

webMethods Mobile Development Help

Version 10.1

October 2017

This document applies to webMethods Mobile Development Version 10.1 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 © 2014-2017 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://softwareag.com/licenses>.

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://softwareag.com/licenses> 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, license terms, additional rights or restrictions, please refer to "License Texts, Copyright Notices and Disclaimers of Third Party Products". For certain specific third-party license restrictions, please refer to section E of the Legal Notices available under "License Terms and Conditions for Use of Software AG Products / Copyright and Trademark Notices of Software AG Products". These documents are part of the product documentation, located at <http://softwareag.com/licenses> and/or in the root installation directory of the licensed product(s).

Table of Contents

About this Guide.....	9
Document Conventions.....	9
Online Information.....	10
Introduction to Mobile Development.....	11
About Mobile Development.....	12
Other Resources for Mobile Development.....	12
Introduction to the Mobile Development User Interface.....	15
Mobile Development Perspective.....	16
Opening the Mobile Development Perspective.....	17
Displaying a Mobile Project in the Outline Editor.....	17
Displaying a Window, View, or Dialog in the Outline Editor.....	18
Changing How Information is Displayed in the Outline Editor.....	19
Adding Objects to a Mobile Project.....	19
Using the Palette to Add Objects to the Mobile Project.....	20
Removing Objects from a Mobile Project.....	21
Setting Properties in the Outline Editor.....	22
Using Mobile Designer Ant Targets.....	23
Mobile Development Preferences.....	23
Creating and Building a Mobile Application.....	25
Creating a New Mobile Project.....	26
Using Mobile Administrator to Manage and Distribute Mobile Applications.....	29
Configuring App Icons.....	30
Add an App Icon.....	30
Import a high-res icon.....	30
Configuring the Orientations Setting for the Application.....	31
Configuring supported Languages.....	31
Setting the Default Language for the Project.....	32
Generating Sources for a Mobile Project.....	33
Java Sources that Mobile Development Generates.....	34
Building a Mobile Project.....	38
Validating the Application Model.....	39
Testing Your Application with Phoney.....	40
Adding the Mobile Application Logic.....	40
Building the User Interface.....	41
Using sample projects.....	43
Import sample projects from Mobile Designer.....	44
Example: Cocktails app.....	44
Example: ARIS Picture2Model app.....	45

Importing Mobile Projects created with previous version.....	47
Solving compile issues.....	48
Migrating app icons and launch screens.....	48
Migrating other icons.....	49
Defining Resources for the Mobile Project.....	51
The Resource Handler.....	52
Adding additional Text Resources.....	52
Text Resources that Mobile Development Creates for a Project.....	53
Specifying Values for Non-Default Language Text Resources.....	53
Adding static Web Resources.....	54
Adding graphical Assets.....	55
Asset Types.....	55
Using Assets.....	57
Migrating old Mobile Projects to use the new Asset Management.....	57
Using custom Android resources.....	57
Adding graphical assets to a Mobile Project.....	59
Adding Services to a Mobile Project.....	61
Adding RESTful Services.....	62
Adding RESTful IS Services.....	63
Adding Task Client Services.....	64
Adding the SyncComponent Object.....	65
Adding Web Services.....	65
Adding Mobile Administrator Services.....	67
Adding ARIS Services.....	67
Adding a Data Model to a Mobile Project.....	69
Building the User Interface for a Mobile Application.....	73
Basic Structure of the Application User Interface.....	74
Defining Panes for the Application Window.....	75
Adding Views to the Application's User Interface.....	79
Renaming a View.....	80
Adding Content to a View.....	81
Programmatically Populating a ListView.....	83
Using a Content Provider to Populate a ListView.....	86
Using a Content Adapter to Populate a ListView.....	90
Responding to user-initiated events.....	91
Defining Dialogs.....	93
Using Colors, Sizes, Styles and Themes.....	95
Using Colors.....	95
Using Sizes.....	95
Using Font Sizes.....	96
Using Styles.....	96

Using ApplicationStyle to Define Android Styles and Themes.....	96
Using Templates to Define Custom Objects for a Mobile Project.....	98
Creating a Template for a Custom Object.....	98
Using a Template in the Mobile Application User Interface.....	99
Predefined Templates.....	100
Configuring a SideView.....	101
User Interface Object Reference.....	103
User Interface Objects.....	104
Application Node Properties.....	104
Objects to Use for Windows.....	106
Window Properties.....	107
Objects to Use for Panes.....	107
HorizontalSplitter Properties.....	108
PaneConfiguration Properties.....	109
PaneDefinition Properties.....	109
VerticalSplitter Properties.....	109
Objects to Use for Views.....	110
ListView Properties.....	111
NavigationView Properties.....	113
View and SideView Properties.....	113
WebView Properties.....	114
TabView Properties.....	116
Objects to Use for the Layout of the User Interface.....	117
Group Properties.....	118
Separator Properties.....	118
Spacer Properties.....	119
Objects to Use for Dialogs.....	119
AlertDialog Properties.....	120
AlertDialogButton Properties.....	120
Objects to Use for Tables.....	121
DynamicTablecell Properties.....	122
DynamicTablerow Properties.....	122
Table Properties.....	123
TableButton Properties.....	124
Tablecell Properties.....	124
Tablerow Properties.....	125
Objects to Use for User Interface Controls.....	125
Button Properties.....	129
ButtonGroup Properties.....	130
CheckBox Properties.....	130
Container Properties.....	131
DateEntry Properties.....	132
DropDownListEntry Properties.....	133
DynamicDisplayObject Properties.....	133

DynamicDisplayObjectArray Properties.....	134
DynamicDropdownlistEntryItems Properties.....	135
Entry Properties.....	135
Image Properties.....	136
NavButton Properties.....	137
Pagination Properties.....	138
ProgressAnim Properties.....	138
RadioButton Properties.....	139
SearchEntry Properties.....	140
StringDropdownlistEntry Properties.....	141
Tab properties.....	141
Textfield Properties.....	141
WebViewElement Properties.....	142
Objects to Use for Content Providers or Content Adapters.....	143
TemplateDataBinding Properties.....	146
DataBinding Properties.....	147
DynamicDataSource Properties.....	147
ContentProvider Properties.....	147
RESTDataSource Properties.....	148
EntityDataSource Properties.....	149
RESTDataProvider Properties.....	149
JsonDataExpression Properties.....	149
ContentAdapter Properties.....	150
TemplateAssignment Properties.....	150
DataTransformer Properties.....	151
CustomDataComparator Properties.....	151
Objects to Use for Event Listeners.....	151
SwipeListener Properties.....	153
RowSelectionListener Properties.....	153
Objects to Use for Event Actions.....	154
Back Properties.....	156
ChangePaneConfiguration Properties.....	156
Delegate Properties.....	157
OpenDialog Properties.....	157
ToggleVisibility Properties.....	157
Transition Properties.....	158
Objects to Use for Templates.....	158
ListViewElement Properties.....	159
Template Properties.....	160
TemplateReference Properties.....	160
Services Object Reference.....	161
Objects to Use for RESTful Services.....	162
Resources Properties.....	162
Resource Properties.....	163

Method Properties.....	163
Request Properties.....	164
Parameter Properties.....	164
Response Properties.....	165
Objects to Use for Mobile Support - Offline Synchronization.....	165
SyncComponent Properties.....	166
Creating Application Logic.....	167
About Adding Application Logic.....	168
About the TransitionStackController.....	169
Logic for a View.....	170
Logic for a Dialog.....	171
Logic to Display and Close a Dialog.....	172
Logic for a Method Name Property.....	172
Logic to Programmatically Set a Property Value at Run Time.....	173
Logic to Transition to Another View.....	174
Common Methods to Override in the Generated Code for the Application.....	175
Common Methods to Override in the Generated Code for a View.....	177
Common Methods to Override in the Generated Code for a Template.....	179
Using the Sync Component Object.....	179
Managing a Project.....	181
Renaming a Mobile Project.....	182
Renaming the Application.....	182
Changing the Package Name.....	183
Code Snippets.....	185
How to use the DateFormatter and DateTransformer.....	186
Detecting the Current Platform.....	186
Using the Context Key Store to Store and Retrieve Application-wide Settings.....	187
Encoding and Decoding Images with Base64.....	188
Configuring a Session Object with Credentials.....	188
Creating Your Own Operation.....	189
Chaining Multiple Operations.....	189
Using Multipart Requests.....	190
Doing Error Handling for Operations.....	191
Getting the Current GPS Position and Translating it into a Human-readable Location.....	192
Downloading an Image from a Remote Host.....	193
Using a Java Class to Communicate with the Natively Injected Code.....	194
Configure your application to receive Push Notifications.....	195
Configure a Content Adapter to use multiple Templates.....	197

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 (...).

Online Information

Software AG Documentation Website

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

Software AG Empower Product Support Website

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

To submit feature/enhancement requests, get information about product availability, and download products, go to [Products](#).

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

Software AG TECHcommunity

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

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

1 Introduction to Mobile Development

■ About Mobile Development	12
■ Other Resources for Mobile Development	12

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 this Mobile Development help, 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 User's Guide*. This publication describes how to configure and use webMethods Mobile Administrator and its build environment for mobile applications. It explains how to set up an app store from which your end users (for example, employees or customers) can download onto their mobile devices the apps that you have developed yourself or apps from other vendor stores that you have made available for download in your own app store.
- *webMethods Mobile Administrator API Reference*. This reference provides information about how you can access webMethods Mobile Administrator through the REST API.

You can download these files from the Software AG Documentation website.

2 Introduction to the Mobile Development User Interface

■ Mobile Development Perspective	16
■ Opening the Mobile Development Perspective	17
■ Displaying a Mobile Project in the Outline Editor	17
■ Displaying a Window, View, or Dialog in the Outline Editor	18
■ Changing How Information is Displayed in the Outline Editor	19
■ Adding Objects to a Mobile Project	19
■ Using the Palette to Add Objects to the Mobile Project	20
■ Removing Objects from a Mobile Project	21
■ Setting Properties in the Outline Editor	22
■ Using Mobile Designer Ant Targets	23
■ Mobile Development Preferences	23

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. The 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 will be shown when you add or display a mobile object. It shows an outline of the Mobile Project. The Outline Editor consists of the following:

- A **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.

- A **Properties** section, which displays the properties for the node that is selected in the **Model** section. Use the **Properties** section to view and edit properties.
- A palette, which lists all objects that can be added to the Mobile Project. To add an object, you drag it to the appropriate node in the **Model** section.

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.

Note: When you open Software AG Designer for the very first time, a Welcome page is shown. You can also open the Mobile Development perspective by clicking the corresponding link in the Welcome page.

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.

Note: If the above-mentioned views do not yet show a Mobile Project, see "[Creating a New Mobile Project](#)" on page 26.

To display a Mobile Project in the Outline Editor

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

The top-level child node of the project represents the root application for the project.
 2. Do one of the following to display the Mobile Project in the Outline Editor:
 - Double-click the root application node.
 - Select the root application node and press ENTER.
- To display a Mobile Project from the Package Explorer:
 1. Locate the project in the Package Explorer.
 2. 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.
 3. Do one of the following to display the Mobile 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:
 1. 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.

2. Double-click the node for the window, view, or dialog.
- From the Mobile Explorer view:
 1. Expand the project and locate the window, view, or dialog with which you want to work.
 2. 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  **Change to Horizontal Layout** which is shown in the toolbar of the Outline Editor.
- To display the **Model** and **Properties** sections vertically, side-by-side, click  **Change to Vertical Layout** which is shown in the toolbar 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 want to add a service that you use to obtain data that your application displays.

Note: You can either add objects to a Mobile Project as described below or by dragging them from the palette to the **Model** section of the Outline Editor as described in ["Using the Palette to Add Objects to the Mobile Project"](#) on page 20.

To add an object to a Mobile Project

1. Ensure the Mobile Project or specific window, view, or dialog to which you want to add an object is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#) or ["Displaying a Window, View, or Dialog in the Outline Editor" on page 18](#).
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 22](#).

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 33](#).

Using the Palette to Add Objects to the Mobile Project

The palette that is available in the Outline Editor allows you to drag objects to the **Model** section of the Outline Editor and thus to add objects to the Mobile Project. You can hide and show the palette by clicking the arrow that is shown at the top right of the palette.

When you drag an object to the **Model** section, you can only drop it on a node that allows the dragged object as a child object. The mouse pointer will indicate on which node it is possible to drop the object.

Using the palette is especially helpful if you want to add the following objects:

- Table
- TableButton
- Resources
- PaneConfiguration
- TaskClient Services
- Web Services

For these objects, a wizard appears in which you can specify the details for the new node. All additionally required child nodes are then automatically added to the **Model** section. You do not have to add them manually.

Note: For detailed information on how to use the above-mentioned service objects, see ["Adding Services to a Mobile Project" on page 61.](#)

To add an object to a Mobile Project using the palette

1. Ensure the Mobile Project or specific window, view, or dialog to which you want to add an object is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#) or ["Displaying a Window, View, or Dialog in the Outline Editor" on page 18.](#)
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. Make sure that the palette is shown. If the palette is currently hidden, click the arrow () that is shown at the top right of the Outline Editor to display it.
4. Click on one of the header-type nodes in the palette (for example, **Views** or **Controls**) to display the objects in that node. Clicking the same node once more closes the node.
5. Drag the desired object from the palette to the parent node in the **Model** section where you want to add the object as a child node. Watch the icon that is shown on the mouse pointer. When it is possible to drop the object, the mouse pointer shows a plus icon. For example:



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 22.](#)

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 33.](#)

Removing Objects from a Mobile Project

In the Outline Editor, you can remove objects from a Mobile Project.

To remove an object from a Mobile Project

1. Ensure the Mobile Project or specific window, view, or dialog from which you want to remove an object is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#) or ["Displaying a Window, View, or Dialog in the Outline Editor" on page 18.](#)
2. In the **Model** section of the Outline Editor, expand the outline so that you can view the node you want to remove.

- 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 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 33](#).

Setting Properties in the Outline Editor

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

To set the properties for a node

- 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 17](#) or ["Displaying a Window, View, or Dialog in the Outline Editor" on page 18](#).
- In the **Model** section of the Outline Editor, expand the outline so that you view the node for which you want to set properties.

You can also use the following icons:  to expand all child nodes or  to collapse all expanded child nodes. Both icons always apply to the node that is currently selected in the editor. For example, when a **Table** node is currently selected, only the child nodes of the **Table** will be expanded or collapsed.

- Select the node for which you want to set properties.
- In the **Properties** section of the Outline Editor, fill in the properties for the selected node.

For more information about the properties, see ["Logic to Transition to Another View" on page 174](#) and ["Services Object Reference" on page 161](#).

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 173](#).

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 33](#).

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*.

Note: Some actions for which Ant targets are available can be run directly from the Outline Editor. For more information, see ["Building a Mobile Project" on page 38](#).

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.

Mobile Development Preferences

You can set preferences for Mobile Development.

To set the Mobile Development preferences

1. In Software AG Designer, select **Window > Preferences**. If you are working with Mac OS, select **Software AG Designer > Preferences** instead.
2. In the Preferences dialog box, expand the **Software AG** node and then select **Mobile Development**.
3. If required, set the following options:

Option	Description
Mobile Designer	The path to the Mobile Designer installation directory. It is recommended that you check this path after the installation to make sure that Mobile Development uses the correct Mobile Designer installation directory. By default, this path is relative to the Software AG Designer installation directory. If a path is not defined, it is not possible to use Mobile Designer to build or simulate apps. In addition, creating new

<u>Option</u>	<u>Description</u>
	projects will fail. Click Browse if you want to select a different installation directory.
Mobile Support Client	The path to the Mobile Support Client installation directory. Set this path if you want to enable offline data synchronization with Mobile Support Client. Click Browse to select the installation directory. See also " Adding the SyncComponent Object " on page 65. Important If you set this option, you have to regenerate the source code for the desired Mobile Project and you also have to reactivate the handset.
Package Name	The Java package name which will be used as the default value when creating new Mobile Projects. See also " Creating a New Mobile Project " on page 26.
Host Name	The Mobile Administrator host name which will be used as the default value when creating new Mobile Projects and connecting these projects to Mobile Administrator. See also " Creating a New Mobile Project " on page 26.
Ant Target Name	Select the Mobile Designer ANT target that should be executed after each run of the code generation.

4. Click **OK** to save your changes and to close the Preferences dialog box.

3 Creating and Building a Mobile Application

■ Creating a New Mobile Project	26
■ Using Mobile Administrator to Manage and Distribute Mobile Applications	29
■ Configuring App Icons	30
■ Configuring the Orientations Setting for the Application	31
■ Configuring supported Languages	31
■ Generating Sources for a Mobile Project	33
■ Building a Mobile Project	38
■ Validating the Application Model	39
■ Testing Your Application with Phoney	40
■ Adding the Mobile Application Logic	40
■ Building the User Interface	41

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 ["Configuring supported Languages" on page 31](#).
- Adds several universal devices that your application supports. You can add additional devices and/or remove devices after the project is created.

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 29](#).

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 Project**

2. Specify the following settings for the Mobile Project:

- a. In the **Project Name** field, type a name for the new project.

The name that you type will automatically be used as the default value in the **Application Name** field and as a prefix in the **Package Name** field.

- b. Optional. If you want to use an application name other than the default value, type the name in the **Application Name** field.

Mobile Development uses the application name internally, for example, as part of the name of the *application_name* ApplicationControllerImpl.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_name* ApplicationControllerImpl.java, you need to take further action. For more information, see ["Renaming the Application" on page 182](#).

- c. Optional. If you want to use a package name other than the default value, type the name in the **Package Name** field. When specifying the name, be sure to only use characters that are valid in a Java package name.

The default value for the package name is defined in the Mobile Development preferences. If you want to change the default value, you can simply click

the **Configure the default Package Name** link. For more information, see "[Mobile Development Preferences](#)" on page 23.

Mobile Development uses the supplied name as part of the package names for the 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 where you want to save the project:
 - 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**.
3. Select the application template that you want to use. The application template helps to speed up development because default configurations that are ready for customization will be generated. The following templates are provided for selection (see the template descriptions in the wizard for more detailed information):
 - Single pane (default)
 - Single pane with navigation bar
 - Two panes for tablets
 - Two panes and navigation bar for tablets
4. If you want to manage and distribute your application via webMethods Mobile Administrator, click **Next** to 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 29.

- a. Select the **Use Mobile Administrator** check box.
- b. When a Mobile Administrator host name has been defined in the Mobile Development preferences, it is shown in the **URL** field. If a host name is not shown in this field, you can simply click the **Configure the default host name** link and then define the host name in the preferences. For more information, see "[Mobile Development Preferences](#)" on page 23. Otherwise, type the URL of the Mobile Administrator instance that you want to use to manage and distribute the application.
- c. In the **Username** field, specify the name that you use to log in to Mobile Administrator.

