mobile Development generates a method and places code to transition to the view in the method you specify. You can then add additional code to the method. For more information, see “Logic for a Method Name Property” on page 122.</p>
On Condition	Whether to perform the transition action at run time.
Pane	Pane in which to display the view.
Style	How to perform the transition to the new view, for example, fade in the new view or have the new view appear.
View	View to transition to when the user-initiated event occurs. Specify the name of a <code>ListView</code> , <code>NavigationView</code> , <code>View</code> , or <code>WebView</code> object you previously defined in the model.

Objects to Use for Templates

Use templates to define custom user interface objects. For more information about templates, see [“Using Templates to Define Custom Objects for a Mobile Project” on page 68](#).

The following table provides descriptions of the user interface objects for using templates.

Object	Description
<code>ListViewElement</code>	<p>Defines a custom object that works like a <code>ListView</code>. When you create templates for <code>ListViewElement</code> objects, you can then reference the templates in a normal view, allowing you to include more than one object in a regular view, each behaving like a <code>ListView</code> object.</p> <p>Add a <code>ListViewElement</code> child object to the <code>Template</code> object if you want to</p> <p>For information about setting properties for the <code>ListViewElement</code> object, see “ListViewElement Properties” on page 109.</p>

Object	Description
Template	<p>Defines a custom user interface object.</p> <p>For information about setting properties for the Template object, see “Template Properties” on page 109.</p>
TemplateReference	<p>Adds the custom user interface object to the user interface.</p> <p>For information about setting properties for the TemplateReference object, see “TemplateReference Properties” on page 110.</p>

ListViewElement Properties

Property	Description
Height	Height you want to use for the object.
List View Header	<p>Template that defines an object to display as the header for the list of data displayed in the ListView.</p> <p>Specify a template that you previously defined. The template should customize an object like a Textfield or Image object.</p>
Separator	<p>Template that defines an object to display between the each list item displayed in the ListView.</p> <p>Specify a template that you previous defined. The template should customize an object like a Separator object.</p>
Width	Width you want to use for the object.

Template Properties

Property	Description
Class Name	<p>Name of the Java class that Mobile Development generates for this template. For more information about the sources that Mobile Development generates for the template and where to add your logic to create a custom user interface object, see “Creating a Template for a Custom Object” on page 68.</p>

TemplateReference Properties

<u>Property</u>	<u>Description</u>
Name	Name you assign the template reference object in the Outline Editor for your own reference purpose. This name does not appear in the application's user interface.
Template	Name of the template for the custom user interface that you want to add to the user interface.

6 Services Object Reference

■ Objects to Use for Services	112
-------------------------------------	-----

Objects to Use for Services

The following table provides a description of the objects you use for services.

Object	Description
Resources	Identifies the server on which the services reside. For information about setting properties for the Resources object, see “Resources Properties” on page 112 .
Resource	Identifies the path to the service you want to use. For information about setting properties for the Resource object, see “Resource Properties” on page 113 .
Method	Identifies the method to execute. For information about setting properties for the Method object, see “Method Properties” on page 113 .
Request	Specifies the content type for the request sent to the REST service. For information about setting properties for the Request object, see “Request Properties” on page 114 .
Parameter	Specifies an input parameter for the service. Use one Parameter object for each input parameter. For information about setting properties for the Parameter object, see “Parameter Properties” on page 114 .
Response	Specifies the format of the response from the REST service. For information about setting properties for the Response object, see “Response Properties” on page 115 .

Resources Properties

Property	Description
Base	Base URI for the REST services you want to use. This part of the URI identifies the server on which the services reside, for example, <code>https://mycompany.com</code> . You can include dynamic URI elements, for example, <code>https://{company}.apps.com/networking</code> , where <code>{company}</code> is a dynamic URI element.

Resource Properties

Property	Description
Path	<p>Path to the service you want to use.</p> <p>You can include dynamic URI elements, for example, <code>record/{objectName}</code>, where <code>{objectName}</code> is a dynamic URI element.</p> <p>You can nest Resource objects under a parent Resource object to provide subpath values.</p> <p>The value you specify for the Path properties of the Resource objects you add to the model are combined with the Base property of the Resources parent object to form the URI to the services you want to use.</p>

Method Properties

Property	Description
Class Name	<p>Name of a Java class that Mobile Development generates in the <code>gen/src</code> folder in the <code>package_name.services.rest</code> package, where <code>package_name</code> is the package name that you specified for your mobile project.</p> <p>The generated Java class contains a constructor that sets the URI and contains the dynamic URI elements as parameters. The dynamic URI elements, if any, are the ones that you specify in the Resources object's Base property and Resource objects' Path properties.</p> <p>The Java class includes an <code>execute</code> method that the mobile application invokes to execute the REST service.</p>
Name	<p>How the request is sent over the network. Select one of the following:</p> <ul style="list-style-type: none"> ■ GET to send the request with the data attached to the request address (URI). ■ POST to send the request with the data sent as a separate data block.

Request Properties

<u>Property</u>	<u>Description</u>
Content Type	<p>Content type for the request sent to the REST service. The application sends the input parameters, which you define using child Parameter objects, in the format you specify. Specify one of the following:</p> <ul style="list-style-type: none">■ NONE if the REST services requires the parameters to be sent in plain text format.■ application/xml if the REST service requires the parameters to be sent in XML format.■ application/json if the REST service requires the parameters to be sent in JSON format.

Parameter Properties

<u>Property</u>	<u>Description</u>
Default Value	<p>Default value for the parameter.</p> <p>If you specify a value, Mobile Development does not create a method parameter for the Parameter object.</p> <p>If you specify a value for Default Value, the Repeating property must be cleared.</p>
Name	<p>Name of the parameter. The value you specify must be unique among all parameters you specify for a specific Request object.</p> <hr/> <p>Note: Do not use <code>postData</code> as a value. The value <code>postData</code> is reserved.</p> <hr/>
Repeating	<p>Whether the parameter contains multiple values. Select the Repeating check box if the parameter contains multiple values.</p> <p>If you select the Repeating check box, you cannot specify a value for the Default Value property.</p>
Style	<p>Whether the parameter is to be added to the header or query string.</p> <ul style="list-style-type: none">■ HEADER if the parameter is to be added to the header.■ QUERY if the parameter is to be added to the query string.

Response Properties

<u>Property</u>	<u>Description</u>
Accept	<p>Format of the response from the REST service. Specify one of the following:</p> <ul style="list-style-type: none">■ NONE if the REST service response is in plain text format.■ application/xml if the REST service response is in XML format.■ application/json if the REST service response is in JSON format.

7 Creating Application Logic

■ About Adding Application Logic	118
■ About the TransitionStackController	119
■ Logic for a View	120
■ Logic for a Dialog	121
■ Logic to Display and Close a Dialog	122
■ Logic for a Method Name Property	122
■ Logic to Programmatically Set a Property Value at Run Time	123
■ Logic to Respond to a Listener Event	124
■ Logic to Transition to Another View	125
■ Common Methods to Override in Generated Code for the Application	126
■ Common Methods to Override in the Generated Code for a View	127

About Adding Application Logic

When you generate sources for a mobile project, Mobile Development generates logic to handle the display of the views and dialogs in the mobile application. The generated code is based on the model you define in the Outline Editor. For example, for a view you design in the Outline Editor, Mobile Development generates logic to operate the user interface based on the user interface objects you add to a view and the property settings for each object. As a result, you do not need to add code for this type of logic for a mobile project. Instead, you can concentrate on the business logic for your application.

When coding the business logic for your application, add your custom code to the *user space*, that is, the Java classes that Mobile Development generates in the mobile project's src folder.

Caution! Do not add logic to the Java classes in the gen/src or gen/api-src folders. When you generate sources or when you generate sources and API for a mobile project, Mobile Development regenerates all the Java classes in those folders. Changes you make will be lost.

The following tables lists the types of business logic you might want to add to a mobile application.

Type of Logic	Description
Genera	Logic that applies to the entire application. For example, you might want to add logic to respond when a user rotates a device from portrait mode to landscape or vice versa.
View	Business logic for a view. For more information, see “Logic for a View” on page 120 .
Dialog	Business logic for a dialog. For more information, see “Logic to Display and Close a Dialog” on page 122 .
Method Name property	Logic for a Method Name property. Several user interface objects have a Method Name property where you can specify a method to invoke at run time. For example, you specify a method name for the <code>DynamicDropDownListEntryItem</code> object to identify the method the application executes at run time to populate the parent <code>DropDownListEntry</code> object. For information, see “Logic for a Method Name Property” on page 122 .