- d. In the **Password** field, specify your password for Mobile Administrator.
- e. Click **Login**.

The authentication token is now requested from the specified Mobile Administrator instance. When the authentication was successful, it is possible to proceed with the remaining options on the wizard page.

- f. 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 from a drop-down list.
 - If you are creating a new Mobile Administrator application, perform the following steps:
 - i. Either keep the value in the **Identifier** field (this is taken from the project name you have specified) or type another 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.

The wizard will make the required call to Mobile Administrator and create the application in Mobile Administrator. If there are issues during the creation, an error message will be shown in the header of the wizard. This feature requires at least Mobile Administrator Version 9.8.0.0.63. Possible error messages are:

Error	Solution
Identifier has already been taken	Specify a different identifier.
Bundle ID has already been taken	<p>The bundle ID identifies an application on the target device and must be unique. Therefore, specify an available bundle ID for the target platform.</p> <p>For iOS, the bundle ID is taken from a specific provisioning profile, which can be found on your target Mobile Administrator instance. For iOS, the</p>

<u>Error</u>	<u>Solution</u>
	bundle ID must also match a given provisioning profile. To find out which bundle IDs are available, log in to your Mobile Administrator instance and check the settings for the provisioning profiles.

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.

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.

Before you can configure a Mobile Project to use Mobile Administrator, perform the following required setup in Mobile Administrator. For more detailed information, see the *webMethods Mobile Administrator User's Guide*.

- Create a Mobile Administrator user account or use an existing one. Ensure the Mobile Administrator user account has the global **Manage Site** permission. You can set this permission in Mobile Administrator on the **Details** page for a user.
- You need one Mobile Administrator application for each Mobile Project. Do one of the following:
 - Either use the New Mobile Development Project wizard to create a new application that will automatically be added to Mobile Administrator,
 - or create an application directly in Mobile Administrator. You can then select this application in the New Mobile Development Project wizard. Make sure that the following minimum application-level permissions are defined: **View and Download Stable Versions** and **Manage Build Jobs**.
- Set up Mobile Designer build nodes if you want to remotely build your project.

Configuring App Icons

An App Icon represents your application and appears on the home screen on your device. It is a visual entry to start your application. Hence, you should create several icons for all screen sizes to ensure that your icons will display properly across the variety of devices and screen resolutions.

To configure your application icons

Open the App Icon Editor. You can open this editor either from the context menu or from the Outline Editor.

- From the context menu:
 1. Select a file in your Mobile Project within the Package Explorer and do a right click.
 2. Select the option **Configure App Icons**.
- From the Outline Editor:
 1. Open the Outline Editor. For more information, see "[Displaying a Mobile Project in the Outline Editor](#)" on page 17.
 2. Select the **Open App Icon Editor** action from the top right toolbar of the editor.

The App Icon Editor shows placeholders for your application icons, sorted by platform, and thus gives a visual overview about your application icons. It also names the size required for each individual icon.

To import an App Icon, you can either drag and drop a single icon into the dedicated placeholder or you can import a single high-resolution icon. After importing an icon, you need to save the editor. Your icons will then be copied into your Mobile Project and renamed accordingly. You will find your application icons within the directory `<project>/gen/assets/icons` sorted by platform. When using a source code management system, make sure to commit this directory.

Add an App Icon

To add new icons, drag and drop an icon from your local file system or from the Package Explorer into the icon placeholder. Be aware that only icons using the PNG format are supported. Make sure that the icon you just dropped matches the required icon size, as there is no scaling using this import approach.

Import a high-res icon

You can also import a single, high-resolution icon to let the editor fill all the icon placeholders with a scaled down version of the imported icon. For this, select the **Import a single high-res icon** from the top right toolbar of the editor. A wizard will be opened

where you can drag and drop an image from your local filesystem. Be aware that only PNG images are supported. After dropping this image, you can preview how the image is scaled down into a variety of sizes at the right pane of the wizard. If you are satisfied with the resulting quality, click the **finish** button to end the import. Optionally, you can select if this icon should be used for every or a particular platform.

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 17](#).
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** if 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 33](#).

Configuring supported Languages

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 32](#).

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 17.
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:

<u>For this property...</u>	<u>Specify...</u>
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 33.

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 53. For information about working with languages that are not the default, see "[Specifying Values for Non-Default Language Text Resources](#)" on page 53.

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 17.
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 33.

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 34.

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.

The generated Java classes contain the current content of the Outline Editor. That is, they contain any changes that have not been saved yet.

Note: Mobile Development also creates .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 53.

To generate sources

- From the Outline Editor:
 1. Display your project in the Outline Editor. For more information, see "[Displaying a Mobile Project in the Outline Editor](#)" on page 17.
 2. In the Outline Editor, right click anywhere and select **Generate Source Code > Application Model**.
- From the Package Explorer:
 1. Locate the project in the Package Explorer.
 2. 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.

<u>Package name in the <code>gen/src</code> folder</u>	<u>Description</u>
<i>package_name</i>	This package contains general model-based Java classes.
<i>package_name</i> .i18n	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 "Configuring supported Languages" on page 31 .
<i>package_name</i> .services.rest	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 61 .
<i>package_name</i> .ui	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</i> .ui.dialog	This package contains Java classes that correspond to the dialogs that you designed in the Outline Editor.
<i>package_name</i> ui.styles	This package contains Java classes corresponding to the definition of every Style in the application model.

Package name in the gen/src folder	Description
<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 98.
<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> data.comparator	<p>This package contains Java classes corresponding to the datasource comparators in the application model. Those classes inherit from <code>com.softwareag.mobile.runtime.toolkit.delegates.IComparator</code> and must be implemented by the developer.</p> <p>The comparator name can be used multiple times within the application model, but will result in a single Java class, named by the comparator name.</p>
<i>package_name</i> data.transformer	<p>This package contains Java classes corresponding to the data transformers in the application model. Those classes inherit from <code>com.softwareag.mobile.runtime.toolkit.delegates.IDataTransformer</code> and must be implemented by the developer.</p> <p>The transformer name can be used multiple times within the application model, but will result in a single Java class, named by the transformer name.</p>
<i>package_name</i> .ui.controller.impl	This package contains Java classes that correspond to the user interface that you designed in the Outline Editor. Mobile Development generates the classes a

Package name in the src folder	Description
	<p>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="646 394 1154 432">■ <i>application_name</i> AppControllerImpl.java <p data-bbox="695 447 1333 617">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="695 638 1373 808">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="646 829 1036 867">■ <i>view_name</i> ControllerImpl.java <p data-bbox="695 882 1317 1052">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="695 1073 1349 1276">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 Abstract<i>view_name</i> Controller.java files, which reside in the gen/src folder in the <i>package_name</i> .ui package.</p> <p data-bbox="646 1297 1328 1402">For more information about the types of logic to add these Java classes, see "About Adding Application Logic" on page 168</p>
<i>package_name</i> .ui.dialog	<p data-bbox="646 1444 1341 1581">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 data-bbox="646 1602 1300 1738">For each dialog you define in the Outline Editor, Mobile Development generates a <i>dialog_name</i> .java class, where <i>dialog_name</i> is the name you assigned the dialog in the Outline Editor.</p> <p data-bbox="646 1759 1328 1856">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>

Package name in the src folder	Description
<i>package_name</i> ui.styles	For each Style you define in the Outline Editor, Mobile Development generates a <code>style_name.java</code> class, where <code>style_name</code> is the name of the Style. Mobile Development generates the Java classes for Styles a single time. You overwrite common methods to change the Style programmatically.
<i>package_name</i> .ui.templates	<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 98.</p> <p>For each template you define in the Outline Editor, Mobile Development generates a <code>template_name.java</code> class, where <code>template_name</code> 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 98.</p>

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. For more information on the build process, see *Using webMethods Mobile Designer*.

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 17.
2. If you have 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, click one of the following toolbar buttons that are available in the Outline Editor:

Button	Description
	<p>Multi-Build. Builds your project for multiple device/language combinations. A dialog appears in which you can specify a version number, select to retain the output build files, and select the language group that is to be used.</p> <p>Note: Even though Mobile Designer allows the use of several language groups, Mobile Development projects only provide the language group "I18N" which contains all configured languages.</p>
	<p>Remote Multi-Build. Only available when the Use Mobile Administrator property has been set for the Mobile Project's root application node. A dialog appears in which you can specify a version number, select the targets, and select to retain the output build files.</p> <p>Note: When you set the above-mentioned Use Mobile Administrator property, you also have to regenerate the source code for the application model. This updates the build files and adds the plug-in that is required for the remote build.</p>
Note:	<p>Instead of using the above toolbar buttons, you can also run the corresponding Ant targets. For more information, see "Using Mobile Designer Ant Targets" on page 23.</p>

Validating the Application Model

You can check whether all of your definitions for the application model are valid. Your model is invalid, for example, if a name you specified contains special characters that are not allowed or if a required property has not yet been defined.

If one or more errors are detected, the number of errors is indicated in a message at the top of the Outline Editor, for example, "9 errors detected". When you move the mouse pointer over this message, a tooltip appears informing you what to have to do to fix the errors.

Mobile Development automatically validates an application model when you save your changes to it. In the case of a name, it even checks the characters while you enter them. However, you can also validate the application model manually at any time you want by clicking a toolbar button as described below.

To validate the application model manually

1. Ensure the Mobile Project is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17.](#)
2. Click  **Validate Application Model** which is shown in the toolbar of the Outline Editor.
3. Check the area to the left of the Outline Editor's toolbar to see whether errors have been detected.

Testing Your Application with Phoney

You can test your application by running it in the Phoney phone simulator. For more information on Phoney, see *Using webMethods Mobile Designer*.

To test your application

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 17.](#)
2. If you have unsaved changes, select **File > Save** to save your project.
3. In the Outline Editor, right-click and select **Generate Source Code > Application Model**. This step is required if the sources for the application model do not yet exist. Otherwise, it is optional.
4. To test the application, click one of the following toolbar buttons that are available in the Outline Editor:

Button	Description
	Runs your application in the Phoney phone simulator.
	Reactivates the last device and then runs your application in the Phoney phone simulator.

Note: Instead of using the above toolbar buttons, you can also run the corresponding Ant targets. For more information, see ["Using Mobile Designer Ant Targets" on page 23.](#)

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 168.

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 74.
- Define the configuration of panes to use for the application's window. For more information, see ["Defining Panes for the Application Window"](#) on page 75.
- 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 79 and ["Adding Content to a View"](#) on page 81.
- 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 ["Responding to user-initiated events"](#) on page 91.
- 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 98.

4 Using sample projects

■ Import sample projects from Mobile Designer	44
■ Example: Cocktails app	44
■ Example: ARIS Picture2Model app	45

Import sample projects from Mobile Designer

Mobile Designer includes a couple of sample projects. Every sample describes how to use a dedicated feature. The Mobile Designer samples are located within the Samples directory in your Mobile Designer installation.

To import a sample project from Mobile Designer

1. Open the import wizard by selecting **File > Import > General > Existing Projects into Workspace**.
2. Select the **Select root directory** option and click **Browse**.
3. In the upcoming dialog, select the Mobile Designer installation and locate the **Samples** directory.
4. Click **Open**.
5. Select the projects you want to import into your workspace.
6. Click **Finish**.

Note: Before running Phoney or doing a multi-build, please make sure to activate a handset first. These samples are not using the Mobile Development runtime API and tooling.

Example: Cocktails app

The cocktails app can be used as a reference or starting point when building a new Mobile Project. It demonstrates how to build complex application logic and uses modern UI concepts without the need of user space code. This example shows how to:

- build a **Tab View**,
- build a **Side View**,
- use a RESTful service to populate a **List View**,
- create Entities from a JSON service response,
- provide graphical assets, and
- use Styles to build a color palette.

To import the Cocktails project

1. Open the import wizard by selecting **File > New > Example > Software AG Mobile Solutions > The Cocktail App**.

2. Optionally you can specify a connection to a Mobile Administrator instance if you want to use Mobile Administrator to manage and distribute your application. See ["Creating a New Mobile Project" on page 26](#).
3. Click **Finish**.

Example:ARIS Picture2Model app

The ARIS Picture2Model application is a starting point for integrating Mobile Suite with ARIS. After importing the example project, its sources will be generated. Furthermore some required libraries will be copied into the Mobile Designer folder to be able to run the app within Phoney. To run (Remote)Multi-Build for this app, some advanced technical pre-requisites need to be fulfilled and are described here .

To import the ARIS Picture2Model project

1. Open the wizard by selecting **File > New > Example > Software AG Mobile Solutions > ARIS Picture2Model**.
2. Optionally you can specify a connection to a Mobile Administrator instance if you want to use Mobile Administrator to manage and distribute your application. See ["Creating a New Mobile Project" on page 26](#).
3. Click **Finish**.

5 Importing Mobile Projects created with previous version

■ Solving compile issues	48
■ Migrating app icons and launch screens	48
■ Migrating other icons	49

Mobile Development supports Mobile Projects, which were built with an older version. This section describes the steps you must do to make your legacy Mobile Project working again.

To import an already existing Mobile Project

1. Select Import file from the main menu of SAG Designer
2. Select General > Existing Projects into Workspace in the upcoming dialog.
3. Click **Next**.
4. Select the root directory of your Mobile Project and click **Finish**.

After the import of the Mobile Project, you must generate the source code of your application.

5. Locate your Mobile Project in the Package Explorer.
6. Right-click the Mobile Project root node and select **Generate Source Code-> Application Model and API**.

Solving compile issues

After generating the source code, it is possible that your application shows compile issues. Those issues occur because of necessary API migration in Mobile Development. To solve them, locate the issues in your Java files and replace them as follows:

Issue	Replacement
<code>boolean AbstractViewController#setVisible(nUIObject, boolean)</code>	<code>void nUIObject#setVisible(boolean)</code>
<code>boolean AbstractViewController#isVisible(nUIObject)</code>	<code>boolean nUIObject#isVisible()</code>
<code>boolean AbstractViewController#forceHide(nUIObject)</code>	<code>void nUIObject#setVisible(boolean)</code>

Migrating app icons and launch screens

Mobile Development supports a graphical way to configure your app icons and launch screens. For details see "[Configuring App Icons](#)" on page 30. To migrate your Mobile Project using the App Icon Editor, do the following:

1. Ensure your application model is opened in the Outline Editor.
2. Locate the toolbar at the right side of the Outline Editor and select **Click to open the App Icon Editor** toolbar icon. Or locate the Mobile Project in your Package Explorer, then

right-click the project node and select **Configure App Icons**. The App Icon Editor will be opened, showing placeholders for every required app icon and launch image.

3. Drag and drop existing icons from <project>/resources/graphics/icons to the dedicated placeholder.
4. Save the App Icon Editor.
5. Delete the old app icons and launch images.

Migrating other icons

Mobile Development supports a definition of icons using an Asset file and the Asset Editor. For detailed information, refer to ["Defining Resources for the Mobile Project" on page 51](#).

Note: The existing procedure to provide icons is still supported. For this reason, this migration step becomes optional but recommended. To migrate the old icons using the Asset Editor, do the following:

1. Locate your icon within <project>/resources/graphics.
2. Create a new asset file by right-clicking your Package Explorer and selecting Mobile Tools > create Asset.
3. In the Asset Editor, enter the name of your icon without the png file extension, then open the **Import Icon** creation page.
4. Drag and drop the icon and its variants to the dedicated placeholders.
5. Save the App Icon Editor.
6. Delete the old icon and all variants within <project>/resources/graphics.

6 Defining Resources for the Mobile Project

■ The Resource Handler	52
■ Adding additional Text Resources	52
■ Adding static Web Resources	54
■ Adding graphical Assets	55
■ Asset Types	55
■ Using Assets	57
■ Migrating old Mobile Projects to use the new Asset Management	57
■ Using custom Android resources	57

The Resource Handler

Each project requires its own resource handler. The resource handler defines all resources to be included into your mobile application, such as graphics, text, icons, and sounds. Mobile Development provides an implementation, named `UniversalResHandler` so that developers do not need to code their own.

The `UniversalResHandler` is responsible for providing resources to the resulting native application as well as to the Phoney runtime, such as:

- Additional Text Resources,
- Static HTML Resources, and
- Graphical Assets.

To run the Resource Handler

1. Execute the **ANT target +Run-Reshandler**, or
2. execute the **ANT target ++Reactivate-Handset**, or
3. do a multi build.

While writing your own resource handler is going to be deprecated, you could still code your own implementation and make the project to take use of it.

4. Ensure the Mobile Project is open in the Outline Editor. For instructions, see "[Displaying a Mobile Project in the Outline Editor](#)" on page 17.
5. Select the top-level child node of the project, which is the root application node.
6. Modify the **Res Handler** property in the **Properties** section of the Outline Editor.
 - a. Enter the fully-qualified name of your implementation, e.g.: `reshandler/my/fully/qualified/class.java`.
 - b. The custom resource handler implementation must be added to the `<project>/reshandler` directory.

Adding additional Text Resources

In addition to the default project language, it is possible to provide more languages including localization. For detailed information on how to create a Language and setup the default Language, refer to "[Configuring supported Languages and Internationalization](#)" on page 31.

Text Resources that Mobile Development Creates for a Project

In addition to generated Java sources, 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 26](#), 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 view. 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.

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 53](#).

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 static Web Resources

It is supported to add static Web Resources, such as HTML, JS, CSS files, to your Mobile Project. The UniversalResHandler is responsible for copying those files to the resulting native project or to the Phoney runtime. Therefore, all static web resources must be copied to `<projectName>/resources/www`. The files can be accessed at runtime after running the resource handler. For detailed information, see ["The Resource Handler" on page 52](#).

You can access every file at runtime using the following code snippet:

```
<Java>
final String content =
AbstractApplicationController.getInstance().loadWWWResource("yourFile");
</Java>
```

For example, you can store a HTML file and load its content at runtime. Afterwards pass the files content to a Web View using the method `nUIWebView#setHTMLText(String)`.

Adding graphical Assets

Mobile Applications often use a large number of icons to customize and emphasize the user interface. Icons have to be provided in different sizes in order to support multiple screen resolutions. With Mobile Development, you don't need to take care of different icon sizes for every screen resolution: You only need to define one asset file. The asset defines how an icon looks like and Mobile Development creates different sizes based from this asset definition.

To create a new Asset

1. Open the application model in the Outline Editor.
2. Right-click the Outline Editor, then select **Create new Asset**.

Alternatively

3. locate the Mobile Project in your package explorer.
4. Right-click the Project node in the package explorer, then select **Mobile > Create new Asset**.