Type of Logic	Description
Property value	<p>Logic to programmatically set a property.</p> <p>Many properties allow you to specify a method name as the property value. At run time, the application executes the method to determine the property value. For example, you might want to programmatically determine the color to use for the BackgroundColor property for a TableRow object. To do so, specify the name of the method that sets the color as the value of the BackgroundColor property. For information, see “Logic to Programmatically Set a Property Value at Run Time” on page 123.</p>
Event handling	<p>Logic to respond to a user-initiated event.</p> <p>For more information, see “Logic to Respond to a Listener Event” on page 124.</p>
View transitions	<p>Logic to transition to another view.</p> <p>For more information, see “Logic to Transition to Another View” on page 125</p>
Templates	<p>Logic for templates to define custom user interface objects.</p> <p>For more information, see “Creating a Template for a Custom Object” on page 68.</p>

About the TransitionStackController

Mobile Development provides a Java class named `TransitionStackController.java` in the `gen.api-src/com.softwareag.mobile.runtime.toolkit.ui.controller` package. Unless you override the use of `TransitionStackController.java`, applications you create using Mobile Development use the logic in `TransitionStackController.java` at run time to:

- Add Back buttons to views when the application creates a view.
- Set the text that displays on Back buttons to display the header text from the previous view.
- Transition to the previous view when a user presses the Back button.

A `TransitionStackController` keeps track of the view displayed in each pane of the application’s window and each `NavButton` object in a `NavigationView` object. It keeps track by pushing view controllers on to a controller stack. For example, as an application transitions the views in a pane from one view to the next, the `TransitionStackController` pushes the new controller onto the controller stack. As a result, when the user presses the Back button, it pops the controller for a view from the controller stack and transitions to that view.

In the Java sources that Mobile Development generates, `TransitionStackController.java` is imported into the abstract controller for each view. For example, if you have a view named `DetailView`, `TransitionStackController.java` is imported into

`AbstractDetailViewController.java`. As a result, you can use the methods of `TransitionStackController.java` to customize the use of the `TransitionStackController` in the logic that you code for a view. For example you can push new controllers on to the controller stack, remove the last controller from the controller stack, or force the controller stack to go to the first controller pushed on the stack. For more information about Java sources generated for your mobile project, see [“Java Sources that Mobile Development Generates” on page 26](#).

If you do not want to use the `TransitionStackController` for a mobile application, you can disable it. However, if you disable the `TransitionStackController` you will need to add application logic to handle transitions back to previous views. To prevent the use of the `TransitionStackController`, override the `createTransitionStackController()` method, which is in `AbstractApplicationController.java` in the `gen.api-src.com.softwareag.mobile.runtime.toolkit` package. To override this method, add the `createTransitionStackController()` method to the `application_nameControllerImpl.java` file that Mobile Development generates in the `src.package_name.ui.controller.impl` package. For example, for a mobile project if you assigned the application the name “MyApp” and the package name “com.mycompany”, you add the `createTransitionStackController()` method to `MyAppControllerImpl.java` in the `src.com.mycompany.ui.controller.impl` package. The following shows sample code. Note that this code sample assumes a view named “MyView” exists.

```
public boolean createTransitionStackController(nUIObject sender,
                                             PaneConfiguration pc,
                                             PaneDefinition pane,
                                             AbstractViewController assignedAVC) {
    if (assignedAVC instanceof MyViewControllerImpl) {
        return false;
    }
    return true;
}
```

Logic for a View

When you generate sources for a mobile project, Mobile Development generates a Java class named `view_nameControllerImpl.java` in the `src.package_name.ui.controller.impl` package. For example, if you assigned the view the name “MyView” and the package name “com.mycompany”, Mobile Development generates `MyViewControllerImpl.java` in the `src.com.mycompany.ui.controller.impl` package.

Add business logic for a view to the `view_nameControllerImpl.java` file.

When initially generated, the `view_nameControllerImpl.java` file contains several methods that are commented out that you might want to uncomment and implement. For more information, see [“Common Methods to Override in the Generated Code for a View” on page 127](#).

Logic for a Dialog

When you add an **AlertDialog** object to the user interface, Mobile Development automatically adds an **AlertDialogButton** as a child object. When you generate sources for the mobile project, Mobile Development generates code to:

- Display the alert dialog with the text you specify in the **AlertDialog** object's **Text** property.
- Close the dialog when the user selects the close **AlertDialogButton** object.

If you want a dialog that simply displays text with a single close button, you do not need to add any further custom code.

If you want to perform additional logic in the dialog or add additional **AlertDialogButton** child objects, you can customize the logic for the dialog.

When you generate sources for a mobile project, Mobile Development generates a Java class for the dialog where you can add your custom logic. The name of the Java class is the name you specify for the **AlertDialog** object's **Class Name** property. Mobile Development generates the class in the `src` folder in the `package_name.ui.dialog` package. For example, if you specify "MyAlertDialog" for the **Class Name** property and assign the mobile project the package name "com.mycompany", Mobile Development generates `MyAlertDialog.java` in the `com.mycompany.ui.dialog` package.

When a user presses a button in the dialog, the `onAlertDialogButtonPressed(final AbstractAlertDialog dialog, final int buttonID)` method in the `iDialogDelegate` Java class is invoked. The `buttonID` is the value of the **Id** property that you specified for the **AlertDialogButton** object in the model. If a dialog contains multiple buttons, your logic can identify the button a user selected and take action based on the specific button the user pressed. The following shows sample logic:

```
public void onAlertDialogButtonPressed(AbstractAlertDialog dialog,
    int buttonId) {
    switch (buttonId) {
    case YMNDialog.YESBUTTON:
        getView().getDialogResult().setText("YES !!!");
        break;
    case YMNDialog.MAYBEBUTTON:
        getView().getDialogResult().setText("MAYBE !!!");
        break;
    case YMNDialog.NOBUTTON:
        getView().getDialogResult().setText("NO !!!");
        break;

    default:
        break;
    }
    dialog.close();
}
```

Logic to Display and Close a Dialog

You can open a dialog in the following ways:

- Open a dialog in response to a listener event.

If you want to open a dialog in response to a listener event, add an `OpenDialog` object to the model and select the name of the dialog you want to open for the `OpenDialog` object's `Dialog` property. For more information, see [“About User-Initiated Events and Listeners”](#) on page 64, [“Adding Listeners for User-Initiated Events”](#) on page 65, and [“OpenDialog Properties”](#) on page 107.

- Open a dialog by creating the dialog class.

To create the dialog class, invoke the class that Mobile Development generates for the dialog, for example, `MyAlertDialog.java`, where `“MyAlertDialog”` is the value you specified for the `AlertDialog` object's `Class Name` property. For example:

```
call new MyAlertDialog(delegate).open();
```

In the code line to create the dialog, `delegate` is an instance of `iDialogDelegate`. By default, all abstract controllers implement `iDialogDelegate`.

- Open a dialog by invoking the `openDialog` method, which is in the `AbstractApplicationController.java` in the `gen/api-src` folder in the `com.softwareag.mobile.runtime.toolkit` package. In this case, pass a generated instance of an `AbstractAlertDialog` to the `openDialog()` method.

For example, you have an application named `MyApplication` and a dialog named `MyAlertDialog`, you might use the following:

```
MyApplicationControllerImpl.openDialog(new MyAlertDialog(dialogDelegate));
```

Note: Only one dialog can be opened at the same time.

A dialog closes when a user presses any button in the dialog.

You can also programmatically close a dialog by invoking the `closeDialog()` method, which is in the `AbstractApplicationController.java` in the `gen/api-src.com.softwareag.mobile.runtime.toolkit` package.

Logic for a Method Name Property

Several user interface objects have a `Method Name` property where you can specify a method to invoke at run time. For example, you specify a method name for the `DynamicDropDownListEntryItem` object to identify the method the application executes at run time to populate the parent `DropDownListEntry` object.

If you specify a method in a `Method Name` property, when you generate sources, Mobile Development generates the method in the abstract controller for the associated view. Mobile Development generates the abstract controller, which is named `Abstractview_nameController.java`, in the `gen/src` folder in the `package_name.ui` package. For

example, if you specify the **Method Name** property for a user interface object in the view named “MyView” and you assign the mobile project the package name “com.mycompany”, Mobile Development generates the method in `AbstractMyViewController.java` in the `com.mycompany.ui` package, which is in the `gen/src` folder.

To add the logic to the method, first you must add the method to the user space, that is, into the `view_nameControllerImpl.java` file in the `package_name.ui.controller.impl` package in the `src` folder. For example, continuing with the previous example, you add the method to `MyViewControllerImpl.java` in the `com.mycompany.ui.controller.impl` package, which is in the `src` folder. After adding the view to the user space, you can add your custom logic.

Caution! Do not add your logic to `Abstractview_nameController.java`. When you generate sources, Mobile Development regenerates this class and your changes will be lost.