After creating a new Asset, the Asset Editor will be opened and you can start to define your Asset. The Asset Editor allows to define an Asset in three different ways. You can either:

- select and customize a Built-In Asset,
- define an Asset based on a SVG,
- or create an Asset Set by importing single PNG icons.

When creating an Asset, developers need to specify a unique, lowercased file name in order to save the Asset. After saving, the Asset file can be located in the <project>/model/resources directory. To open an already existing Asset:

5. Locate the Asset file you like to open.
6. Open the Asset Editor by double-clicking the Asset file or right-click the Asset file, select **Open > Open With** and select the Asset Editor.

Asset Types

Create a built-in Asset

Mobile Development includes an icon library, containing hundreds of icons, which you can customize and use in your application. When creating a new Asset, the Asset Editor already shows the **Built-In Icon** creation page. Here, you can choose icons from different categories. The selected icon is shown in the Preview at the right side of the editor. In addition, it is possible to customize the selected icon as follows:

- change the Size of the icon, or
- change the Padding around the icon, or
- change the Stroke Color of the icon, or
- change the Fill Color of the icon, or
- change a background of the icon and choose between a round-rectangle or a circle background decoration, or
- copy the source of the selected icon to the clipboard.

When saving the icon, Mobile Development automatically creates icons for every screen resolution. Those icons are located within the <project>/gen/.assets/graphics directory. Make sure to commit this directory when working with a source code management system.

Create a custom SVG Asset

Before creating a new SVG Asset, developers need to create a new Asset file and choose the “Define a Path” option in the Editor. Afterwards, you can specify a custom SVG or load an example. The created SVG will be previewed at the right side of the Asset Editor. When saving the icon, Mobile Development automatically creates icons for every screen resolution. Those icons are located within the <project>/gen/.assets/graphics directory. Make sure to commit this directory when working with a source code management system.

Tip: Copy the SVG-document of a Built-In icon to the clipboard from the **Built-In** creation page and start to compose different SVG paths to one SVG Asset using the **Define a Path** creation page of the Asset Editor.

Create an Asset Set

An Asset Set is a collection of single PNG files, which will be grouped under one Asset file. To configure an Asset Set, open the **Import Icon** creation page of the Asset Editor. This page shows placeholders for every platform density. To import an icon, drag and drop the icon into the dedicated placeholder. When saving the Asset Editor, the dropped PNG icons will be copied to <project>/gen/.assets/graphics directory. Make sure to commit this directory when working with a source code management system.

Tips:

- The Asset Set can be used to migrate Mobile Projects – created before 10.1 – to the new Asset Management.
- You can use the Asset Set to define different icons per platform.

Using Assets

Configure an Image element to use an Asset

When working with the Outline Editor, you can specify the name for an Asset within the properties section of the Outline Editor for the following elements:

- Image: **Image** property,
- Tab: **Image** property,
- Navbutton: **Icon** property.

The entered value must be the file name of an Asset including the `.asset` file extension.

Load an Asset at runtime

An Asset can be loaded at runtime using the `com.softwareag.mobile.runtime.toolkit.AbstractApplicationController.loadImage(String)` method. Make sure to pass the file name including the `.asset` file extension. This method returns an instance of `javax.microedition.lcdui.Image`. By loading an asset, the runtime implementation returns the corresponding Asset for the target screen density, so that you do not need to scale the image at runtime. For detailed information, refer to the Mobile Designer documentation.

Migrating old Mobile Projects to use the new Asset Management

Mobile Development still supports the Asset Management from previous version. When importing an older project, every icon will still be loaded. To take use of the new Asset Management, developers need to migrate their icons. Create a new Asset Set with the Asset Editor and delete the old icons afterwards.

Using custom Android resources

The Android platform build has been extended to take custom resources into account, as this is a common source for native code injections. During the Android build, resources will be copied into the `<project>/res` directory and thus must follow Android's resources guidelines. To include custom Android resources (e.g. specific drawables, colors.xml), users need to use the property **android.project.build.resources.drawables** in `targets/_defaults_.xml`, e.g:

```
<property name="android.project.build.resources.drawables"
value="native/graphics/android" />
```

Resources will be copied into the Android project using the following order:

- Copy default files provided by Mobile Designer.
- Copy (and overwrite existing) resources generated by Mobile Development (i.e. when using Styles, Themes).
- Copy (and overwrite existing) resources from **`${android.project.build.resources.drawables}`**.

7 Adding graphical assets to a Mobile Project

Use the +Run-Reshandler 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.

Note: 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-Reshandler**.

The Ant target compiles the resources for the current device.

8 Adding Services to a Mobile Project

■ Adding RESTful Services	62
■ Adding RESTful IS Services	63
■ Adding Task Client Services	64
■ Adding the SyncComponent Object	65
■ Adding Web Services	65
■ Adding Mobile Administrator Services	67
■ Adding ARIS Services	67

Adding RESTful Services

You can use RESTful services as data sources for a mobile application. An application can execute RESTful services to obtain data to display in the application's user interface. Because RESTful services typically return multiple data elements, it is common to use a **ListView** object to display the data you obtain from a RESTful service. For more information, see ["Using a Content Provider to Populate a ListView" on page 86](#).

To add RESTful services

1. Ensure the Mobile Project is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#).
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**.

Note: You can also drag the **Resources** object from the palette of the Outline Editor to the **Services** node. In the palette, you can find this object when clicking the header-type node **RESTful Services**. In this case, a wizard appears in which you can specify the details for the new node. All additionally required child nodes as described below are then automatically added to the **Model** section. You do not have to add them manually. For more information, see ["Using the Palette to Add Objects to the Mobile Project" on page 20](#).

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 162](#).
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 163](#).
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 RESTful 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.

- 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 163](#).

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

- 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 164](#).
- If the RESTful service requires input parameters, perform the following steps for each input parameter:
 - Right-click the **Request** node and select **New Child > Parameter**.
 - 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 164](#).
- 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 165](#).

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 [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.](#)

Adding RESTful IS Services

Adding RESTful IS

will enable you to trigger services hosted on webMethods Integration Server. These services can be used like the normal RESTful services as described in ["Adding RESTful Services" on page 62](#). For detailed information about RESTful IS Services, see the *webMethods Service Development Help*.

This feature requires the installation of webMethodsService Development.

To add RESTful IS services

- Ensure the Mobile Project is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#).
- 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. Drag the **RESTful IS Services** object from the palette of the Outline Editor to the **Services** node. In the palette, you can find this object when clicking the header-type node **RESTful Services**. You can then see it under the heading `webMethods`. If the palette entry **RESTful IS Service** is not visible, the `webMethodsService` Development is not installed.
4. In the resulting dialog box, specify a server connection you like to use services from. If the selection box does not list any server connection, open your Integration Server preferences page and connect to an Integration Server. You can open the Integration Server preferences page by following the link below the selection box. For more information refer to the *webMethods Service Development Help*.
5. After selecting an established server connection, the tree selection below becomes visible. Then dive into the required `webMethods` Package and select the services you like to add to your Mobile Project.
6. Click **OK** to add the child nodes.

Adding Task Client Services

The Task Client services enable you to work with the `webMethods` Task Engine. These services can be used like the normal RESTful services as described in "[Adding RESTful IS Services](#)" on page 63. For detailed information on the available services, see the *webMethods Task Engine API and Service Reference*.

To add Task Client services

1. Ensure the Mobile Project is open in the Outline Editor. For instructions, see "[Displaying a Mobile Project in the Outline Editor](#)" on page 17.
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. Drag the **TaskClient Services** object from the palette of the Outline Editor to the **Services** node. In the palette, you can find this object when clicking the header-type node **RESTful Services**. You can then see it under the heading `webMethods`.
4. In the resulting dialog box, specify the following settings:
 - a. Specify the URL for the machine on which the Task Engine is running.
 - b. Either clear the **Create all built-in services** check box and then select the built-in services that you want to create, or select this check box if you want to create all built-in services.
5. Click **OK** to add the child nodes for the built-in services.

Adding the SyncComponent Object

The SyncComponent object provides the required client implementation for offline data synchronization with the Mobile Support Client. It uses the path to the Mobile Support Client that has been set in the Mobile Development preferences. See also "[Mobile Development Preferences](#)" on page 23.

SyncComponent acts like a service and can be used as a REST method for data sources. It establishes all the connections to the webMethods Integration Server and retrieves/synchronizes data.

For more information on data synchronization with the Mobile Support Client, see *Developing Data Synchronization Solutions with webMethods Mobile Support*.

To add the SyncComponent object

1. Ensure the Mobile Project is open in the Outline Editor. For instructions, see "[Displaying a Mobile Project in the Outline Editor](#)" on page 17.
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. Drag the **SyncComponent** object from the palette of the Outline Editor to the **Services** node. In the palette, you can find this object when clicking the header-type node **RESTful Services**. You can then see it under the heading **Mobile Support**.

Alternatively, right-click the **Services** node and select **New Child > SyncComponent**.
4. Select the **SyncComponent** node, and in the **Properties** section of the Outline Editor, set the properties for the **SyncComponent** node. For more information, see "[Services Object Reference](#)" on page 161.

Adding Web Services

If you want to generate Java classes based on a specific WSDL file, you have to import this WSDL file into your Mobile Project. After the import, you can find the WSDL file in the wsdl subfolder of your Mobile Project.

To generate the Java classes, use **Generate Source Code > Application Model and API**. The Java classes will be placed in the `gen/api-src/com.softwareag.mobile.runtime.toolkit.ws/wsdlFileName` folder.

Several XSD elements and data types are not supported by Mobile Development. These are:

- Unsupported elements:

- `<xsd:simpleContent>`
- `<xsd:union>`
- `<xsd:complexType>` variable elements are assumed to be in `<sequence>`, even if they are not.
- `<port>` as a child of `<definitions>`
- `<extension>` as a child of `<xsd:complexContent>`
- **Unsupported data types:**
 - `base64Binary` (`byte[]`) - use string type instead
 - `integer` (`java.math.BigInteger`) - use int, long or string decimal instead
 - (`java.math.BigDecimal`) - use string instead (float and double would lose precision on financial data, but you might consider them acceptable in other use cases)
 - `dateTime` (`java.util.Calendar`) - use string date instead
 - (`java.util.Date`) - use string instead
 - `map`
 - All classes defined in `com.ibm.ws.webservices.*`

In addition, keep in mind the following when working with web services:

- Mandatory variables for objects are not featured in their constructors at this time.
- `simpleType` restrictions are parsed, but no code is output to enforce them.
- All setups are currently assumed to be SOAP 1.1 compliant.

To add a web service

1. Ensure the Mobile Project is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#).
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. Drag the **WebService** object from the palette of the Outline Editor to the **Services** node. In the palette, you can find this object when clicking the header-type node **Web Services**.

Alternatively, select **File > Import > Other > Import WSDL File** and click **Next**.
4. In the WSDL File Import dialog box, click **Browse** and select the WSDL file that you want to import.
5. Click **Finish**.

Adding Mobile Administrator Services

webMethods Mobile Administrator - as part of Mobile Suite - enables customers to deploy and manage mobile applications. Therefore, Mobile Administrator offers a couple of predefined RESTful Services. They are listed within the *webMethods Mobile Administrator API Reference*. Users can add those services manually to a Mobile Project or use the predefined services from the palette.

To add Mobile Administrator Services

1. Ensure the Mobile Project is open in the **Outline Editor**. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#).
2. In the **Model** section of the **Outline Editor**, expand the project in order to display 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. You can find Mobile Administrator services, when clicking the headertype node **RESTful Services**.

Note: Currently, Mobile Development only supports the Push Register service from within the palette. This request will enable your Mobile Application to receive Push Notifications sent from Mobile Administrator.

4. Drag the **Push Register** object from the palette of the Outline Editor to the **Services** node.

Tip: For more information about Push Notifications, have a look at ["Configure your application to receive Push Notifications" on page 195](#).

Adding ARIS Services

The ARIS Mobile API enables you to receive information about databases and their process models from a remote ARIS instance. ARIS Mobile represents a set of predefined RESTful Services. These services can be used like the normal RESTful services as described in ["Adding RESTful Services" on page 62](#).

To add ARIS Services

1. Ensure the Mobile Project is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#)
2. In the **Model** section of the Outline Editor, expand the project in order to display 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. Drag the ARIS Services object from the palette of the Outline Editor to the **Services** node. In the palette, you can find this object, when clicking the **header-type** node **RESTful Services**. You can then see it under the heading **ARIS Mobile**.
4. In the resulting dialog box, specify the following setting:
 - a. Specify the URL for the machine on which ARIS is running.
 - b. Either clear the **Create all built-in services** check box and select the built-in services you want to create, or select this check box if you want to create all built-in services.
5. Click **OK** to add the child nodes for the built-in services

Tip: There are several request parameters that must be x-www-form-urlencoded, e.g. **key** in the **PostLogin** service. Mobile Development does support configuring parameters to be url encoded by selecting the option **Url Encode** for each specific parameter. To avoid appending sensitive information (e.g. username, password) in plain text to the request URL (and therefore showing up in the server log file), the **Content Type** of the request should be set to **x-www-form-urlencoded**.

Tip: If you want to use the **umc/api/tokens** service for authentication, you need to set the Content Type of the Request in the **PostLogin** service to **NONE**.

9 Adding a Data Model to a Mobile Project

Data Modeling includes the Entity Modeling Language to define a project specific data model and its support within the application model. Developers are able to declare a data model using a Java-like syntax, including syntax highlighting, auto-completion and formatting.

Every EML data model consists of Entities and Enumerations. An Enumeration consists of a list of literals and is represented by a Java class, that contains a public static member for each literal. Each member is identified by an int value (starting with 0). An Entity represents a Java class and may own a number of Variables, Collections and Operations.

- Variable:
 - Integer, Float, Double, Char, Byte, String, Object, Image, Boolean, Long
- Collection:
 - List<Type>, Map<Type, Type>, Stack<Type>
- Operations:
 - void print(String[] args);

Variables can also be used as an Array. The language supports cross-referencing (using another Entity as the type of a Variable) and inheritance (using the "extends" keyword) of Entities to build a hierarchically data model.

A convenient way to model a bi-directional cross-referencing relationship is to use the **@Opposite** annotation. If at runtime the reference to an entity is changed, the generated code takes care to also change the opposite reference of that relationship. Supported Use-Cases:

- 1:1 relation between variables of type Entity
- 1:Many relation between variable of type Entity and List of type Entity
- Many:Many relation between two Lists of type Entity

Two convenience methods will be generated, if the opposite is a List:

- addItemOn<Entity>(final Abstract<Entity> item)
- removeItemOn<Entity>(final Abstract<Entity> item)
- a setter will not be generated in that case

The @Opposite annotation is not supported for:

- Arrays, Enums, Maps, Stacks
- Variables using a simple type (e.g. int, boolean, etc.)

After code generation, Entities and Enumerations can be used as normal Java classes. An Entity must have a name, which will be taken as name for the resulting Java class. As an example, the Entities:

```
package entities;
entity ProcessInstance {
    String id;
    String name;
    int status;
    @Opposite(member=processInstance)
    List taskInstances;
    void print(String[] args)
}
entity TaskInstance {
    String description;
    @Opposite(member=taskInstances)
    ProcessInstance processInstance;
}
```

will result in the Java class `ProcessInstance.java` and `TaskInstances.java`, located in your `src/<yourBundleName>model/entities` package. The resulting Java class will then be generated to include each of the Variables and Collections as Java class properties:

```
public abstract class AbstractProcessInstance implements IEntity {
    private String id;
    private String name;
    private Integer status;
    private Vector taskinstances = new Vector();
    public abstract void print(final String[] args);
    public AbstractProcessInstance setTaskInstancesAtIndex(
        final AbstractTaskInstance item, final int index){
        if (taskinstances != null && taskinstances.size() > index && index >= 0){
            final AbstractTaskInstance oldItem = (AbstractTaskInstance)
                taskinstances.elementAt(index);
            taskinstances.setElementAt(item, index);
            item.setProcessInstance(this);
            oldItem.setProcessInstance(null);
        }
        return this;
    }
}
```

The class will also contain getter and setter methods for your properties and helper methods to access contents of Collections. Furthermore convenience methods like `updateFromJson()`, `getAppController()` and a constructor with a **JSONObject** parameter are generated. With these, it is easily possible to create or update an entity based on a `JSONObject/JSONString` and access the `AppController`.

By using the keyword `abstract` for an Entity, the generated Java class will be an abstract class. An Entity must not have a relation to an abstract Entity by defining a variable of the abstract type. Entities can inherit from abstract Entities.

Data Modeling also includes two Entity based data sources - called `EntityDataSource` and `EntityStore`. An `EntityDataSource` can be configured to request a remote `Json` resource. The `Json` document is automatically parsed into an Entity. Furthermore the Entity is transformed back into a `Json` document to be able to send the data back to a remote `Json` resource. The Entities will be used as a list data source for **ListView** and **ListViewElement** objects. For more information, see "[EntityDataSource Properties](#)" on

[page 149](#). The EntityStore provides a single Entity and can be used to bind a single data object against UI elements. E.g. having a login mask with input fields for user name and password, you can bind both input fields against a Credentials Entity using a DataBinding object.

To add a Data Model:

1. Ensure the Mobile Project is open in the **Outline Editor**. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#).
2. In the **Model** section of the **Outline Editor**, expand the project in order to display the **Entities container** node. If the model does not have a Entities container node, add one by right-clicking the root application and selecting **New Child > Entities**.
3. Select the **Entities container** node and add a new Package by right-clicking the Entities node and selecting **New Child > Package**.
4. In the resulting dialog box, enter a new name for the Package and press **Ok**.

A new EML file has now been added to the Mobile Project and an editor opens immediately to let you edit the Data Model.

10 Building the User Interface for a Mobile Application

■ Basic Structure of the Application User Interface	74
■ Defining Panes for the Application Window	75
■ Adding Views to the Application's User Interface	79
■ Renaming a View	80
■ Adding Content to a View	81
■ Programmatically Populating a ListView	83
■ Using a Content Provider to Populate a ListView	86
■ Using a Content Adapter to Populate a ListView	90
■ Responding to user-initiated events	91
■ Defining Dialogs	93
■ Using Colors, Sizes, Styles and Themes	95
■ Using Templates to Define Custom Objects for a Mobile Project	98
■ Configuring a SideView	101

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 75](#).

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 79](#). For information about the types of views you can add, see ["Objects to Use for Views" on page 110](#).

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 81](#).

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 93](#).

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.