Logic to Programmatically Set a Property Value at Run Time

Many properties allow you to specify a method name as the property value. At run time, the application executes the method to determine the property value. For example, you might want to programmatically determine the color to use for the **Background Color** property for a `TableRow` object. To do so, specify the name of the method that sets the color as the value of the **Background Color** property.

At run time, no input parameters will be passed to the method. The output from the method must be a suitable value for the property. For example, if you are using a method for a color, the output must be a value that specifies a color in a suitable format.

Where you place the code for the method depends on whether you specified a relative method name or a fully-qualified method name for the property value.

■ Relative method name

When you specify a relative method name, Mobile Development generates the method in the abstract controller for the associated view. The actions you take to add logic to the method are the same as when you use a **Method Name** property. For more information, see [“Logic for a Method Name Property” on page 122](#).

■ Fully-qualified name

When you specify a fully-qualified method name, the method you specify must exist in a Java class you create. It is recommended that you save the Java class in a location within the project’s `src` folder so that all the code you maintain is in one folder.

Ensure the methods that you create are static so that no instance of the class needs to be in existence.

Logic to Respond to a Listener Event

When you add an event listener object to the model, for example, a `GainFocusListener` object, you are required to add a child event action object that indicates how the application responds when the user-initiated event occurs. For more information, see [“About User-Initiated Events and Listeners” on page 64](#) and [“Objects to Use for Event Actions” on page 104](#).

The table below lists the event action objects and more information about the logic to perform for each action.

Event Action Object	Description
Back	Mobile Development generates the logic to perform this action.
ChangePaneConfiguration	Mobile Development generates logic for this action. However, you can customize the logic. If you specify a value for the <code>ChangePaneConfiguration</code> object's <code>Method Name</code> property, Mobile Development generates the method you specify and places the logic to change pane configurations in the method. If you want to customize the logic, the method into the user space and make your customizations. For more information about how to provide code for the method, see “Logic for a Method Name Property” on page 122 .
Delegate	Mobile Development generates a method you specify in the user space. However, this method does not perform any useful task. You must add the logic you want performed when the user-initiated event occurs. For more information about how to provide code for the method, see “Logic for a Method Name Property” on page 122 .
OpenDialog	Mobile Development generates the logic to perform this action.
ToggleVisibility	Mobile Development generates the logic to perform this action.

Event Action Object	Description
Transition	<p>Mobile Development generates logic for this action. However, you can customize the logic.</p> <p>If you specify a value for the Transition object's Method Name property, Mobile Development generates the method you specify and places the logic to transition to the new view in the method. If you want to customize the logic, copy the method into the user space and make your customizations. For more information about how to provide code for the method, see “Logic for a Method Name Property” on page 122.</p>

Logic to Transition to Another View

Where you add logic to transition from one view to another depends on when you want the application to transition:

- If you want to transition based on a user-initiated event, for example, when a user presses a button in the user interface, you can use the **Transition** event action object.

When you use the **Transition** event action object, you do not need to add any code if you simply want to transition to another view. However, if you want to perform actions before or after the transition, you can add custom code. For more information, see [“Logic to Respond to a Listener Event” on page 124.](#)

- If you want to transition back to the previous view when the user presses the **Back** button, typically you do not need to add logic. The Mobile Development **TransitionStackController** provides logic for transitioning to previous views. For more information, see [“About the TransitionStackController” on page 119.](#)

The following lists circumstances when you need to add logic to transition back to previous views:

- You disabled the **TransitionStackController**.

When you disable the **TransitionStackController**, your application logic must keep track of the views that the application displays and how to transition back.

- You set the **Hide Back Button** property for the view to `true`.

When you hide the view's back button, but have the **TransitionStackController** enabled, you can use the following code to transition back to a previous view:

```
getTransitionStackController().popViewController();
```

- If you want to transition to another view for any other reason, you need to add code to your view to perform the transition.

Unless you take steps to disable the Mobile Development `TransitionStackController`, it is enabled and you can use the methods in `TransitionStackController.java` to transition to the new view. To perform the transition you need to:

- Push the controller for the view to which you are transitioning on the controller stack.
- Transition to the new view.