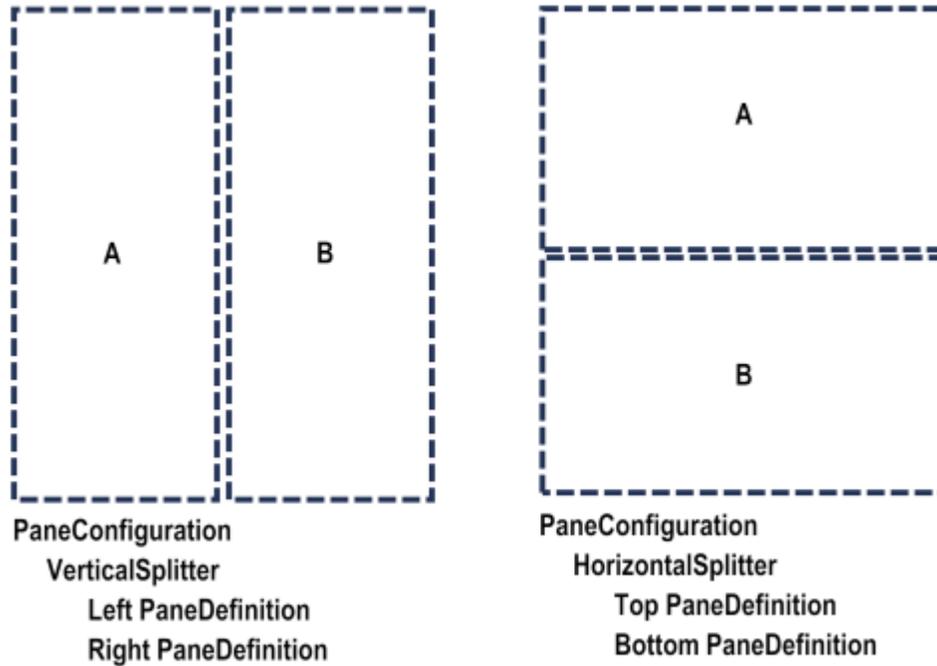
You define panes using the **PaneDefinition** object. You define a pane configuration using the **PaneConfiguration** object.

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.



PaneConfiguration
PaneDefinition

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.

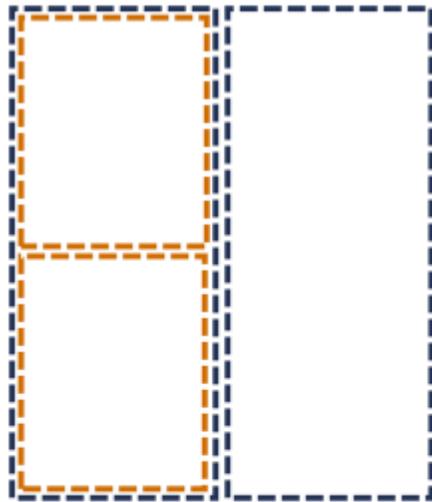


If you need a more complex arrangement of panes for an application's user interface, you can nest **VerticalSplitter** and **HorizontalSplitter** objects under parent splitter objects. For example, the following shows a layout with four panes arranged horizontally.



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 108](#).

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.

- 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 109](#).

Adding Views to the Application's User Interface

To add a view to the user interface, you add a **ListView**, **NavView**, **View**, **WebView** or **TabView** 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 17](#).
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 110](#).

4. Set the properties for the view.

For more information, see ["Setting Properties in the Outline Editor" on page 22](#).

For descriptions of the projects, see:

- ["ListView Properties" on page 111](#)
- ["NavView Properties" on page 113](#)
- ["View and SideView Properties" on page 113](#)
- ["WebView Properties" on page 114](#)
- ["TabView Properties" on page 116](#)

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 154 and ["Transition Properties"](#) on page 158.

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 174.

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 33.

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_name ControllerImpl.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 17 or ["Displaying a Window, View, or Dialog in the Outline Editor"](#) on page 18.
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 33.

Mobile Development generates a new `new_view_name ControllerImpl.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_name ControllerImpl.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_name ControllerImpl.java` Java class.

5. Update the `new_view_name ControllerImpl.java` class with any custom code that you added to the `old_view_name ControllerImpl.java` class.

- a. In the Package Explorer or Navigator view, locate the **src > package > ui > controller > impl** folder, which contains both the *new_view_name* ControllerImpl.java and *old_view_name* ControllerImpl.java Java classes.
- b. Open both Java classes and copy all custom code from the *old_view_name* ControllerImpl.java to *new_view_name* ControllerImpl.java.
- c. Save both files.
- d. Delete the *old_view_name* ControllerImpl.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 117](#)
- ["Objects to Use for Tables" on page 121](#)
- ["Objects to Use for User Interface Controls" on page 125](#)
- ["Objects to Use for Content Providers or Content Adapters" on page 143](#)
- ["Objects to Use for Event Listeners" on page 151](#)
- ["Objects to Use for Event Actions" on page 154](#)

The user interface objects that are valid in a view are based on the specific type of view, that is, whether you are adding the objects to a ListView, NavController, View, or WebView. For example, the only valid object that you can add to a NavController is a **NavController** 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 83](#) and ["Using a Content Provider to Populate a ListView" on page 86](#).

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:



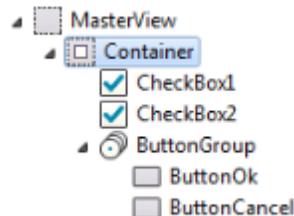
The following shows what the user interface might look like when the application executes:



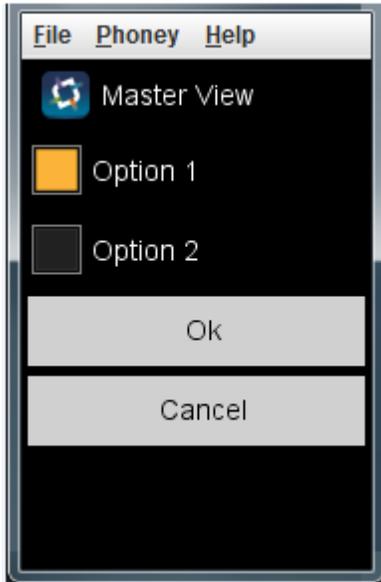
Nesting Objects in a View

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

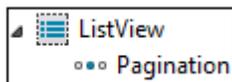
Use a **ListView** object to display a list of items. You can populate a **ListView** by adding logic to the view's controller.

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 86.

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.

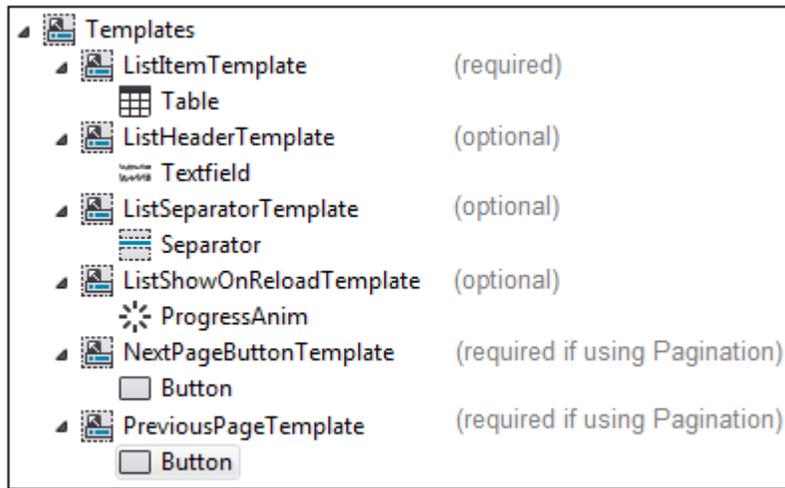
- In the **UserInterface > Views** section of the model, you need to add the following objects.



Object	Description
ListView	Required. Defines the ListView .

Object	Description
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	<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 Table or TableButton 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 ListView. For example, you might create a template for an object like a Textfield or an Image.</p> <p>If you want to provide a header for the ListView, specify the template in the ListView object's List View Header 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 ListView. For example, you might create a template for an object like a Separator, Spacer, or Image object.</p>

Template for...	Description
	If you want to provide a separator for the list of data, specify the template in the ListView object's Separator property.
Control to show when reloading data	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 ProgressAnim , Image , or Textfield object. If you want the application to display an object when refreshing the data, specify the template in the ListView object's Show On Reload property.
Control to display the next page of results	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. This template is required if you are using the Pagination object. You specify this template in the Pagination object's Next Page Template property.
Control to display the previous page of results	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. This template is required if you are using the Pagination object. You specify this template in the Pagination object's Previous Page Template property.

Logic in the Controller for the View

When you generate sources for a Mobile Project, Mobile Development generates a Java class named *view_name* ControllerImpl.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 **DynamicDataSource**, **RESTDataSource** and **EntityDataSource**. For more information about data sources, see ["Objects to Use for Content Providers or Content Adapters" on page 143](#).

Note: Instead of using a **ContentProvider**, you can programmatically populate the **ListView**. For more information, see ["Programmatically Populating a ListView" on page 83](#).

Tip: If your **ListViews** requires displaying more than one Template instance, it is recommend to use the **ContentAdapter**.

Objects to Add to the Application Model

1. Add a **ListView** object to your application (or use a template containing a **ListViewElement** object).
2. Add a data source to your application, as a child of the **Datasources** node. This can be a **DynamicDataSource**, a **RESTDataSource** or an **EntityDataSource**.
3. Add a **ContentProvider** object to the **ListView** object. The **ContentProvider** object indicates from where you obtain data and the template to be used to present the data. Set the following properties:

Property	Description
List Data Source	Required. Specifies the data source.
No Rows Template	Optional. A template that is to be shown when the data source of the ContentProvider contains no elements.
Reload On Transition To	Select this if the ContentProvider is to be reloaded each time this view is accessed.
Row Template	Required. A template used to show the contents of the ListView .

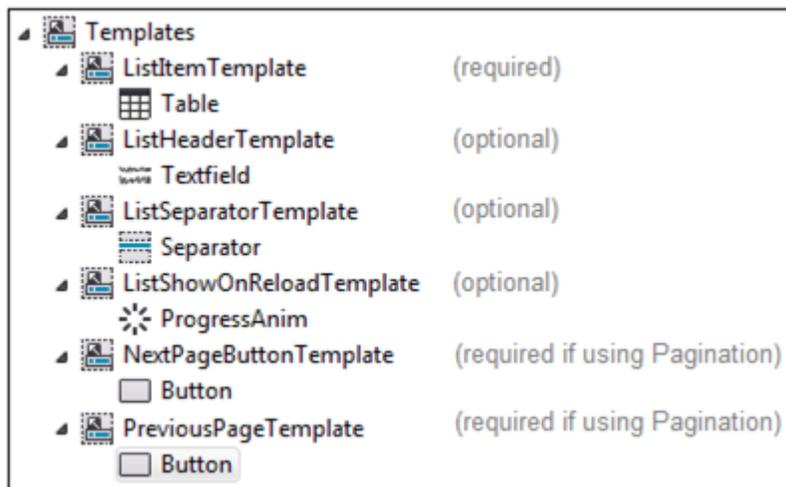
4. Add one or more **TemplateDataBinding** objects to the **ContentProvider** object. A **TemplateDataBinding** maps a control contained in the **RowTemplate** of the **ContentProvider** with a given set of data identified by the **Expression** property. Set the following properties:

Property	Description
Control	Required. Select a control that is contained in the RowTemplate of the ContentProvider .
Data Source	Select the data source to populate the data for this control.
Expression	Enter an expression value to identify the data.

5. Optional. Add a **RowSelectionListener** to react on the user interaction with the **ListView**.
6. Optional. Add a **Pagination** object if you want to display the contents of the **ListView** on different pages. Set the following properties:

Property	Description
Max Number Per Page	The maximum number of cells displayed per page.
Next Page Template	A triggerable object (such as a Button) which needs to be triggered in order to load the next page.
Previous Page Template	A triggerable object (such as a Button) which needs to be triggered in order to load the previous page.

In addition, you can add templates to customize 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 Table or TableButton object.</p> <p>You reference this template in the ContentProvider object's Template property.</p>
List header	<p>Optional. You can create a template to provide a header for the ListView. For example, you might create a template for an object like a Textfield or an Image.</p> <p>If you want to provide a header for the ListView, specify the template in the ListView object's List View Header 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 ListView. For example, you might create a template for an object like a Separator, Spacer, or Image object.</p> <p>If you want to provide a separator for the list of data, specify the template in the ListView object's Separator 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 ProgressAnim, Image, or Textfield object.</p>

Template for...	Description
	If you want the application to display an object when refreshing the data, specify the template in the ListView object's Show On Reload property.
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>

If you decided to use a **RESTDataSource**, you also need to specify a RESTful service which will be triggered by the data source. For more information, see ["Adding Services to a Mobile Project" on page 61](#).

Example: How to Use Expressions with a RESTDataSource

Consider you have a RESTful service that has a `getEmployeeInfo` method that returns information about employees. The following illustrates a sample JSON response given by the service:

```
{
  "Employees": [
    {
      "ID" : "19",
      "name" : "Leanna Jones",
      "Department": {
        "name" : "Research",
        "location" : "Boston"
      }
    },
    {
      "ID" : "30",
      "name" : "Zane Smith",
      "Department": {
        "name" : "Research",
        "location" : "Reston"
      }
    }
  ]
}
```

If you like to add a **ListView** containing all employees, you need to set the **ContentProvider** object's **Expression** property to `Employees`. The **ListView** will then contain two elements. To bind employees information to a particular template, you need to use an expression

which is now relative to `Employees`. Having a template which visualizes the employee name and department name of each employee, you need to add two **TemplateDataBinding** objects with an **Expression** property pointing to `name` and `Department.name`.

Using a Content Adapter to Populate a ListView

You can populate a **ListView** object using a **ContentProvider** object to retrieve data from a data source. The **ContentProvider** is designed to initialize one given type of **Template** and bound it against the underlying data, provided by the data source. However, this approach enables developers to quickly setup a **ListView** object, but is limited, because only one type of **Template** can be populated. The **ContentAdapter** has been introduced to close this gap and to populate multiple types of **Templates** to a **ListView** object. More information, see ["Using a Content Provider to Populate a ListView" on page 86](#).

Note: Instead of using a **ContentAdapter**, you can programmatically populate the **ListView**. For more information, see ["Programmatically Populating a ListView" on page 83](#).

To add a **ContentAdapter** to the **ListView** or **ListViewElement**:

1. Add a **ListView** object to your application or use a **Template** containing a **ListViewElement** object.
2. Add a data source to your application, as a child of the **Datasources** node. This can be a **DynamicDataSource**, a **RESTDataSource** or a **EntityDataSource**.
3. Add a **ContentAdapter** object to the **ListView** object. The **ContentAdapter** object indicates from where you obtain data. Set the following properties:

Property	Description
List Data Source	Required. Specifies the data source.
No Rows Template	Optional. A Template that is to be shown when the data source of the ContentAdapter contains no elements.
Reload On Transition To	Optional. Select this if the ContentAdapter is to be reloaded each time this view is accessed.

4. Add one or more **TemplateAssignment** to the **ContentAdapter**. The **TemplateAssignment** configures the **Template** to display. Set the following properties:

Property	Description
BindMethodName	Optional. A name of a Java method will be generated. The generated contents will initialize your Template and bind it against the data. This method is overwriteable in your ListView controller implementation.
Filter Expression	Required. A string value evaluated at runtime, if the specified Template should be initialized for a given row index.
Template	Required. A Template used to show the contents of the ListView .

5. Add one or more **TemplateDataBinding** objects to the **TemplateAssignment** object. A **TemplateDataBinding** maps a control contained in the **Template** of the **TemplateAssignment** with a given set of data identified by the **Expression** property. Set the following properties:

Property	Description
Control	Required. Select a control that is contained in the Template of the TemplateAssignment . The control must have a name.
Data Source	Required. Select the data source to populate the data for this control.
Expression	Required. Enter an expression value to identify the data.

6. Optional. Add one or more **RowSelectionListener** to the **ContentAdapter** to react on the user interaction with the **ListView**. A **RowSelectionListener** also has a property **filterExpression**. This will act like the **filterExpression**, set for a **TemplateAssignment**, and indicates, which of the **RowSelectionListeners** needs to be triggered, when the user selects on row of the **ListView**.

Responding to user-initiated events

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 if a user-initiated event occurs for a control, your application can respond by taking an appropriate action.

Types of Listeners Mobile Development supports:

- **GainFocusListener** listens for, when a user selects an object so that the user interface object gains focus.
- **LoseFocusListener** 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** listens for, when a user edits an object, for example an entry field, and generates an event after the object is edited.
- **PreEditListener** 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.
- **CompletedEditListener** listens for, when a user edits an object, for example an entry field, and generates an event when the user ends the editing. End of editing means closing the keyboard or focusing another UI element.
- **TriggerListener** listens for when, a user uses an object, for example, presses a button.
- **RowSelectionListener** listens to trigger events for each row in a `ListView` object, for example, when a user selects one row. This listener can only be added as a child of a `ContentProvider` or `ContentAdapter` object.
- **SwipeListener** listens to swipe events. You can define either a `LeftToRight` or `RightToLeft` swipe event. This listener can only be added to a normal `View` control (it is not possible to add it to a `ListView`, `NavigationView` or `WebView` control). You can only add one `SwipeListener` to a `View` control.

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 these 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.

- **ReloadContentProvider action** to let a ContentProvider object reload its data source.
- **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.
- **OpenSideView action** to open a desired SideView.
- **CloseSideView action** to close an already opened SideView.

For more information about these objects, see ["Objects to Use for Event Listeners" on page 151](#) and ["Objects to Use for Event Actions" on page 154](#).

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 17](#) or ["Displaying a Window, View, or Dialog in the Outline Editor" on page 18](#).
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 previously mentioned event listener, like `TriggerListener`.
3. Right-click the **event listener** object you added and select **New Child > event_action_object**, where `event_action_object` is one of the previously mentioned event action objects, e. g. `OpenDialog`. Please be aware, that every event listener supports any kind and number of actions. It is up to the developer to add the correct semantics to the project.
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 22](#) and ["Objects to Use for Event Actions" on page 154](#).

Note: Based on the event action you are using, you might also need to add application logic for the action.

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 33](#).

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 only one 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 17](#).
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 120](#).

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 120](#).
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 154](#), ["Responding to user-initiated events" on page 91](#), and ["OpenDialog Properties" on page 157](#).
 - For information about how to add code to open a dialog, see ["Logic to Display and Close a Dialog" on page 172](#).
 - For information about the code you can add to a dialog, see ["Logic for a Dialog" on page 171](#).

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 33](#).

Using Colors, Sizes, Styles and Themes

You can centralize the style and user interface definitions of your application by using any of the model elements offered in the **Styles** category.

Using Colors

An easy way to define commonly used colors is to add a new **Color** entry in the **Styles** category. A defined color element can then be referenced everywhere in the model instead of directly specifying a color. The advantage is that you can easily change the color that is used in several elements within the model.

To define a Color 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 17.
2. In the **Model** section of the **Outline Editor**, expand the outline so that you view the **Styles** node.
3. Expand the **Styles** node, select the **Values** node and select **New Child > Color**.
4. Select the **Color** object, and in the **Properties** section of the **Outline Editor** set the properties.
5. Give the **Color** object a unique name and the value you wish it to have.
6. Using the Color in the user interface is really easy. Just open a model element that contains a color property (Background Color, Font Color, etc.), e.g. a "[TextField](#)" on page 141. In that property you specify the name of the previously defined Color object. Using the CTRL+SPACE keys will even show you a list of all defined Color objects for you to select from.

Note: (Android only) All defined Color values will be saved in a colors.xml resource file to be used in your Android project, e.g. for native code injections. For more information, refer to <https://developer.android.com/guide/topics/resources/more-resources.html#Color>.

Using Sizes

Similar to Colors, you can also define Sizes. A size is an integer value to be used as a property for any dimensional User Interface property, e.g. innerX, height, position X.

Note: (Android only) All defined Sizes will be saved in a dimens.xml resource file to be used in your Android project, e.g. for native injections. For more information, refer to <https://developer.android.com/guide/topics/resources/more-resources.html#Dimension>.

Using Font Sizes

In addition to Colors and Sizes, you are also able to define Font Sizes. They can be used as value for any User Interface property named "Font Size".

Using Styles

A Style is a collection of common User Interface properties that you can apply to most User Interface elements. Using Styles enables you to easily reuse properties without defining them again in every control. In addition, it eases the way to change properties as they are defined in a single spot.

To define a Style 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 17.
2. In the **Model** section of the Outline Editor, expand the outline so that you view the **Styles** node. The **Styles** node is a child of the **User Interface** node.
3. Right-click the **Styles** node and select **New Child > Style**.
4. Select the **Styles** object, and in the **Properties** section of the Outline Editor set the properties.
5. You can define an inheritance of styles. A Style inherits all properties from a parent style. If both styles define a value for the same property, the value of the child take precedence.
6. To use a **Style** object, open an User Interface related model element (View, Table, TextField, etc). In that element you will find the property **Style**, where you can select a defined Style from the list. If both, the Style and User Interface element itself, define a property, then the property defined in the User Interface element takes precedence.

Using ApplicationStyle to Define Android Styles and Themes

This section describes how to customize the Look and Feel of your Android application.

Note: Applies to Android only.

The ApplicationStyle allows you to define Styles and Themes for your Android project. In Android, a Style is a collection of properties that specify the look and format for a View. For more information about Android Styles and Themes, please refer to <https://developer.android.com/guide/topics/ui/themes.html>.

Please make sure your Application model contains an ApplicationStyle.

To define an ApplicationStyle 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 17.](#)
2. In the **Model** section of the Outline Editor, expand the outline so that you view the **Styles** node.
3. Right-click the **Styles** node and select **New Child > ApplicationStyle**.

The ApplicationStyle node provides a child called Android Theme. You can add a Android Theme by right-clicking the ApplicationStyle node and selecting **New Child -> Android Theme**. After adding an Android Theme to the Application Style, a new editor opens showing all the defined styles and themes. You will find the corresponding themes.asl file in your `<project>/model/resources/` directory.

Note: A Mobile Project can only contain one Android Theme file.

A project created with Mobile Suite 9.12 already contains an Android Theme with default values set. Projects created with versions of Mobile Suite before 9.12 take these values as defaults as well:

```
values {
    <style name="ToolbarTheme" parent="ThemeOverlay.AppCompat.Dark">
    </style>
    <style name="ToolbarPopupTheme" parent="ThemeOverlay.AppCompat.Light">
    </style>
}
values-v14 {
    <style name="AppBaseTheme" parent="Theme.AppCompat.Light">
    </style>
}
```

The editor allows editing of Styles and Themes in a way familiar to Android developers: in a XML-based language. Developers can group a collection of Styles within a Values category. The values category can contain an optional resource qualifier, stated by the suffix "-qualifier", e.g. "values-v14". For more information about Android resource qualifiers, please refer to <https://developer.android.com/guide/topics/resources/providing-resources.html>. In Android, each style can contain a set of items (<https://developer.android.com/guide/topics/resources/style-resource.html>).

You can customize the theme of an Android project by adding the style named "AppBaseTheme" to your Values category. Please note that themes are limited to AppCompat, e.g. Theme.AppCompat.Light, Theme.AppCompat and Theme.AppCompat.DayNight.

After code generation, an XML file is created for each of your Values categories. Generated files are placed in the `<project>/gen/.assets/android/res` directory. Each of the Values categories will land in a directory named after the category itself, e.g. `/values-v14`. This directory contains a single styles.xml file containing your declared styles and themes. During the multi build, the entire content of the `<project>/gen/.assets/android/res` directory will be copied to the cross compiled Android project.

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
- DateEntry
- Entry
- 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 98](#).

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 99](#).

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 17](#).
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 160.

8. Save the Mobile Project and generate sources for the Mobile Project. For more information, see "[Generating Sources for a Mobile Project](#)" on page 33.

Mobile Development generates the following Java classes for the template:

- **Abstract***template_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 83](#) and ["Using a Content Provider to Populate a ListView" on page 86](#).

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 17](#) or ["Displaying a Mobile Project in the Outline Editor" on page 17](#).
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	<p>Template that you want to use.</p> <p>The list includes the templates that you have added to your project.</p>

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 33](#).

Predefined Templates

There are some predefined Templates available which can be used to quickly create commonly used Templates without the need to manually create every single element in these Templates. These predefined Templates need to be transferred via drag and drop to the Templates section in the model. Then the whole content of the predefined Template will be added to the model. Afterwards the template can be changed and used as any other template.

Template	Content
Avatar with Text and Icon	<ul style="list-style-type: none"> ■ Contains a TableButton with two dedicated TextFields for title and description and images for an Avatar and an arrow. ■ Is typically used in a ListView to show a list element which is linked to a details view.
Two-line item	<ul style="list-style-type: none"> ■ Contains a TableButton with two dedicated TextFields for title and for description.

Configuring a SideView

A **SideView** is a common pattern that displays the main navigation options for your application. It is hidden most of the time, unless the user opens it using the dedicated `OpenSideView` action or a programmatically implemented solution. On Android, it's possible to open the SideView using a left to right swipe gesture.

Mobile Development supports one SideView per window.

To configure the SideView

1. Ensure the Mobile Project is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#).
2. Follow these steps, if you have not added a SideView to your application.
 - a. In the **Model** section, locate the **Views container** node.
 - b. Add a new **SideView** to your application. Right-click the **Views** node and select **New Child > SideView** or use the palette item SideView.
3. In the **Model** section of the Outline Editor, expand the project so that you view the **MainWindow container** node.
4. Select the SideView you like to configure with the left **SideView** property of your MainWindow.
5. Mobile Development will automatically configure the application's MainWindow to be ready to open the SideView if the current state of the application has at least one open View, containing the **OpenSideView** action. Please refer to the *webMethods Mobile Designer Native User Interface Reference* more information how to programmatically open and close a **SideView**.
6. Don't forget to re-generate the Java source files. For instructions see ["Java Sources that Mobile Development Generates" on page 34](#).

Tip: Use TableButtons in your SideView to enable a transition from the SideView to another View. In the application model, add a TableButton to your

SideView and configure the Transition action. For instructions, see ["Responding to user-initiated events" on page 91](#). Before your Transition is executed, it is required to close the SideView. Therefore, please add the CloseSideView action as first child to your listener.

11 User Interface Object Reference

■ User Interface Objects	104
■ Application Node Properties	104
■ Objects to Use for Windows	106
■ Objects to Use for Panes	107
■ Objects to Use for Views	110
■ Objects to Use for the Layout of the User Interface	117
■ Objects to Use for Dialogs	119
■ Objects to Use for Tables	121
■ Objects to Use for User Interface Controls	125
■ Objects to Use for Content Providers or Content Adapters	143
■ Objects to Use for Event Listeners	151
■ Objects to Use for Event Actions	154
■ Objects to Use for Templates	158

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 104
Windows	"Objects to Use for Windows" on page 106
Panes	"Objects to Use for Panes" on page 107
Views	"Objects to Use for Views" on page 110
Layout	"Objects to Use for the Layout of the User Interface" on page 117
Dialogs	"Objects to Use for Dialogs" on page 119
Tables	"Objects to Use for Tables" on page 121
Controls	"Objects to Use for User Interface Controls" on page 125
Content Providers	"Objects to Use for Content Providers or Content Adapters" on page 143
Event Listeners	"Objects to Use for Event Listeners" on page 151
Event Actions	"Objects to Use for Event Actions" on page 154
Templates	"Objects to Use for Templates" on page 158

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	<p>Package name for your Mobile Project.</p> <p>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 183.</p>
Context Persistence Mode	<p>If set to Permanent, all values set for the context key store are made persistent. This allows users to reuse the values in the context key store even when restarting the app.</p>
Default Language	<p>Default language for the application. For more information, see "Setting the Default Language for the Project" on page 32.</p>
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 182.</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 31.</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 .</p>

Property	Description
Use Camera	Whether you want to set the Mobile Designer property <code>project.handset.uses.camera</code> to <code>true</code> . See <i>Using webMethods Mobile Designer</i> for more information on this property.
Use Location	Whether you want to set the Mobile Designer property <code>project.handset.uses.Location</code> to <code>true</code> . See <i>Using webMethods Mobile Designer</i> for more information on this property.
Use Mobile Administrator	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 29.
Use PIM	PIM stands for Personal Information Manager. Whether you want to set the Mobile Designer property <code>project.handset.uses.FCPIM</code> to <code>true</code> in order to list all contacts on your device. See <i>Using webMethods Mobile Designer</i> for more information on this property.
Use Push Notifications	Whether you want to set the Mobile Designer property <code>project.handset.push.notifications</code> to <code>true</code> . See <i>Using webMethods Mobile Designer</i> for more information on this property.
Use Sensors	Whether you want to set the Mobile Designer property <code>project.handset.uses.Sensors</code> to <code>true</code> . See <i>Using webMethods Mobile Designer</i> for more information on this property.
Use WMA	WMA stands for Wireless Messaging API. Whether you want to set the Mobile Designer property <code>project.handset.uses.WMA</code> to <code>true</code> . See <i>Using webMethods Mobile Designer</i> for more information on this property.

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	Defines the application's window.

Object	Description
	For information about setting properties for the Window object, see "Window Properties" on page 107 .

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 PaneConfiguration object that you previously defined for the Mobile Project.
Left Side View	A reference to a Side View object. For more information, see "Configuring a SideView" on page 101 . This Side View will be taken into account for the OpenSideView and CloseSideView actions. For more information about setting properties for the OpenSideView and CloseSideView , see "Objects to Use for Event Actions" on page 154 .

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 75](#).

Object	Description
HorizontalSplitter	Indicates that you want to display two panes horizontally, one on top of the other. For information about setting properties for the HorizontalSplitter object, see "HorizontalSplitter Properties" on page 108 .
PaneConfiguration	Specifies the name of a configuration of panes. Add HorizontalSplitter , VerticalSplitter , and/or PaneDefinition child objects to define how to place panes in the

Object	Description
	<p>application's Window object when using this pane configuration.</p> <p>For information about setting properties for the PaneConfiguration object, see "PaneConfiguration Properties" on page 109.</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 PaneDefinition object, see "PaneDefinition Properties" on page 109.</p>
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 109.</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

Property	Description
	<ul style="list-style-type: none"> ■ 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> <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>

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 View , ListView , NavView , or WebView 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:

Property	Description
	<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 111.</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 113.</p>
View	<p>Defines a general purpose view for your application.</p> <p>For information about setting properties for a View object, see "View and SideView Properties" on page 113.</p>

Object	Description
SideView	A SideView is a common pattern that displays the main navigation options for your application. It is hidden most of the time, unless the user opens it using the dedicated OpenSideView action or a programmatically implemented solution.
TabView	Is a common pattern that lets you group several other views and allows quick switching between them. Optionally you can define User Interface elements, which will be displayed above the tabs. For information about setting properties for a TabView object, see " TabView Properties " on page 116.
WebView	Defines a view in which you want to display Web content. For information about setting properties for a WebView object, see " WebView Properties " on page 114.

ListView 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 Foreground Color	Foreground 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.

Property	Description
	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 <code>ListView</code> . Specify a template that you previously defined. The template should customize an object like a Textfield or Image 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 <code>ListView</code> . Specify a template that you previously defined. The template should customize an object like a Separator object.

<u>Property</u>	<u>Description</u>
Show On Reload	<p>Template that defines an object to display while the application is obtaining data to refresh the list of items in the view.</p> <p>Specify a template that you previously defined. The template should customize an object like a ProgressAnim, Image or Textfield object.</p>
VScrollable	Whether you want to allow vertical scrolling in the view.

NavView Properties

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

View and SideView Properties

<u>Property</u>	<u>Description</u>
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 Foreground Color	Foreground 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.

Property	Description
	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

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.

Property	Description
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 Foreground Color	Foreground 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.
Overscrolling Enabled	Whether you want to allow overshooting the bounds of a scrolling operation (only for Android).

Property	Description
Popup Dismiss Text	For the iOS platform, the text to use on a label that closes an open keyboard or drop-down list.
Scale To Fit Enabled	Whether you want to scale the WebView to fit the parent element (only for iOS).
Url	URL to the web page to load into the view.
VScrollable	Whether you want to allow vertical scrolling in the view.

TabView Properties

A **TabView** offers the same properties as a **View** object, see "[View and SideView Properties](#)" on page 113. Furthermore it offers the following properties. To add a Tab ("[Tab properties](#)" on page 141) to a **TabView**, right click the **TabView** and select **New Child > Tab**.

Property	Description
Tab Background Color	Background color of the tabs.
Tab Font Size	Font size of text elements in the tabs
Tab Font Style	Font style of text elements in the tabs.
Tab Indicator Color	Color of the selected tab indicator (the small line at the bottom of the tab).
Tab Indicator Height	Height of the selected tab indicator (the small line at the bottom of the tab).
Tab Mode	The way the tabs are displayed; either FIXED or DYNAMIC (Android only).
Tab Selected Text Color	Color of the text elements in the selected tab.
Tab Text Color	Color of text elements in the tabs.

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 118.</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 118.</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 119.</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

Property	Description
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Visible	Whether you want the separator line to be visible.
Width	Width of the separator line.

Spacer Properties

Property	Description
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
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 120.</p>

Object	Description
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 120.</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 171.</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>
Text	<p>Text you want displayed on the button.</p>

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 DynamicTablecell object is the child of a TableRow object.</p> <p>For information about setting properties for the DynamicTablecell object, see "DynamicTablecell Properties" on page 122.</p>
DynamicTablerow	<p>Specifies a method that executes at run time to dynamically define the layout for the table and populate the table. A DynamicTablerow object is the child of a Table object.</p> <p>For information about setting properties for the DynamicTablerow object, see "DynamicTablerow Properties" on page 122.</p>
Table	<p>Displays a table.</p> <p>To specify the rows in the table, add one or more TableRow objects or a single DynamicTablerow object as children of the Table object.</p> <p>For information about setting properties for the Table object, see "Table Properties" on page 123.</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 TableButton object, see "TableButton Properties" on page 124.</p>
TableCell	<p>Adds a cell to a table row. A TableCell object is the child of a TableRow object.</p> <p>To specify user interface objects that you want to display in the table cell, add the objects as children of the TableCell object.</p>

Object	Description
	For information about setting properties for the TableCell object, see " TableCell Properties " on page 124.
TableRow	<p>Adds a single row to a table. A TableRow object is the child of a Table object.</p> <p>A child of a Table object. Use to add a single row to the table.</p> <p>To specify the contents of the table row, add one or more TableCell objects or a single DynamicTableCell object as children of the TableRow object.</p> <p>For information about setting properties for the TableRow object, see "Tablerow Properties" on page 125.</p>

DynamicTableCell Properties

Property	Description
Method Name	<p>Name of a method you code to populate the table cell.</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 172.</p>

DynamicTablerow Properties

Property	Description
Method Name	<p>Name of a method you code to populate the table.</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 172.</p>

Table Properties

Property	Description
Background Color	Background color of the table.
Border Color	Color of the table's border.
Border Thickness	Thickness of the table's border.
Create On Condition	Whether to create the table at run time.
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.
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.

Property	Description
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Visible	Whether the table is visible or hidden.
Width	Width of the table.

TableButton Properties

Property	Description
Background Color Highlight	Color of the table button when the table button has focus.
Create On Condition	Whether to create the table button at run time.
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Visible	Whether the table button is visible or hidden.

Tablecell Properties

Property	Description
Background Color	Color of the table cell.
HAlign	Horizontal alignment of the contents in the cell.
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.

Property	Description
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 129.
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.

Object	Description
	<p>For information about setting properties for the Button Group object, see "ButtonGroup Properties" on page 130.</p>
Checkbox	<p>Displays a check box.</p> <p>For information about setting properties for the Checkbox object, see "CheckBox Properties" on page 130.</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 Container object.</p> <p>You can set the Container 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 Container object, see "Container Properties" on page 131.</p>
DateEntry	<p>Displays a date or time selector control.</p> <p>For information about setting properties for the DateEntry object, see "DateEntry Properties" on page 132.</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 StringDropDownListEntryItem objects or a single DynamicDropDownListEntryItem object as children of the DropDownListEntry object.</p> <p>For information about setting properties for the DropDownListEntry object, see "DropDownListEntry Properties" on page 133.</p>
DynamicDisplayObject	<p>Name of a method you code to display a user interface object. For information about setting</p>

Object	Description
	properties for the DynamicDisplayObject object, see "DynamicDisplayObject Properties" on page 133 .
DynamicDisplayObjectArray	Name of a method you code to display an array of user interface objects. For information about setting properties for the DynamicDisplayObject object, see "DynamicDisplayObject Properties" on page 133 . For information about setting properties for the DynamicDisplayObjectArray object, see "DynamicDisplayObjectArray Properties" on page 134 .
DynamicDropDownListEntryItem	Specifies a method that executes at run time to provide the list of entries to display in the drop-down list. A DynamicDropDownListEntryItem object is the child of a DropDownListEntry object. For information about setting properties for the DynamicDropDownListEntryItem object, see "DynamicDropdownlistEntryItems Properties" on page 135 .
Entry	Displays a text entry box. You can set the Entry object's Input Type property to: <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). For information about setting properties for the Entry object, see "Entry Properties" on page 135 .
Image	Displays an image. To specify the image that you want to display, set the Image object's Image property. You can use an Image object as a button if you add a TriggerListener object as a child object. For information about setting properties for the Image object, see "Image Properties" on page 136 .