The following shows sample code to transition to a new view named “`MySecondView`”.

```
getTransitionStackController().pushViewController(new  
MySecondViewControllerImpl());
```

The following shows a sample method you can use to transition from a view named “`MasterView`” to a view named “`DetailsView`”. You would place this code in the `MasterViewControllerImpl` in the `src` folder.

```
public void doTransition()  
    final AbstractViewController target = new DetailsViewController();  
    getTransitionStackController().pushViewController(target);  
}
```

If you disabled the `TransitionStackController`, add the logic that you provide to keep track of the view to which you are transitioning and to perform the transition to the new view.

Common Methods to Override in Generated Code for the Application

When you generate sources for a mobile project, Mobile Development generates a Java class named `application_nameAppControllerImpl.java`, where `application_name` is the name you assigned the application. The `application_nameAppControllerImpl.java` file resides in the `src.package_name.ui.controller.impl` package, where `package_name` is the package name you specified for your mobile project.

When initially generated, the `application_nameAppControllerImpl.java` file contains no executable code. However, Mobile Development does include the set up for methods for which you might want to add code. These methods are commented out. If you want to use them, uncomment the methods and add your custom code. The table below describes these methods.

Note: Besides using the commented out methods that Mobile Development adds to the Java class, you can add other logic for your application to the `application_nameAppControllerImpl.java`.

Method	Description
<code>onCreateWindow()</code>	At run time, this method is executed after the main window for the application is created. Add logic to this method if you want to customize the main window of the application.
<code>OnOrientationChange()</code>	<p>At run time, this method is executed when the user rotates the device and changes the device's orientation from portrait to landscape or vice versa. Add logic to this method that you want performed when a device is rotated, for example, redisplay the user interface for the new orientation.</p> <p>By default, this method updates the dimensions of the panes in the window. If you need to take further action, you can uncomment the <code>OnOrientationChange()</code> method and add your custom logic.</p>

Common Methods to Override in the Generated Code for a View

When you generate sources for a mobile project, Mobile Development generates a Java class named `view_nameControllerImpl.java`, where `view_name` is the name you assigned the view. The `view_nameControllerImpl.java` file resides in the `src.package_name.ui.controller.impl` package, where `package_name` is the package name you specified for your mobile project.

When initially generated, the `view_nameControllerImpl.java` file contains no executable code. However, Mobile Development does include the set up for methods for which you might want to add code. These methods are commented out. If you want to use them, uncomment the methods and add your custom code. The table below describes these methods.

Note: Besides using the commented out methods that Mobile Development adds to the Java class, you can add other logic for your application to the `view_nameControllerImpl.java`.

Method	Description
<code>onTransitionTo()</code>	At run time, this method is executed after the view is created, but before the application transitions to the view. Add logic to this method if you want to customize the view, for example to add or remove controls.
<code>onTransitionFrom()</code>	At run time, this method is executed before transitioning from the current view to another view. Add logic to this method if you want to take action before the view is removed, for example, to save data.

Method	Description
<code>onAlertDialogButtonPressed()</code>	At run time, this method is executed when a user presses a button, (an <code>AlertDialogButton</code> object) in an alert dialog (an <code>AlertDialog</code> object). The method is passed the identifier that you specify in the <code>AlertDialogButton</code> object's <code>Id</code> property so that your logic can determine the button the user selected. Add logic to this method to perform the actions you want to take when a user presses a button.
<code>onBackPressedEvent()</code>	At run time, this method is executed when a user presses the view's Back button. By default, applications you create using Mobile Development use the provided <code>TransitionStackController</code> , and as a result, the default behavior is to transition back to the previous view, if any. Add logic to this method if you want to override this default Back button behavior. For more information about the <code>TransitionStackController</code> , see “About the TransitionStackController” on page 119 .
<code>hidesBackButton()</code>	At run time, this method is executed when the view is about to be displayed for all views except the first view, which does not have a Back button. By default, applications you create using Mobile Development use the provided <code>TransitionStackController</code> , and as a result, the default behavior is that the view is created with a Back button. Use this method to hide the Back button if you do not want the view to have a Back button. <hr/> Note: If your application does not use the <code>TransitionStackController</code> , views do not automatically have a Back button. You manually add a Back button to the view using the <code>addBackButton()</code> method. <hr/>
<code>getBackButtonTitle()</code>	For more information about the <code>TransitionStackController</code> , see “About the TransitionStackController” on page 119 . At run time, this method is executed when adding the Back button to the view. By default, applications you create using Mobile Development use the provided <code>TransitionStackController</code> , and as a result, the default behavior is to use the header text of the previous view on the Back button. Use this method to override the text used on the Back button, for example, to change the text to simply “Back”.
<code>nUIEventCallback()</code>	At run time, this methods receives control when a user-initiated event occurs for any control in the view. Add logic to this method if you want to override event handling.