Object	Description
NavButton	<p>Displays a button that an application uses for navigation.</p> <p>For information about setting properties for the NavButton object, see "NavButton Properties" on page 137.</p>
Pagination	<p>Adds objects that a user selects to display the next or previous page of list items in a ListView.</p> <p>For information about setting properties for the Pagination object, see "Pagination Properties" on page 138.</p>
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 138.</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 139.</p>
SearchEntry	<p>Displays a search entry field.</p> <p>For information about setting properties for the SearchEntry object, see "SearchEntry Properties" on page 140.</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 141.</p>
Textfield	<p>Displays plain text in a label or for a block of text.</p>

Object	Description
	For information about setting properties for the Textfield object, see " Textfield Properties " on page 141.
WebViewElement	Use to display rich Web content from a local file or by specifying the URL of the content to display. For information about setting properties for the WebViewElement object, see " WebViewElement Properties " on page 142.

Button Properties

Property	Description
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.
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Text	Text to display on the button.

Property	Description
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.
Position Y	Distance from the check box's top edge to its parent object's inner Y position.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.

Property	Description
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

Property	Description
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.

Property	Description
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
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.
Fallback Text	The text to be displayed if no date is set.
Font Color	Color of the text.
HAlign	Horizontal alignment of the contents in the entry.
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Visible	Whether the date or time selector control is visible or hidden.

Property	Description
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.
HAlign	Horizontal alignment of the contents in the entry.
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Visible	Whether the drop-down list is visible or hidden.
Width	Width of the drop-down list.

DynamicDisplayObject Properties

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

Property	Description
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 172.
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Visible	Whether the user interface object is visible or hidden.

DynamicDisplayObjectArray Properties

Property	Description
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 172.
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Visible	Whether the array of user interface objects is visible or hidden.

DynamicDropdownlistEntryItems Properties

Property	Description
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 172.

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.

Property	Description
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
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.
Image	Image to display. Note: The image must be one of your application resources. For more information about resources, see .
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.

Property	Description
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.

NavButton Properties

Property	Description
Create On Condition	Whether to create the button at run time.
Icon	Icon image to display on the button. Note: The icon must be one of your application resources. For more information about resources, see .
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.
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.

Property	Description
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.

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.
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.
Visible	Whether the object is visible or hidden.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.

RadioButton Properties

Property	Description
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.

Property	Description
Width	Width of the radio button.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.

SearchEntry Properties

Property	Description
Context Key	<p>Name of a context key you want to reference using syntax such as <code>#{CONTEXT_key}</code>.</p> <p>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.</p>
Create On Condition	Whether to create the search entry box at run time.
Hint	<p>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.</p> <p>Leave this property blank if you do not want to provide hint text.</p>
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.

StringDropDownlistEntry Properties

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

Tab properties

<u>Property</u>	<u>Description</u>
Background Color	Background color of the tab.
Image	Icon that should be displayed in the tab.
Name	Name of the tab.
Text	Text that should be displayed in the tab.
View	View that should be displayed, when you click the tab. Only View and WebView can be selected.

Textfield Properties

<u>Property</u>	<u>Description</u>
Clip Type	Define the truncation mode that should be applied if not the whole text value can be displayed. Must be one of CLIP_TYPE_CLIP or CLIP_TYPE_ELLIPSIS.
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.
Font Style	How the text should be formatted, for example, bold, italic, or underlined.
HAlign	Horizontal alignment of the text.

Property	Description
Max Lines	<p>Maximum number of lines of text to display in the text field when line-wrapped.</p> <p>The value you specify for Max Lines must be greater than or equal to the value you specify for Min Lines.</p>
Min Lines	<p>Minimum number of lines of text to display in the text field when line-wrapped.</p> <p>The value you specify for Min Lines must be less than or equal to the value you specify for Max Lines.</p>
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.
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
Text	Text to display.
Visible	Whether the text field is visible or hidden.
Width	Width of the text field.

WebViewElement Properties

Property	Description
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.

Property	Description
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.
Overscrolling Enabled	Whether you want to allow overshooting the bounds of a scrolling operation (only for Android).
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.
Scale To Fit Enabled	Whether you want to scale the WebView to fit the parent element (only for iOS).
Tag	Used to determine the context of the element at runtime, e.g. in a Delegate to find out which element was pressed.
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 or Content Adapters

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
DynamicDataSource	Holds a list of elements to be bound to the user interface elements. You need to implement <code>IListDatasource</code> . The

Object	Description
	<p>DynamicDataSource object can be used if the underlying data set is not in JSON format or if the underlying data consists of a custom data structure. A DynamicDataSource object is the child of the application's Datasources node.</p> <p>For information about setting properties for the DynamicDataSource object, see "DynamicDataSource Properties" on page 147.</p>
RESTDataSource	<p>Holds a list of elements to be bound to the user interface elements. The underlying data set must conform to the JSON format, and it is retrieved using a RESTful service specified with the Rest Method property. All expressions pointing to a RESTDataSource object are relative to the underlying JSON objects. A RESTDataSource object is the child of the application's Datasources node.</p> <p>For information about setting properties for the RESTDataSource object, see "RESTDataSource Properties" on page 148.</p> <p>See also "Objects to Use for RESTful Services" on page 162.</p>
EntityDataSource	<p>Holds a list of elements to be bound to the user interface elements. The data objects from this data source must be instances of generated Entity models.</p> <p>For more information about setting properties to the EntityDataSource object, see "EntityDataSource Properties" on page 149.</p>
RESTDataProvider	<p>A RESTDataProvider can be added as child for an EntityDataSource. It acts like a service for your EntityDataSource. The given REST Method will be executed when the data source is reloaded.</p> <p>For more information about setting properties to the RESTDataProvider, see "RESTDataProvider Properties" on page 149.</p>
FileDataProvider	<p>A FileDataProvider can be added as child for an EntityDataSource. It acts like a service for your EntityDataSource. You can only use it for local/offline data, which is stored on your device for example. The file must be located in the projects resources/www folder (subdirectories are not allowed). You need a DataExpression to define the format the data is stored</p>

Object	Description
	<p>in the file and the Entity to map the data to. Currently only <code>JsonDataExpression</code> for Json formatted data files is supported.</p>
Json Data Expression	<p>A Json Data Expression represents a path in your Json reponse document. This path is identified by the expression property. The underlying Json document will be transformed into the specified Entity.</p> <p>Fore more information about setting properties to <code>JsonDataExpression</code>, see "JsonDataExpression Properties" on page 149.</p>
ContentProvider	<p>Specifies the content you want listed in a ListView object.</p> <p>For information about setting properties for the ContentProvider object, see "ContentProvider Properties" on page 147.</p>
ContentAdapter	<p>Specifies the content you want listed in a <code>ListView</code> object.</p> <p>Fore more information about setting properties to the <code>ContentAdapter</code> object, see "ContentAdapter Properties" on page 150.</p>
TemplateAssignment	<p>The TemplateAssignment configures the Template to display by the ListView object. Each Content Adapter must provide at least one TemplateAssignment object.</p> <p>Fore more information about setting properties to the <code>TemplateAssignment</code> object, see "TemplateAssignment Properties" on page 150.</p>
DataTransformer	<p>Allows you to transform a given input to a specific format required by a particular user interface element. For example, when you get the name of an icon, the data transformer will get the file name as its value and return an icon which is then used by an ImageElement. A DataTransformer object is the child of a DataBinding or TemplateDataBinding object.</p> <p>For information about setting properties for the DataTransformer object, see "DataTransformer Properties" on page 151.</p>

Object	Description
Custom Data Comparator	<p>Allows developers to define an order for the elements a ListView displays. Can be added as a child to a DynamicDataSource, RESTDataSource or EntityDataSource.</p> <p>For more information about setting properties to the CustomDataComparator object, see "CustomDataComparator Properties" on page 151.</p>
TemplateDataBinding	<p>Defines how to bind data from the data source to an object in the user interface. A TemplateDataBinding object is the child of a ContentProvider or TemplateAssignment object.</p> <p>For more information about setting properties for the TemplateDataBinding object, see "TemplateDataBinding Properties" on page 146.</p>
DataBinding	<p>Defines how to bind data from the data source to an object in the user interface. A DataBinding object is the child of a Textfield, Image, Checkbox, RadioButton, Entry, SearchEntry or DateEntry object.</p> <p>For information about setting properties for the DataBinding object, see "DataBinding Properties" on page 147.</p>
Row Selection Listener	<p>Listens to trigger events for each row in a ListView object, for example, when a user selects one row. Multiple Row Selection Listener objects can be added to a ContentAdapter. A ContentProvider supports only one Row Selection Listener.</p> <p>For more information about setting properties to a Row Selection Listener object, see "RowSelectionListener Properties" on page 153.</p>

TemplateDataBinding Properties

Property	Description
Control	Name of the user interface object to which to bind data from the data source.
Data Source	Name of the DynamicDataSource , RESTDataSource or EntityDataSource object to use to obtain the data to bind to the control.

Property	Description
Expression	Expression that identifies the data to bind to the control specified by the Control property.

DataBinding Properties

Property	Description
Data Source	Name of the DynamicDataSource , RESTDataSource or EntityDataSource object to use to obtain the data to bind to the control.
Expression	Expression that identifies the data to bind to the user interface object.

DynamicDataSource Properties

Property	Description
Method Name	<p>Name of a method you code to populate a ListView. 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 172.</p> <p>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.</p>
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.

Property	Description
No Rows Template	Name of a Template object that you previously defined. It will be shown if the data source contains no elements after having been reloaded.
Reload On Transition To	A boolean value indicating whether the data source is to be reloaded each time when the view containing this ContentProvider object is accessed.
Row 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 98 and "Using a Content Provider to Populate a ListView" on page 86.</p>

RESTDataSource Properties

Property	Description
Expression	Identifies a JSON array which will be used as the source to get the elements managed by the data source. For more information about JSON expressions, see <code>com.softwareag.mobile.runtime.toolkit.rest.JPath</code> .
Filter	A JSON expression value to filter the underlying JSON data. For more information about JSON expressions, see <code>com.softwareag.mobile.runtime.toolkit.rest.JPath</code> .
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>

EntityDataSource Properties

<u>Property</u>	<u>Description</u>
Name	Name you assign the data source in the Outline Editor for your own reference purpose.

RESTDataProvider Properties

<u>Property</u>	<u>Description</u>
Name	Name you assign the data source in the Outline Editor for your own reference purpose.
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>

JsonDataExpression Properties

<u>Property</u>	<u>Description</u>
Expression	<p>Identifies a JSON array which will be used as the source to get the elements managed by the data source.</p> <p>For more information about JSON expressions, see <code>com.softwareag.mobile.runtime.toolkit.rest.JPath</code>.</p>
Filter Expression	<p>A JSON expression value to filter the underlying JSON data.</p> <p>For more information about JSON expressions, see <code>com.softwareag.mobile.runtime.toolkit.rest.JPath</code>.</p>
Name	Name you assign the data source in the Outline Editor for your own reference purpose.

Property	Description
Entity	A reference to an Entity. Each JSON Object will be transformed into this Entity. Therefore, the Entity must contain at least one @JsonProperty annotation.

ContentAdapter Properties

Property	Description
List Data Source	A reference to a DynamicDataSource, RESTDataSource or EntityDataSource which contains the elements to populate to the ListView.
Reload On Transition To	A boolean value indicating whether the data source is to be reloaded each time when the view containing this ContentAdapter object is accessed.
No Rows Template	Name of a Template object that you previously defined. It will be shown if the data source contains no elements after having been reloaded.

TemplateAssignment Properties

Property	Description
Bind Method Name	A name of a Java method will be generated. The generated contents will initialize your Template and bind it against the data.
Filter Expression	A String value evaluated at runtime, if the specified Template should be initialized for a given row index.
Template	A reference to a Template to populate to the ListView object.

DataTransformer Properties

Property	Description
Class Name	The value of Class Name can be the simple name of a class (e.g. <i>MyTransformer</i>) or a fully qualified class name (e.g. <i>com.company.transformers.MyTransformer</i>). If a simple name is specified, the corresponding Java class will be generated into <code>src/bundleId/data/transformer/classname.java</code> . If a fully qualified class name is provided, the transformer class will not be generated. It has to be defined in the projects class path then. Press CTRL +SPACE to view the list of Built-In transformers provided by Mobile Development. All transformer implementations need to implement <i>IDataTransformer</i> .
Constructor Arguments	A bunch of string values can be specified here to be passed to the transformer as constructor arguments. Only data type String is supported. The transformer needs to offer a constructor, to handle the specified number of String arguments. E.g. a pattern for date conversion can be specified here.

CustomDataComparator Properties

Property	Description
Class Name	A name of the class which contains the comparison logic. This class will be generated into the <code>src/<yourBundleId>/comparator</code> package and must implement <code>com.softwareag.mobile.runtime.toolkit.delegates.IComparator</code> .

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.

For more information on the supported listeners, see "[Responding to user-initiated events](#)" on page 91.

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>
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 LoseFocusListener object as a child of the Entry object.</p> <p>The LoseFocusListener object does not have any properties.</p>
PostEditListener	<p>Generates a user-initiated event after a user edits an object.</p> <div data-bbox="574 1083 1370 1182" style="background-color: #f0f0f0; padding: 5px;"><p>Note: In the case of keyboard entry, a new event is generated for each character typed or deleted.</p></div> <p>For example, if you want to generate a user-initiated event after a user types an entry in a SearchEntry object, add a PostEditListener object as a child of the SearchEntry object.</p> <p>The PostEditListener 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 SearchEntry object, add a PreEditListener object as a child of the SearchEntry object.</p> <p>The PreEditListener 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 Button object, add a TriggerListener object as a child of the Button object.</p>

Object	Description
	<p>Note: If you add a TriggerListener as the child of an Image object, the image acts as a button.</p> <p>The TriggerListener object does not have any properties.</p>
SwipeListener	<p>Generates a user-initiated event when a user swipes to the left or right.</p> <p>A SwipeListener is the child of a View object.</p> <p>For information about setting properties for the SwipeListener object, see "SwipeListener Properties" on page 153.</p>
CompletedEditListener	<p>Listens for, when a user edits an object, for example an entry field, and generates an event when the user ends the editing. End of editing means closing the keyboard or focusing another UI element.</p> <p>The CompletedEditListener object does not have any properties.</p>
RowSelectionListener	<p>Generates a user-initiated event when a user selects a row.</p> <p>A RowSelectionListener is the child of a ContentProvider object.</p>

SwipeListener Properties

Property	Description
Direction	Either a LeftToRight or RightToLeft swipe event.

RowSelectionListener Properties

Property	Description
Filter Expression	The Filter Expression property acts like the Filter Expression set for the TemplateAssignment object. It identifies at runtime, if the added actions should be executed or not by evaluating the Filter Expression. This

Property	Description
	property becomes optional for RowSelectionListener objects added to a ContentProvider.

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.

For more information on how the application responds to a user-initiated event, see ["Responding to user-initiated events" on page 91](#).

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 156.</p> <div style="background-color: #f0f0f0; padding: 10px; border: 1px solid #ccc;"> <p>Note: This action is only supported if the application uses the Mobile DevelopmentTransitionStackController. If the application override the TransitionStackController, this event action will not work properly. For more information about the TransitionStackController, see "About the TransitionStackController" on page 169.</p> </div>
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 156.</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 157.</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 157.</p>

Object	Description
ReloadContentProvider	<p>The application reloads the data source.</p> <p>The ReloadContentProvider object does not have any properties.</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 157.</p>
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 158.</p>
OpenSideView	<p>A SideView is a common pattern that displays the main navigation options for your application. It is hidden most of the time, unless the user opens it using the dedicated OpenSideView action or a programmatically implemented solution. A OpenSideView action opens a desired SideView.</p> <p>For information about setting properties for the OpenSideView object, see "Responding to user-initiated events" on page 91.</p> <p>The OpenSideView object does not have any properties yet.</p>
CloseSideView	<p>A SideView is a common pattern that displays the main navigation options for your application. It is hidden most of the time, unless the user opens it using the dedicated OpenSideView action or a programmatically implemented solution. A CloseSideView action closes an already opened SideView.</p> <p>For information about setting properties for the CloseSideView object, see "Responding to user-initiated events" on page 91.</p>

Object	Description
	The CloseSideView object does not have any properties yet.

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 <code>TransitionStackController</code> opens the first pushed view. For more information about the <code>TransitionStackController</code>, see "About the TransitionStackController" on page 169.</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 172.</p>
On Condition	Whether to perform the change pane configuration action at run time.
Pane Configuration	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.

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 172.</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 AlertDialog 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>

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.

If you nest templates inside templates, only identify controls in the top-level template.

Transition Properties

Property	Description
Method Name	<p>Name of a method that performs the transition to the new view. Specifying the Method Name property is optional.</p> <p>When you specify Method Name, 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 172.</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 ListView , NavigationView , View , or WebView 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 98](#).

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

Object	Description
ListViewElement	Defines a custom object that works like a ListView . When you create templates for ListViewElement 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 ListView object.

Object	Description
	For information about setting properties for the ListViewElement object, see " ListViewElement Properties " on page 159.
Template	Defines a custom user interface object. For information about setting properties for the Template object, see " Template Properties " on page 160.
TemplateReference	Adds the custom user interface object to the user interface. For information about setting properties for the TemplateReference object, see " TemplateReference Properties " on page 160.

ListViewElement Properties

Property	Description
Adjust Height	Adjusts the height of the object to the total size of its child controls.
Height	Height you want to use for the object.
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 Textfield or Image object.
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 Separator object.
Width	Width you want to use for the object.

Template Properties

<u>Property</u>	<u>Description</u>
Class Name	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 98 .

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.

12 Services Object Reference

■ Objects to Use for RESTful Services	162
■ Objects to Use for Mobile Support - Offline Synchronization	165

Objects to Use for RESTful Services

The following table provides a description of the objects you use for RESTful 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 162 .
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 163 .
Method	Identifies the method to execute. For information about setting properties for the Method object, see "Method Properties" on page 163 .
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 164 .
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 164 .
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 165 .

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, https://mycompany.com .

Property	Description
	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"> ■ DELETE to delete the resource identified by the request URI on the server.

Property	Description
	<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. ■ PUT to send the request with the data sent as a separate data block.

Request Properties

Property	Description
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. ■ multipart/form-data if the REST service requires the parameters to be sent in multipart format.

Parameter Properties

Property	Description
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>

Property	Description
	<p>Note: Do not use <code>postData</code> as a value. The value <code>postData</code> is reserved.</p>
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 URL.
Url Encode	<p>Specifies that the value of the parameter must be url encoded.</p>

Response Properties

Property	Description
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.

Objects to Use for Mobile Support - Offline Synchronization

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

Object	Description
SyncComponent	<p>Provides an implementation to synchronize data with a remote host using the Mobile Support library. Data will be stored on the device using a SQLite database. Synchronization is a periodic process which automatically relaunches itself after the amount of milliseconds specified</p>

Object	Description
	<p>in the Update Interval property. A SyncComponent object can be used as a REST method for a REST data source.</p> <p>For information about setting properties for the SyncComponent object, see "SyncComponent Properties" on page 166.</p>

SyncComponent Properties

Property	Description
App Name	Name of the mobile application. For more information, see <i>Developing Data Synchronization Solutions with webMethods Mobile Support</i> .
App Version	Version of the mobile application. For more information, see <i>Developing Data Synchronization Solutions with webMethods Mobile Support</i> .
Hostname	Host name of the Enterprise Gateway Server.
Msc Alias	Alias used to identify the mobile sync component. For more information, see <i>Developing Data Synchronization Solutions with webMethods Mobile Support</i> .
Port	Port on the above host.
Update Interval	A value in milliseconds (default 60000 = 1 minute). After this time period, the SyncComponent object automatically starts a new synchronization process.

13

Creating Application Logic

■ About Adding Application Logic	168
■ About the TransitionStackController	169
■ Logic for a View	170
■ Logic for a Dialog	171
■ Logic to Display and Close a Dialog	172
■ Logic for a Method Name Property	172
■ Logic to Programmatically Set a Property Value at Run Time	173
■ Logic to Transition to Another View	174
■ Common Methods to Override in the Generated Code for the Application	175
■ Common Methods to Override in the Generated Code for a View	177
■ Common Methods to Override in the Generated Code for a Template	179
■ Using the Sync Component Object	179

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 170 .
Dialog	Business logic for a dialog. For more information, see "Logic to Display and Close a Dialog" on page 172 .
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 DynamicDropDownListEntryItem object to identify the method the application executes at run time to populate the parent DropDownListEntry object. For information, see "Logic for a Method Name Property" on page 172 .

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 173.</p>
Event handling	<p>Logic to respond to a user-initiated event.</p> <p>For more information, see "Responding to user-initiated events" on page 91.</p>
View transitions	<p>Logic to transition to another view.</p> <p>For more information, see "Logic to Transition to Another View" on page 174</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 98.</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 **NavView** 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 34](#).

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_name ControllerImpl.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_name ControllerImpl.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_name ControllerImpl.java` file.

When initially generated, the `view_name ControllerImpl.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 177](#).

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 ["Responding to user-initiated events" on page 91](#) and ["OpenDialog Properties" on page 157](#).

- 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_name ControllerImpl.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_name Controller.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 172](#).

■ 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 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 ["Responding to user-initiated events" on page 91](#).

- **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 DevelopmentTransitionStackController provides logic for transitioning to previous views. For more information, see ["About the TransitionStackController" on page 169](#).

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 DevelopmentTransitionStackController, 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 MySecondViewController());
```

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 the Generated Code for the Application

When you generate sources for a Mobile Project, Mobile Development generates a Java class named `application_name` `AppControllerImpl.java`, where `application_name` is the name you assigned the application. The `application_name` `AppControllerImpl.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.

Mobile Development provides a set of methods for the controller implementation that represents the application's life-cycle. These methods are starting points for defining the custom business logic.

Method	Description
<code>onCreateWindow()</code>	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>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>
<code>onMessage(PushNotification notification)</code>	This method is called after a push notification was received by the device while the app is running. If the app is not running, the <code>onStartAppFromPushNotification(PushNotification)</code> method is called. The passed PushNotification object contains a string (representing the message) and parameters which were sent with the push notification.

Method	Description
<code>onPushNotificationError(String)</code>	This method will be called if a push notification results in an error.
<code>createSideViewController()</code>	This method is called once the user calls <code>getSideViewController()</code> on the AbstractApplicationController . It will create a new SideViewController . Overwrite this method to publish your own implementation.
<code>onDeviceRegistered(String)</code>	<p>This method is called after the device is registered to receive push notifications.</p> <p>Once the device is registered on the platforms push notification service (GCM, APNS or WPNS), a registration id is created to identify this application. This id will be send to the application, which results in calling this method. Overwrite this method to publish the registration id to Mobile Administrator.</p>
<code>updatePaneDimensions()</code>	This method is called by <code>onOrientationChange</code> after the user has rotated the device. The default implementation tries to re-size all panes to the new dimensions. You can overwrite this method if you want to have pane dimensions other than those provided by the default implementation.
<code>onStartAppFromPushNotification</code> (PushNotification notification)	This method will be called if the app receives a push notification, but is not running. As a counterpart, <code>onMessage(PushNotification)</code> will be called if the app receives a push notification and is running or in the background.
<code>onStartAppFromURL(String scheme, String url)</code>	Will be called when the application is started from an URI scheme it is registered for. The registration needs to be defined in <code>targets/_default.xml</code> . For detailed information, refer to <i>Defining Data Sharing for an Application</i> in the Mobile Designer documentation available in Software AG Designer via Help > Help Contents > Software AG Designer Guides > webMethods Mobile Designer Help .
<code>onStartAppFromFile(String mimeType, String absolutePath)</code>	Will be called when the application is started by passing a file into the application. The mime-type is provided as well as the absolute path to read the file from.

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_name* ControllerImpl.java, where *view_name* is the name you assigned to the view. The *view_name* ControllerImpl.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.

Mobile Development provides a set of methods for the controller implementation that represents the view's life-cycle. These methods are starting points for defining the custom business logic.

Method	Description
onTransitionTo()	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.
onTransitionFrom()	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.
onAlertDialogButtonPressed()	This method is executed when a user presses a button, (an AlertDialogButton object) in an alert dialog (an AlertDialog object). The method is passed the identifier that you specify in the AlertDialogButton object's Id 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.
onBackButtonEvent()	This method is executed when a user presses the view's Back button. By default, applications you create using Mobile Development use the provided TransitionStackController , 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 TransitionStackController , see " About the TransitionStackController " on page 169.
hidesBackButton()	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

Method	Description
	<p>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.</p> <p>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.</p> <p>For more information about the <code>TransitionStackController</code>, see "About the TransitionStackController" on page 169.</p>
<code>getBackButtonTitle()</code>	<p>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".</p>
<code>nUIEventCallback()</code>	<p>This method 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.</p> <p>You have to call the super method so that the default event implementations are not disabled:</p> <pre data-bbox="592 1312 1373 1396">public void nUIEventCallback() { super.nUIEventEventCall(); }</pre>
<code>initBindings()</code>	<p>This method is about to be executed when the controller tries to update the modeled bindings. This method is executed after all parts of the view have been created. All user interface elements for the view will be created at execution time.</p>
<code>onUISynced()</code>	<p>This method is called after <code>onUIViewDisplay#onUISynced()</code> is called by the platforms UI layer. This is typically the case after the elements within the view are arranged or changes on the elements are applied. When this method is called, it is safe to get all absolute dimensions for the view or the containing UI elements.</p>

Common Methods to Override in the Generated Code for a Template

When you generate sources for a Mobile Project, Mobile Development generates a Java class named *template_name*.java, where *template_name* is the class name you assigned to a template. The *template_name*.java file resides in the *src.package_name.ui.template* package, where *package_name* is the package name you specified for your Mobile Project.

Mobile Development provides a set of methods for the template. These methods are starting points for defining the custom business logic.

Method	Description
<code>init()</code>	This method is executed after the template has been initialized. It creates the contents of the template. If you want to override this method, you have to add a call to the super method.
<code>onTemplateCreated()</code>	This method is executed by <code>init()</code> after all contents have been created. You can add custom business logic to this method.
<code>onAlertDialogButtonPressed()</code>	This method is executed when a user presses a button, (an AlertDialogButton object) in an alert dialog (an AlertDialog) object. The method is passed the identifier that you specify in the AlertDialogButton object's Id 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>initBindings()</code>	This method is about to be executed when the controller tries to update the modeled bindings. This method is executed after all parts of the template have been created. All user interface elements for the template will be created at execution time.

Using the Sync Component Object

The **webMethods** Mobile Suite provides capabilities to enable offline data synchronization. This means that a specific set of data can be transmitted from a device to a dedicated server, and vice versa, and conflicts are automatically resolved. With the Mobile Suite, the Mobile Support library does all the communication with the host and provides information when a data object is added, changed or deleted. For more

information about Mobile Support, see *Developing Data Synchronization Solutions with webMethods Mobile Support*, which is part of the Integration Server documentation.

Mobile Support, however, is only responsible for synchronizing the data with a server. In the application, the data still needs to be persisted to enable proper offline support. To facilitate the development with Mobile Support, Mobile Development builds a library on top to achieve the offline support and provides a **SyncComponent** object to configure and implement the connection. See ["Adding the SyncComponent Object" on page 65](#) and ["Services Object Reference" on page 161](#).

When the **SyncComponent** object is added to a project, Mobile Development generates a number of API classes. After adding the **SyncComponent** object, it is therefore required to regenerate the API classes. The **SyncComponent** object itself behaves like a RESTful service and can be added to a **RESTDataSource** object as a REST method. Once the **SyncComponent** object has been added to the project, the singleton instance can be accessed using the `AbstractApplicationController#getSyncClient` method. You can thus start or stop the synchronization or even attach a `com.softwareag.mobile.runtime.toolkit.delegates.ISyncClientListener` to get information about status changes, for example, when the synchronization is about to start or if an exception has occurred.

When you use the **SyncComponent** object as a REST method for a **RESTDataSource**, the synchronization will start automatically. Otherwise, you have to start it programmatically by calling `#start()` on the common instance. Before starting, make sure that all credentials have been set. As known from RESTful services, the **SyncComponent** object is prepared to use the credentials placed on the common **Session** object. For more information about providing credentials, see the code snippet in ["Configuring a Session Object with Credentials" on page 188](#).

To synchronize data, the **SyncComponent** object establishes a synchronization process:

1. First, it tries to load local data from the underlying SQLite database.
2. If no data is stored, it downloads all data from the dedicated server. If data is stored, it asks the server whether there are any updates for this data set.
3. Incoming data is stored in the database. It performs delete, update and add operations to keep the local data up-to-date.

When the synchronization process is complete, it is restarted after a specific update interval, as specified in the application model or as set programmatically. This ensures an ongoing synchronization during the entire run time of the application. Synchronization only stops if an exception occurs with the communication, for example, if there is no connection to the network. For such a case, you have to implement an `ISyncClientListener` and restart the **SyncComponent** object.

14 Managing a Project

■ Renaming a Mobile Project	182
■ Renaming the Application	182
■ Changing the Package Name	183

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 182](#).

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 182](#).

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 17](#).
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 33](#).

Mobile Development generates a new *new_app_name* `AppControllerImpl.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_name* `AppControllerImpl.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_name* `AppControllerImpl.java` Java class.

7. Update the *new_app_name* `AppControllerImpl.java` Java class with any custom code that you added to the *old_app_name* `AppControllerImpl.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_name* `AppControllerImpl.java` and *old_app_name* `AppControllerImpl.java` Java classes.
 - b. Open both Java classes and copy all custom code from the *old_app_name* `AppControllerImpl.java` to *new_app_name* `AppControllerImpl.java`.
 - c. Save both files.
 - d. Delete the *old_app_name* `AppControllerImpl.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 17](#).
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.

15 Code Snippets

■ How to use the DateFormatter and DateTransformer	186
■ Detecting the Current Platform	186
■ Using the Context Key Store to Store and Retrieve Application-wide Settings	187
■ Encoding and Decoding Images with Base64	188
■ Configuring a Session Object with Credentials	188
■ Creating Your Own Operation	189
■ Chaining Multiple Operations	189
■ Using Multipart Requests	190
■ Doing Error Handling for Operations	191
■ Getting the Current GPS Position and Translating it into a Human-readable Location	192
■ Downloading an Image from a Remote Host	193
■ Using a Java Class to Communicate with the Natively Injected Code	194
■ Configure your application to receive Push Notifications	195
■ Configure a Content Adapter to use multiple Templates	197

How to use the DateFormatter and DateTransformer

When working with data driven applications, it is a common use case to convert a **Date** object into a **String** or a **String** into a **Date** based on a given pattern. The recommended way to handle date format conversions is using the class `com.softwareag.mobile.md.text.DateFormat`. If required, **DateFormat** also allows to use the **Timezone** offset and the **Locale** identifier for the conversions. The **Locale** is important, if you need to convert date strings, which contain the name of a month or a day.

Simple example for using `DateFormat`:

```
public String formatDate(java.util.Date date) {
    DateFormat dateFormat = new DateFormat("MM/dd/yyyy hh:mm:ss a");
    String formattedDate = dateFormat.format(date);
    return formattedDate;
}
public java.util.Date parseDateString(String dateString) throws ParseException {
    DateFormat dateFormat = new DateFormat("dd.MM.yyyy");
    java.util.Date date = dateFormat.parse(dateString);
    return date;
}
```

There is also a new Transformer implementation as wrapper for **DateFormat**, that allows date conversions in code as well as in the application model file (for example in **ContentAdapter** or **Template** bindings). The transformer implementation `com.softwareag.mobile.runtime.toolkit.transformer.DateTransformer` implements

`com.softwareag.mobile.runtime.toolkit.delegates.IDataTransformer` and can therefore be used in all transformer definitions in the application model. It can also be used programmatically, as you can see in the snippets below:

```
public String formatDate(java.util.Calendar calendar) {
    DateTransformer dateTransformer = new DateTransformer("dd.MM.yy");
    String formattedDate = (String) dateTransformer.transform(calendar.getTime(),
        IDataTransformer.TYPE_STRING);
    return formattedDate;
}
public java.util.Date parseDateString(String dateString) {
    DateTransformer dateTransformer = new DateTransformer("dd.MM.yy");
    java.util.Date date = (java.util.Date) dateTransformer.transform(dateString,
        IDataTransformer.TYPE_DATE);
    return date;
}
```

Detecting the Current Platform

At runtime, it is often required to get information on the platform on which the application is running. This information may be used, for example, to return a color value that depends on the current platform.

The `com.softwareag.mobile.runtime.toolkit.AbstractApplicationController` contains public methods for determining the current platform:

```
boolean runningOnAndroid();
boolean runningOnIOS();
boolean runningOnWinPhone();
boolean runningOnWin8();
```

These methods can also be used if you want to simulate your application using Phoney. The return value then depends on the activated handset.

See the following snippet for an example:

```
import com.softwareag.mobile.runtime.toolkit.AbstractApplicationController;
public int getColorValue() {
    int colorValue = 0xff000000;
    if (AbstractApplicationController.getInstance().runningOnAndroid()) {
        colorValue = 0xff00ff00;
    }
    return colorValue;
}
```

Using the Context Key Store to Store and Retrieve Application-wide Settings

Mobile Development follows the Model-View-Controller architecture which splits user-written business logic from the generated sources. A frequently-asked question is how to take over a set of data from one view into another. The `ContextKeyStore` offers a simple way to achieve this goal from the application model and also programmatically.

When you use the application model, the input fields (**Entry** and **SearchEntry**) populate the property **Context Key**, which requires a String value as a unique name. Setting up **Entry** or **SearchEntry** elements with a context key will generate a logic which automatically saves the entered value to the `ContextKeyStore`. To identify the data, the specified context key is taken into account as a unique key. In addition to storing the data, any already contained value will be automatically applied to the UI element. All entered context keys will be generated to the `ContextKeys.java` interface. The context key can be embedded in any text property using the `{@CONTEXT_myContextKey}` syntax. This token will be automatically resolved and replaced with the actual value.

Using a programmatic approach, you need to put a key value pair into the `ContextKeyStore`, as described by the following snippet:

```
import com.softwareag.mobile.runtime.toolkit.util.ContextKeyStore;
ContextKeyStore.set("MyKey", "MyValue");
String myValue = (String) ContextKeyStore.get("MyKey");
```

Supported values are all primitive types including String. To store a date value, you need to put the corresponding long value into the `ContextKeyStore`, using the `java.lang` wrapper classes for primitive types:

```
final Date date = new Date();
final long dateAsLong = myDataObject.getTime();
ContextKeyStore.set("MyDate", new Long(dateAsLong));
```

The values contained in the `ContextKeyStore` can be persisted. For this, the application property **Context Persistence Mode** must be set to **Permanent**. All values and their keys are then placed in a `RecordStore` when you close the application. They are automatically loaded when you start the application again.

Encoding and Decoding Images with Base64

When using RESTful services or web services, it is often required to send binary data (such as images) encoded as `String`. This can be achieved by using several API classes, as described in the following snippet:

```
import org.apache.axis.j2me.rpc.Base64;
import javax.microedition.lcdui.Image;
import com.softwareag.mobile.runtime.media.ImageHandler;
public String encodeImage(final Image image) {
    String encodedImage = null;
    if (image != null) {
        final byte[] bytes = ImageHandler.pngCreateByteArray(image);
        final byte[] encodedBytes = Base64.encode(bytes);
        encodedImage = new String(encodedBytes);
    }
    return encodedImage;
}
public Image decodeImage(final String encodedImage) {
    Image image = null;
    if (!AppUtility.instance.isEmpty(encodedImage)) {
        final byte[] decodedBytes = Base64.decode(encodedImage.getBytes());
        if ((decodedBytes != null) & (decodedBytes.length > 0)) {
            image = Image.createImage(decodedBytes, 0, decodedBytes.length);
        }
    }
    return image;
}
```

Configuring a Session Object with Credentials

When using RESTful services, it is often required to send authentication information with the HTTP header. This can be achieved using the **Session** object. A session is used to manage cookies and user credentials across different HTTP requests. Only one session exists at a time.

Use the **Session** object to add credentials to be automatically applied to any RESTful service.

```
import com.softwareag.mobile.runtime.toolkit.rest.Credentials;
import com.softwareag.mobile.runtime.toolkit.rest.Session;
final Session session = Session.get();
final Credentials credentials = new Credentials("user", "password");
credentials.addHostname("hostname");
credentials.addProtocol("http");
credentials.addRealm("realm");
session.addCredentials(credentials);
```

Setting credentials this way will ensure that each request uses these credentials. By default, credentials will be added to the request header as HTTP basic authentication.

It is also possible to delete a session. Be aware that all credentials and cookies set before destroying will be lost. If there is currently no session, executing a request will create one.