8 Managing a Project

- Renaming a Mobile Project 130
- Renaming the Application 130
- Changing the Package Name 131

Renaming a Mobile Project

You initially set the name of the mobile project when you create the project using the New Mobile Development Project wizard. If needed, you can change the mobile project name.

Note: If you want to change the name of the application, see [“Renaming the Application” on page 130](#).

To rename the mobile project

- 1 In the Package Explorer, right-click the top-level node for the project and select Refactor > Rename.
- 2 In the New name field, type the new name you want to assign to the mobile project.
- 3 Leave the Update references check box as is. This check box will not affect mobile projects.
- 4 Click OK.

Renaming the Application

You initially set the application when you create the project using the New Mobile Development Project wizard. If you want to change the application name you specified in the wizard, you can do so by updating the Name property for the root application node.

Note: If you want to change the name of the mobile project, see [“Renaming a Mobile Project” on page 130](#).

To rename the application

- 1 Ensure the mobile project is displayed in the Outline Editor. For instructions, see [“Opening the Mobile Development Perspective” on page 15](#).
- 2 In the Model section of the Outline Editor, expand the project so that you can view the top-level child node that represents the root application for the project.
- 3 Select the root application node.
- 4 Type the new name for the application in the Name property, which is displayed in the Properties section of the Outline Editor.
- 5 Save the mobile project.

- 6 Generate sources for the updated mobile project. For instructions, see [“Generating Sources for a Mobile Project”](#) on page 25.

Mobile Development generates a new `new_app_nameAppControllerImpl.java` Java class for the application where `new_app_name` is the new name you assigned to the application.

Mobile Development does *not* remove the `old_app_nameAppControllerImpl.java` Java class, where `old_app_name` is the previous name of the application. Mobile Development retains this file in the event that you previously added custom code to the `old_app_nameAppControllerImpl.java` Java class.

- 7 Update the `new_app_nameAppControllerImpl.java` Java class with any custom code that you added to the `old_app_nameAppControllerImpl.java` Java class.
 - a In the Package Explorer or Navigator view, locate the `src > package > ui > controller > impl` folder, which contains both the `new_app_nameAppControllerImpl.java` and `old_app_nameAppControllerImpl.java` Java classes.
 - b Open both Java classes and copy all custom code from the `old_app_nameAppControllerImpl.java` to `new_app_nameAppControllerImpl.java`.
 - c Save both files.
 - d Delete the `old_app_nameAppControllerImpl.java` Java class.

Changing the Package Name

You initially define the package name for a mobile project when you create the project using the New Mobile Development Project wizard. If you want to change the setting you specified in the wizard, you can do so by updating the **Bundle Id** property for the root application node.

To change the package name for a mobile project

- 1 Ensure the mobile project is displayed in the Outline Editor. For instructions, see [“Displaying a Mobile Project in the Outline Editor”](#) on page 15.
- 2 In the **Model** section of the Outline Editor, expand the project so that you can view the top-level child node that represents the root application for the project.
- 3 Select the root application node.
- 4 In the **Properties** section of the screen, type the new package name in the **Bundle Id** property.
- 5 Save the mobile project.
- 6 In the Outline Editor, right-click and select **Generate Mobile Designer Sources** to regenerate the sources for your project so that your changes are incorporated.

Mobile Development generates new Java classes for the project that use the new package name.

In the project's `gen/src` folder, Mobile Development removes the Java classes that use the old package name, replacing them with Java classes that use the new name.

In the `src` folder, Mobile Development creates new Java classes that use the new package name, but retains all existing Java classes that use the old package name. Mobile Development retains the files with the old package name in the event that you previously added custom code to them.

- 7 For each Java class in the `src` folder that contains custom code that you need added, copy the code from the old Java class files into the corresponding new Java class files.

Look for custom code that you might have added in Java classes that reside in the following, where *old_package* is the old package name:

- `src.old_package.ui.controller.impl`
- `src.old_package.ui.templates`

- 8 Delete the following folders, where *old_package* is the old package name:

- `src.old_package.ui.controller/impl`
- `src.old_package.ui.templates`

Note: Mobile Development does not automatically delete these files because you might have added custom code that you need to copy, as described in the previous step.