```
final Session session = Session.get();
session.destroy();
```

Creating Your Own Operation

Larger operations, such as database access or remote communication, often require a lot of time and must therefore be performed in a thread different from the main thread so that the responsiveness of the user interface is not blocked. Mobile Development provides the `AbstractThreadedOperation` class, which can be used to create asynchronous, thread-safe operations. This class creates its own thread once it is executed and provides thread-safe notifications. The following snippet shows how to create a subclass:

```
import com.softwareag.mobile.runtime.toolkit.operations.AbstractThreadedOperation;
public class MyOperation extends AbstractThreadedOperation {
    protected void runInternal() throws Exception {
        // put code here, let's have exceptions being thrown
    }
}
```

To extend `AbstractThreadedOperation`, you need to put logic into `runInternal()`. This method is executed after `AbstractThreadedOperation#execute()` has been called. Because `AbstractThreadedOperation` inherits from `IOperation`, it is possible to attach multiple `IOperationDelegate` classes for being notified of the operation's state. The following snippet shows how to start the operation and how to add listeners:

```
final MyOperation operation = new MyOperation();
operation.addOperationDelegate(new IOperationDelegate() {
    public void onOperationFinished(final IOperation operation) {
    }
    public void onOperationFailed(final IOperation operation) {
    }
});
operation.execute();
```

It is recommended to add the delegates before executing the operation. The `IOperationDelegate#onOperationFinished(IOperation)` method is called after `runInternal()` has completed without any exception. If an exception is thrown, `onOperationFailed` is called. The result of the operation can be set using the protected member `AbstractThreadedOperation#result`.

Chaining Multiple Operations

Let us assume that you need to call an operation `GetID` to get a particular ID of an object and that you need to call `GetDetails` afterwards to get more information about this object. In this case, you need to chain `GetID` and `GetDetails`, and you have to keep error and

result handling in mind. To achieve this goal, it is recommended that you handle one particular result code for an operation with one `IOperationDelegate`. In our case, you need to introduce three classes which implement `IOperationDelegate`:

```
public class ErrorDelegate implements IOperationDelegate {
    public void onOperationFailed(final IOperation operation) {
        new MessageDialog(ApplicationContext.getInstance()).open();
    }
    public void onOperationFinished(final IOperation operation) {
        if (operation.getResultCode() != HttpURLConnection.HTTP_OK) {
            onOperationFailed(operation);
        }
    }
}
public class GetIDResultDelegate extends ErrorDelegate {
    public void onOperationFinished(final IOperation operation) {
        super.onOperationFinished(operation);
        if (operation.getResultCode() == HttpURLConnection.HTTP_OK) {
            final GetDetails getDetailsOperation = new GetDetails();
            getDetailsOperation
                .addOperationDelegate(
                    new GetDetailsResultDelegate());
            getDetailsOperation.execute();
        }
    }
}
public class GetDetailsResultDelegate extends ErrorDelegate {
    public void onOperationFinished(final IOperation operation) {
        super.onOperationFinished(operation);
        if (operation.getResultCode() == HttpURLConnection.HTTP_OK) {
            final String details = operation.getResult();
            // do something
        }
    }
}
```

Extending from `ErrorDelegate` allows for implicit error handling. If an operation requires error handling that differs from the standard behavior in `ErrorDelegate`, it is also possible to use different `OperationDelegates`.

For complex scenarios, see "[Getting the Current GPS Position and Translating it into a Human-readable Location](#)" on page 192 and "[Downloading an Image from a Remote Host](#)" on page 193.

Using Multipart Requests

The `Request` object for creating services in the Outline Editor supports `multipart/form-data` content types. A file upload is a typical use case for a multipart request.

The following snippet shows a sample implementation of an Agile Apps photo upload:

```
final Image img = getMyImage();
if (img != null) {
    final UpdatePhotoRequest request = new UpdatePhotoRequest();
    request.addOperationDelegate(this);
    final StringBuffer xmlData = new StringBuffer();
    xmlData.append("<platform><record>\r\n");
    xmlData.append("<photo>photo.png</photo>\r\n");
    xmlData.append("</record></platform>\r\n");
}
```

```

// add two parts to the multipart request
// first part is XML to tell Agile Apps about the data being uploaded
// second part is the image data to be uploaded
request.addBytePart(xmlData.toString().getBytes(), "_xml_data_",
    "application/xml", false);
request.addImageFilePart(img, "photo.png", "photo", "image/png", false);
request.execute();
}

```

As of Mobile Development 9.8, `AbstractMultipartRestOperation` is available, which extends `AbstractRestOperation` for full support of multipart requests. Adding the following parts is currently supported by `AbstractMultipartRestOperation`:

- `addBytePart(...)`
- `addFilePart(...)`
- `addImageFilePart(...)`

When using `addImageFilePart(...)`, only `image/png` content type is supported in Mobile Development 9.8. This converts the given `javax.microedition.lcdui.Image` data automatically to PNG data.

For more information, see the Javadoc for `AbstractMultipartRestOperation`.

Doing Error Handling for Operations

Each operation provides a response code as a protected member. The interpretation of this integer value depends on the operation type. Mobile Development uses this type to differentiate between RESTful operations (which inherit from `AbstractRestOperation`) and customized operations (which inherit from `AbstractThreadedOperation`). See also "[Creating Your Own Operation](#)" on page 189.

An `AbstractRestOperation` is meant as a super class for generated RESTful services. It opens an HTTP connection so that the response code can easily be compared with the HTTP result codes. The interface `HTTPConnection` lists all possible result codes for this kind of operation. It is recommended to use one `IOperationDelegate` for one HTTP result code. For a sample, see "[Chaining Multiple Operations](#)" on page 189.

An `AbstractThreadedOperation` is mostly used to encapsulate a single piece of business logic. Therefore, the response code must be set in its implementation class. Let us assume that you want to use a simple operation as described in "[Creating Your Own Operation](#)" on page 189. The recommended way is to introduce a set of public variables that can also be used to determine the result in an `IOperationDelegate`. If everything works correctly, you can simply set the protected member `responseCode` to `OK`. If an exception occurs, set the `responseCode` to the error and also throw the nested exception. Throwing the exception will force the operation to call `onOperationFailed` for each added `IOperationDelegate`.

```

public class MyOperation extends AbstractThreadedOperation {
    public static int RESULT_OK = 1;
    public static int RESULT_ERROR = 2;
    protected void runInternal() throws Exception {
        try {
            // some logic in here
            responseCode = RESULT_OK;

```

```

    } catch(Exception e) {
        responseCode = RESULT_ERROR;
        throw e;
    }
}
}
}

```

Getting the Current GPS Position and Translating it into a Human-readable Location

Using Mobile Designer, it is possible to get the current GPS position. From a user experience perspective, it is often required to show a readable location to the user. This use case can be seen as complex example of how to chain multiple requests (see also "[Chaining Multiple Operations](#)" on page 189). You first have to get the current GPS location, and after this, you have to ask a common service to translate the coordinates into a readable location.

The first step is to access the GPS coordinates. It is recommended to use an `AbstractThreadedOperation` because this operation may take several seconds.

```

public class GetGPSCoordinates extends AbstractThreadedOperation {
    protected void runInternal() throws Exception {
        final LocationProvider lp = LocationProvider.getInstance(null);
        final Location l = lp.getLocation(20);
        final Coordinates c = l.getQualifiedCoordinates();
        responseCode = 0;
        if (c != null) {
            result = (String.valueOf(c.getLatitude()) + "," + String.valueOf(c
                .getLongitude()).getBytes());
            responseCode = HttpURLConnection.HTTP_OK;
        }
    }
}
}

```

For more information about the `LocationProvider`, see the Mobile Designer documentation.

The second step is to ask a common service provider for a readable location. This can be achieved, for example, using the public Google API:

<http://maps.googleapis.com/maps/api/geocode/json>

To execute this request, create a new RESTful service using the palette, enter the above URL and a name for the service class to be executed from Java. This request also requires a query parameter named `latlng`. In this example, the RESTful service to execute this request is named `GetLocation`. If `GetGPSCoordinates` results in `HttpURLConnection.HTTP_OK`, you can trigger `GetLocation` with the result of the previous operation.

```

final GetLocation getLocation = new GetLocation();
getLocation.execute(operation.getResult());

```

Downloading an Image from a Remote Host

Downloading an image from a server is a typical scenario for using `AbstractThreadedOperation`. First, you have to subclass `AbstractThreadedOperation`. After this, you have to open an `URLConnection`, and the result must be read using a `DataInputStream`.

```
import java.io.ByteArrayOutputStream;
import java.io.DataInputStream;
import java.io.InputStream;
import javax.microedition.io.Connector;
import javax.microedition.io.HttpURLConnection;
import javax.microedition.lcdui.Image;
import com.softwaeag.mobile.runtime.toolkit.operations.AbstractThreadedOperation;
public class ImageLoaderOperation extends AbstractThreadedOperation {
    private final String url;
    private byte[] imageData = new byte[0];
    public ImageLoaderOperation(final String url) {
        this.url = url;
    }
    protected void runInternal() throws Exception {
        HttpURLConnection hpc = null;
        DataInputStream dis = null;
        final ByteArrayOutputStream output = new ByteArrayOutputStream();
        try {
            hpc = (HttpURLConnection) Connector.open(url);
            final byte[] data = new byte[8192];
            int read = 0;
            final InputStream stream = hpc.openInputStream();
            dis = new DataInputStream(stream);
            while ((read = dis.read(data, 0, 8192)) > 0) {
                output.write(data, 0, read);
            }
            imageData = output.toByteArray();
        } finally {
            if(dis != null) {
                dis.close();
            }
            if(output != null) {
                output.close();
            }
        }
    }
    public Image getImage() {
        Image image = null;
        if ((imageData != null) && (imageData.length > 0)) {
            image = Image.createImage(imageData, 0, imageData.length);
        }
        return image;
    }
}
```

Using a Java Class to Communicate with the Natively Injected Code

With Mobile Designer, it is possible to extend the given capabilities, especially those for interacting with the target operation system. This feature can be used to add platform-specific features to a Mobile Project, such as integrating a barcode scanner or displaying PDF files. Adding platform-specific code to a Mobile Project is called "native code injection".

The principle behind native code injection is to use a Java class as a bridge to the native code. The Java class provides a signature and with that an entry point for communicating with the native code. Furthermore, the Java class is replaced during the build process. Because of this, the native equivalent must also implement the signature to avoid any compile issues that may occur during the build process. For technical examples, see the Mobile Designer sample projects NativeUI My Native Element and the NativeUI PDF Demo.

To implement native code injection, proceed as follows:

1. Create a Java class with a common signature (for example, a static method).
2. Do a local build for the desired native platform.
3. Configure the Mobile Project to use native code injection. To do so, implement the `project.hookpoint.target.postcrosscompiler` hookpoint in your `defaults.xml` file and configure which classes need to be replaced. For more information about hookpoints, see *Using webMethods Mobile Designer*.

With Mobile Development, the `project.hookpoint.target.postcrosscompiler` hookpoint is already pre-configured in each Mobile Project. You only need to add the native classes to the `project_name/native/platform` folder and ensure that the folder structure matches the Java package. The platform folders are:

Platform	Folder
Android	<code>project_name/native/android</code>
iOS	<code>project_name/native/ios</code>
Phoney	<code>project_name/native/phoney</code>
Windows	<code>project_name/native/winphone</code>

For example, when you have a Java class named `com.softwareag.mobile.Helper.java`, you have to add the native source code for Android to `project_name/native/android/com/softwareag/mobile/Helper.java`.

4. The local build (see above) cross-compiles the Java class to the target platform, which results in a native source file already translated to the target programming language. You can copy this source file and customize it without having to take care of how to add imports or method declarations in the target programming language.
5. Do a build for the platform once more. The result will contain the natively injected code.

Configure your application to receive Push Notifications

Mobile Suite enables customers to setup the application to receive Push Notifications.

To prepare your Mobile Project, do the following steps:

1. Create or reuse an application on Mobile Administrator
2. Open your application model and locate the **Application** node. For more information, see "[Using Mobile Administrator to Manage and Distribute Mobile Applications](#)" on page 29.
3. In the **Properties** section, check the **Use Push Notifications** property. For more information, see "[Application Node Properties](#)" on page 104.
4. Regenerate the **Java sources**. For more information, see "[Generating Sources for a Mobile Project](#)" on page 33.

Your Mobile Project is now prepared to receive Push Notifications. However, depending on the target platform, there are still some steps required. Your project requires a unique ID representing the device and the application. This ID depends on the target platform and requires different prerequisites.

For Android:

Android applications using Google Messaging Services (GCM) to send Push Notifications. To register your device, you need a unique sender ID. The sender ID is a project number you will get from Android's development console:

1. Open Google's development console at <https://console.developers.google.com> and login with your Google account.
2. Create or use a Project and copy the **Project Number** from the project's overview. It must consist of 12 numbers.
3. Enable **Cloud Messaging for Android** in the **APIs & Auth** section.
4. Create a new API key for public API access.
5. Open your application in Mobile Administrator, select **Manage App** and go to **Push**. There you need to enter the API key as the **GCM API Key**.
6. Open your Mobile Project with the Mobile Development perspective.
7. Open the `targets/_defaults_.xml` file.

8. Add the following properties:

Property	Description
<code><property name="android.push.sender.id" value="yourProjectNumber"/></code>	Required. This property represents the sender ID. It's the project number you'll get from the development console.
<code><property name="android.push.additional.icon" value="resources/icons/myIcon.png"/></code>	Optional. Specify an icon to display for the notification item within the status bar.
<code><property name="android.push.additional.icon.pre21api" value="resources/icons/myIcon.png"/></code>	Optional. Specify an icon to display for the notification item within the status bar. This icon will be used for devices with Android up to API 20.

For iOS:

iOS applications using the Apple Push Notification Service (APNS). You need to setup a SSL Certificate for your App ID and configure a provisioning profile with **Push Notifications** enabled and build/rebuild your application:

1. Open your application in Mobile Administrator, select **Manage App** and go to **Push**.
 - a. Select **Add Certificate (Production)**.
 - b. Create a **Certificate Signing Request (CSR)**.
 - c. Download the **Certificate Signing Request (CSR)** to be used in the following steps.
2. Open the Apple Developer Portal at <https://developer.apple.com/account/ios/>.
 - a. Select Identifiers.
 - b. Create a new App ID with **Push Notifications** enabled.
 - c. Edit the App ID and look for **Push Notifications**. There you can add production and development SSL certificates needed to send Push Notifications via APNS.
 - d. Select **Create Certificate**.
 - e. Continue by providing the CSR file downloaded from Mobile Administrator and Generate a new certificate.
3. In your Mobile Administrator application you can now upload the Certificate for the Certificate Signing Request created earlier.
4. In Apple Developer Portal you can now create or regenerate the Provisioning Profile for your app and download.

5. Add the Provisioning Profile to Mobile Administrator.
6. Edit your iOS build configuration and check the Register for Push Notifications App Policy.
7. Rebuild the application using the new Provisioning Profile. Mobile Administrator enriches the application with code to do the registration to APNS and sends the registration ID back to Mobile Administrator.

For Windows:

No additional steps are required to prepare a Windows application.

Once your registration to the platforms Push Notification service is completed, you need to publish the registration ID to Mobile Administrator. This work is required for Android and Windows applications. iOS applications will automatically upload the registration ID to Mobile Administrator, if the **Register for Push Notification App Policy** is checked in your build configuration. To upload the registration ID to Mobile Administrator, open your **Application Controller** in Software AG Designer and overwrite the `onDeviceRegistered()`. For more information see ["Common Methods to Override in the Generated Code for the Application" on page 175](#). Have a look at the following snippet:

```
public void onDeviceRegistered(final String newPushUID) {
    final PostPushRegister request = new
    PostPushRegister(Parameters.PARAM_MOBILE_ADMIN_IDENTIFIER);
    if (runningOnAndroid()) {
        request.execute((String) null,
            Parameters.PARAM_MOBILE_ADMIN_IDENTIFIER, newPushUID,
            null, null);
    } else if (runningOnWin8() || runningOnWinPhone()) {
        request.execute((String) null,
            Parameters.PARAM_MOBILE_ADMIN_IDENTIFIER, null, null,
            newPushUID);
    }
}
```

The snippet uses the built-in **PostPushRegister** service. Have a look at ["Adding Mobile Administrator Services" on page 67](#) for more details. After uploading the registration ID to Mobile Administrator, you can send a Push Notification using Mobile Administrator. Open your application in Mobile Administrator, select **Manage App** and go to **Push**. Then type a text and trigger **Send Push Notification**. The message will then be forwarded as a Push Notification for all registered devices.

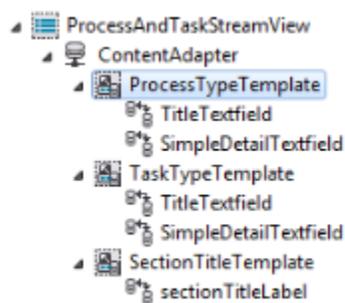
Configure a Content Adapter to use multiple Templates

In a **Content Adapter** you can declare to use multiple Templates. This is useful if the selected Datasource contains different types of elements. Each type can then use a dedicated Template to display its data.

To add multiple Templates to a Content Adapter

1. Ensure the Mobile Project is open in the Outline Editor. For instructions, see ["Displaying a Mobile Project in the Outline Editor" on page 17](#).

2. In the **Model** section of the Outline Editor, expand the project so that you view the **Content Adapter** where you want to add multiple Templates.
3. Right-click and select **New Child > TemplateAssignment**. Select the Template you want to use from the Dropdown list. If you need create a new Template, please refer to ["Creating a Template for a Custom Object" on page 98](#).
4. Enter the Filter Expression for this type of element. This String is required later on to identify the different types of elements from the Datasource. The String should be unique across all **TemplatesAssignments** within that **Content Adapter**.
5. Repeat steps 3 and 4 for all different types of elements that you want to display. See below for an example of a model using 3 different Templates.



6. Open the Java Source file for the Datasource that you defined in the **Content Adapter**.
7. Overwrite the method `public boolean matchesExpression(String filterExpression, int elementIndex)`.
8. This method is called for each element of the Datasource to identify which Template should be used for displaying it.
 - a. The parameter **filterExpression** is one of the Filter Expression that you just defined in the TemplateAssignment.
 - b. The parameter **elementIndex** defines the index of the element in the Datasource. You can use the methods `getEntityForIndex(int)` (in an EntityDatasource) or `getElementAtIndex(int)` (in a ListDatasource) to get the element at this position.
9. In the given example we could implement this method like this:

```
public boolean matchesExpression(final String filterExpr, final int elementIndex) {
    final IEntity entity = getEntityForIndex(elementIndex);
    if ((entity instanceof TaskTypeImpl) && "task".equalsIgnoreCase(filterExpr)) {
        return true;
    } else if ((entity instanceof ProcessTypeImpl)
        && "process".equalsIgnoreCase(filterExpr)) {
        return true;
    } else if ((entity instanceof SectionTitleImpl)
        && "section".equalsIgnoreCase(filterExpr)) {
        return true;
    }
    return false;
}
```